Technology
Arts Sciences
TH Köln

Master's Thesis

Author:	Kristan Böttjer
Matrikelnummer:	11422684
Degree Programme:	Digital Sciences (M.Sc.)
Specialisation:	Data and Information Science
In Cooperation with:	SCOUTASTIC GmbH
First Supervisor:	Prof. Dr. Richard Sieg
Second Supervisor:	Dr. Mehmet Hakan Akdag

Title

Implementation of an Agentic Chat Interface for a Football Scouting Platform using Retrieval-Augmented Generation and Natural Language Database Querying

Submitted to TH Köln – University of Applied Sciences, Campus Südstadt · Faculty of Information and Communication Sciences

Cologne, October 2026

Abstract

Football clubs hold what they know about players as structured data, such as a player's age, position and the grades in rating fields, and as unstructured data in the text of their scouting reports. A scout's question often needs both. This thesis designs a chat system that answers such questions in natural language, builds it for the scouting platform of SCOUTASTIC and evaluates it on a test database that joins the platform's football data from production, club data such as watchlists made for the test, and 486 scouting reports written by one club's scouts. The system is a routing workflow, in which a router divides a question into facts and traits. A text-to-query agent answers the facts with queries in MongoDB's query language. It takes over steps that added accuracy inside fixed workflows, such as reading the stored values before it answers, and keeps the freedom to choose its next step. A trait search, a fixed entity retrieval procedure, ranks the players by what their reports and rating fields say about each trait. Since the rating fields record part of what the text of the reports says about the same quality, it reads the two as two views of one trait. Its judge, a language model, reads for every retrieved passage whether the passage speaks for or against the trait, and evidence against an asked trait removes the player from the ranking. To evaluate the system, a test collection of 154 questions in English and German was built, 43 on facts, 65 on traits and 46 on both. Each fact question carries a gold query written by hand, against whose result the agent's answer is scored. For the trait questions, a language model labelled every report for each of 33 traits with the direction and strength of what it says. The relevant players of a question follow from these labels by a stated rule, so the players that the trait search misses can be counted, and a player whose reports speak against an asked trait counts as harm. The router sent every fact and trait question down its route, and behind it both components scored about as well as behind an oracle router that divides every question as the gold does. The agent answered 89.8 % of the fact questions correctly on GPT-6 Sol and 82.0 % on the smaller GPT-6 Luna, and most of its wrong answers arose in interpreting the question and the data rather than in writing the query. Neither leaving out its worked examples, choosing them per question nor preloading every schema changed its success measurably at the size of this database, so cost and time decide between them. The trait search ranked the relevant players at an nDCG@10 of 0.78, against 0.22 for a random ranking, and its judge halved the number of players among the first ten whose reports speak against an asked trait.

Contents

1 Introduction **1**

2 Theoretical Foundation **4**
2.1 Large Language Models, from Prompting to Context Engineering . . . 4
2.2 Natural Language Database Querying . . . 8
2.3 Information Retrieval . . . 14
2.4 Hybrid Systems over Structured and Unstructured Data . . . 21
2.5 Open Questions from the Literature . . . 24

3 System Design **26**
3.1 Requirements and Overview . . . 26
3.2 Router and Dispatch . . . 27
3.2.1 Facts and Traits . . . 27
3.2.2 Routing . . . 28
3.2.3 Dispatch . . . 29
3.3 MQL Sub-Agent . . . 29
3.3.1 Planning Approach . . . 30
3.3.2 Agent Loop . . . 30
3.3.3 Context Composition . . . 33
3.3.4 Deterministic Safeguards . . . 35
3.4 Trait Search . . . 35
3.4.1 Design History . . . 36
3.4.2 Text Evidence . . . 37
3.4.3 Rating Evidence . . . 39
3.4.4 Evidence Combination . . . 40
3.5 Answer Synthesis . . . 42

4 Implementation **43**
4.1 Environment . . . 43
4.2 Stack . . . 43
4.3 Components . . . 45
4.3.1 Router and Dispatch . . . 45
4.3.2 MQL Sub-Agent . . . 46
4.3.3 Trait Search . . . 48
4.3.4 Answer Synthesis . . . 50

5 Evaluation **51**
5.1 Evaluation Setup . . . 51
5.1.1 Question Set . . . 51
5.1.2 Frozen Database and Runs . . . 52
5.1.3 Hypotheses and Tests . . . 53
5.2 MQL Sub-Agent . . . 56
5.2.1 Fact Questions . . . 56
5.2.2 Measures and Tags . . . 57
5.2.3 Hypotheses . . . 58

5.2.4 Results . . . 59
5.3 Trait Search . . . 62
5.3.1 Relevance Without Pooling . . . 62
5.3.2 Trait Vocabulary and Corpus . . . 64
5.3.3 Labelling . . . 65
5.3.4 Relevant Players . . . 68
5.3.5 Measures and Populations . . . 68
5.3.6 Hypotheses . . . 70
5.3.7 Results . . . 71
5.4 Router . . . 75
5.4.1 Route and Division Gold . . . 75
5.4.2 Measures . . . 75
5.4.3 Hypothesis . . . 76
5.4.4 Results . . . 76

6 Discussion **79**
6.1 MQL Sub-Agent . . . 79
6.2 Trait Search . . . 81
6.3 Router . . . 83
6.4 Model Choice per Component . . . 83
6.5 Limitations . . . 84

7 Conclusion and Future Work **86**
7.1 Research Questions and Contributions . . . 86
7.2 Recommendations . . . 87
7.3 Future Work . . . 88

Bibliography **90**

A Model Capability Reference **102**

B Prompts **103**

C Construction of the Rating Fields **106**

D Statistics of the Test Collection **108**

E Further Analyses of the Trait Search **114**

F Questions of the Test Collection **117**

G Accepted Second Answers **122**

H Wording Differences Between the English and the German Questions **124**

I Test Results of the Hypotheses **126**

J Glossary **127**

Use of AI Tools **129**

1 Introduction

Professional football clubs compete to find good players before other clubs do, and to find them, their scouts watch thousands of players a season and record what they see in written reports. As a result, much of what a club knows about a player lies in the text of these reports, while facts such as a player's age, position or market value are stored in a database. A scout's question often needs both, as in a search for left-footed defenders under 23 whom the club's own scouts describe as calm under pressure. Language models now allow such a question to be asked in natural language, but a system that answers it has to read the database and the reports together.

The setting of this thesis is the scouting platform of SCOUTASTIC GmbH, on which professional clubs organise their scouting. The platform holds football data from two sources. The first is data on players, clubs, competitions and matches, which states its facts in fixed fields, such as a player's position or the result of a match. The platform takes this data from the football data provider Transfermarkt, with which SCOUTASTIC is a joint venture. The second is the data each club creates when it organises its scouting, its own entries such as watchlists, tags and the status of a player in its scouting, and above all the reports of its scouts, which describe in free text what the scouts saw and can carry rating fields that the club defines. A report refers to its player, and through this link what scouts write can be combined with the facts stored about the player. Transfermarkt's data, the club's own entries and the rating fields are structured data, which a database query can search, whereas the text of the reports is unstructured and has to be searched by retrieval. Structured data and unstructured text therefore call for different techniques.

For a club, the reports are the most valuable part of its data, because no other club can buy them. Transfermarkt's data is sold to every club that pays for it, as is analytical data, such as tracking data. Since the better such data is, the more clubs buy it, it gives no club a lasting advantage over another. A club's reports, by contrast, are written by its own scouts and stay within the club. They hold what the club knows about a player and other clubs do not, and finding a player before other clubs do rests on exactly that knowledge.

The rating fields exist to make the assessments in the reports queryable. In a rating field the scout records an assessment as a value, such as a grade for heading, and the club can then query its players by it. These fields, however, capture only what the club expected to need for every player, since the club has to define them before the reports are written. No field holds an observation that the club did not foresee or that fits only a few players, such as a winger's habit of cutting inside onto the stronger foot, nor the reason why a scout gave a grade, and such detail therefore appears only in the text of the report.

Rating fields also lose their value over time, because a club changes them. A club defines its fields to match the profiles of the players it looks for and the way its scouts assess players, and in SCOUTASTIC's experience both change often, for instance with a new head of scouting. The fields then change as well, which is why clubs are often reluctant to define many rating fields. Reports written before such a change lack the new fields, and the players they describe consequently cannot be compared on these fields with the players rated since. Nor can the old reports be rated again on the new fields, because what the scout observed at the time survives only in the text the scout wrote. Of all that a report holds, the text thus keeps its value longest,

yet the platform's built-in player search cannot search these texts, since it lets a user filter players only by the values of stored fields.

The filters also limit how the structured data can be searched, in two ways. (1) A search with filters has to be learned before it can be used. The platform holds a large amount of data and serves many kinds of question, and its search therefore offers many filters, which take practice to choose and combine. A question in natural language, by contrast, needs no such practice, since any user can state in their own words what they want. (2) A search with filters offers only the conditions that were built into it, and a new kind of condition thus needs a new filter and with it a change of the interface.

A chat system in which the scouts and staff of a club ask their questions in natural language can make the report texts searchable and, at the same time, free the structured data from the limits of the filters. For the report texts, the system retrieves what scouts wrote about a player, so that a question can ask about the text as it asks about the structured data. For the structured data, a question takes the place of the filters, and a user no longer has to learn which filters exist but can state any condition that the data can answer, such as "players who scored more goals this season than last season".

This thesis designs such a chat system from the literature, builds it for the SCOUTASTIC platform and evaluates it. The questions it answers are of three kinds. A question can ask (1) only for what the structured data states, as in "left-footed centre-backs under 23 from the Bundesliga", (2) only for a quality that scouts assess in their reports, as in "players with an explosive first step", or (3) for both, as in "left-footed centre-backs under 23 from the Bundesliga with an explosive first step". The system therefore has two legs, one for each kind of data. The structured leg is an agent that translates a question into queries over the club's database in MongoDB, while the unstructured leg retrieves what the reports say about a quality and ranks players by it. A question that needs both kinds of data is divided between the two legs, and the system combines their results into one answer.

Systems that answer a question from structured and unstructured data together exist, but none was built for data like a club's. Many earlier chat systems sent each question to a single source and could therefore not answer a question that needs both (Liu et al., 2024b). The systems that combine both decide for each question or each part of it which source holds the answer (Liu et al., 2024b; Xia et al., 2025; Bai et al., 2026), whereas scouts record the same quality both in the text of a report and in its rating fields. These systems were also built over relational databases, while a club's data lies in MongoDB, whose query language MQL got its first benchmark only in 2024 (Sivasubramaniam et al., 2024). How the component that writes the queries should be built is open as well, since on a benchmark of large databases and on one of schema-less document databases, a fixed workflow, whose steps are set in code, answered more questions than the free agent it was compared with, which chooses each next step itself (Deng et al., 2025; Lu et al., 2026). For such an agent, practice is also ahead of research, since context engineering, the design of what the agent is given at each step, follows mostly the recommendations of practitioners (Rajasekaran et al., 2025), which the research literature has rarely tested. On the side of the reports, a retriever scores how much a report is about a quality, not whether it affirms or denies it, and neural retrievers barely tell a statement from its negation (Weller et al., 2024). A report against a quality can therefore raise a player's score. Finally, no test collection exists for questions over scouting reports, and whether such a system answers them well can therefore not yet be measured.

These gaps lead to the central research question of this thesis.

> *How should an agentic chat system that answers questions from a club's structured data and from the text of its scouting reports be built and its context engineered, and how can it be evaluated?*

It is divided into three sub-questions.

- **RQ1, Test Collection:** Which properties does a test collection need for scoring the division of a question between the two legs, the queries of the structured leg and the ranking of the unstructured leg, and how can it be built from a club's own reports?
- **RQ2, Architecture:** Which architecture, drawn from the literature, lets a system answer questions in natural language that need a club's structured data, the text of its reports or both, and how should its component that writes database queries be built?
- **RQ3, Context Engineering:** For a selection of the recommendations on context engineering from research and practice, which hold for an agent that queries a club's database?

The thesis makes four contributions. (1) It builds a test collection of 154 questions in the scouting domain. Each question carries the gold for every component it reaches: the division of the question, a query written by hand for the structured leg and the relevant players for the unstructured leg. The relevant players are derived from labels of every report, and each label records whether the report speaks for or against a quality. (2) Unlike the earlier systems that combine structured and unstructured data, it uses the text of the reports and the rating fields as evidence for the same quality. In a club's data the rating fields answer part of what the text of the reports answers, but not all of it, so the unstructured leg also reads the rating fields. (3) It proposes an alternative to choosing between a fixed workflow and a free agent for writing database queries. Its agent takes over steps that added accuracy inside the fixed workflows that beat an agent, such as reading the stored values before it writes the query, leaves out their vote over several candidates and keeps the freedom to choose its next step. (4) It brings the direction of the evidence, for or against a quality, into entity retrieval, which ranks entities by the documents written about them. Earlier work recorded this direction for documents or ranked them by it (Ounis et al., 2006, 2008), or averaged it into an entity's relevance grade (Ganesan and Zhai, 2012). The system reads for every passage whether it speaks for or against the quality asked for, and the evaluation counts a player whom the reports speak against as harm rather than as a result that is merely not relevant.

Chapter 2 reviews the literature on large language models, on querying databases in natural language, on information retrieval and on systems that combine structured and unstructured data, in the order in which the system uses them. Chapter 3 describes the design of the system and Chapter 4 its implementation. Chapter 5 builds the test collection and evaluates the components on it. Chapter 6 discusses the results, and Chapter 7 concludes with the answers to the research questions and with future work.

2 Theoretical Foundation

The system described in Chapter 3 rests on four bodies of prior work, and this chapter covers them in the order in which the system uses them. The first is large language models (LLMs) and how a model is adapted through what it is given in a call, from prompt engineering to context engineering (Section 2.1). Every stage of the system is a call to a hosted LLM, so the model is its backbone and comes first. The second is natural-language interfaces to databases (Section 2.2), the basis of the structured leg, which translates a question into a query over the scouting database. The third is information retrieval (Section 2.3), the basis of the unstructured leg, which ranks players by the content of their reports, and of the evaluation of that leg. The fourth is systems that combine structured and unstructured data (Section 2.4), the basis of the combination of the two legs. Section 2.5 relates the findings of the four sections to each other and lists the recommendations on context engineering. Each section proceeds chronologically, from the work that defined the problem to the current state, and ends with what this thesis adopts and what it leaves aside.

2.1 Large Language Models, from Prompting to Context Engineering

A language model is adapted to a task either by training its weights or through what it is given when it is called. The context of a call is everything the model is given, the instruction, the definitions of the tools it may call, the conversation so far, the results of earlier tool calls and retrieved passages, and it has to fit into the model's context window. Until 2020 a model was adapted to a task through its weights. Since then the adaptation has moved into the call, first into the instruction and its examples, then into everything else the window holds. This section covers that move in seven periods, from the prompt that replaced training to the context of an agent.

The task moves into the prompt. Every model in this section is a transformer, the architecture of Vaswani et al. (2017), which computes over all positions of a sequence in parallel and is what made training at the sizes below affordable. Until 2020 each task had its own model on top of it, with its own output layer and its own training objective, trained on labelled examples of that task. Raffel et al. (2020) wrote every task as text in and text out, with the task named in a prefix of the input, so that one model with one training objective could be fine-tuned for any task. Given four times the compute, a larger model gained more than training longer. Their model, T5, is the one the text-to-SQL systems of the following years fine-tuned.

Brown et al. (2020) then showed with GPT-3, a model of 175 billion parameters, that a task can also be given in the prompt, as an instruction and examples, with no gradient update. On many tasks this came close to fine-tuned systems and on a few it surpassed them, while on some tasks that compare two sentences it did little better than chance. The gap between few-shot and zero-shot accuracy often grew with model size, which the authors read as a sign that larger models learn better from examples in the prompt. From then on the prompt carried the task, and examples in it were a lever that the database querying of the following years measured again (Section 2.2).

Code is a language and execution is the test. A programming language is a language, so a model that generates text can generate code, and translating a question into a query is the capability the structured leg of this thesis rests on. Chen et al. (2021) trained a GPT on code and,

with it, changed how a generated program is judged. A translation between languages is scored by its similarity to a reference, but a program cannot be, since it is correct when it does what was asked and not when it resembles a solution. Their benchmark, HumanEval, therefore scores a program by running it against unit tests. Their model, Codex, solved 28.8% of its problems with one sample, and with 100 samples at least one was correct for 72.3%. When it generated 100 samples and picked the one it was most confident about, it solved 44.5%. Without training on the task, Codex later wrote SQL about as well as a fine-tuned system of 2020 (Rajkumar et al., 2022).

Reasoning in the prompt, and prompt engineering. Wei et al. (2022) showed that a model reasons better when its prompt shows reasoning, a technique they called chain of thought. Eight hand-written exemplars whose answers spell out their intermediate steps, and no fine-tuning, took PaLM 540B from 17.9% to 56.9% on GSM8K. Their ablations showed that the gain comes from natural-language steps written before the answer. Wang et al. (2023) added sampling, which they called self-consistency. Several reasoning paths, sampled from the same prompt and resolved by majority vote, raised GSM8K by a further 17.9 percentage points, and accuracy rose with the number of paths with diminishing returns. Every path is a full call, so the cost of sampling is linear in the number of paths.

Reasoning was one of many kinds of prompting techniques, beside examples, instructions, decomposition into sub-tasks and self-criticism, and there were enough of them to be surveyed. Schulhoff et al. (2025) call the design of the instruction and its examples prompt engineering, and their survey adds a case study in which an expert engineered a prompt for a labelling task over dozens of measured steps. More examples did not reliably help, since twenty scored below ten and adding one lowered the score. A simple search that kept the instruction fixed, drew many sets of labelled examples at random and kept the set that scored best ended above the best of the expert's prompts, and the searched prompt after a slight revision by a person scored best of all. The authors conclude that a model reacts strongly to small details of its prompt, that the search and a person work best together, and that prompt engineers should work with experts of the domain.

The model acts. The reasoning described so far works only with what a model has learned. A chain of thought can therefore invent facts, since none of its steps is checked against the world. A model could also be prompted to act on an environment, but it then chose each action without reasoning about it and could not plan or revise a strategy. Reasoning and acting had been studied as separate lines of work (Yao et al., 2023), and Yao et al. interleaved them in a loop they called ReAct. In it a reasoning step is followed by an action, the observation of the action brings knowledge from outside the model into the context, and the next reasoning step starts from it. On ALFWorld this took success from 45% for the same loop without the reasoning steps to 71%, and the trajectory can be read, step by step, to see where a run went wrong. Two things follow from the loop. (1) The context grows with every step, since each result is added to it, so for every agent it has to be decided which results enter the context and which of them stay there for later steps. (2) The loop became the standard form of an agent, and the database agents of Section 2.2 and the retrieval agents of Section 2.4 build on it.

In the agents that followed, an action is a tool call, which a program executes. The output of a model is text, while a call consists of fixed words, the name of one of the tools and the fields it takes. A model chooses each token by its probability among all tokens of its vocabulary, so a valid name or field is likely but not guaranteed. The more fields and rules a call has to obey, the more places it offers for a slip, such as a field name that does not exist or a required field that is left out. Constrained decoding rules such slips out by allowing, at each token, only the tokens that can lead to an allowed continuation (Willard and Louf, 2023).

Reasoning moves into the model. By 2023 the commercial AI labs had stopped saying how their models were built. The report on GPT-4 (OpenAI, 2023) states that it discloses neither the size nor the architecture nor the training data of the model and reports its benchmark and exam results instead, and from that point on the capability of a hosted model is known only by measuring it. How a model learns to reason is therefore known from open models, whose weights and training recipes are published, such as DeepSeek-R1 (Guo et al., 2025). Guo et al. trained DeepSeek-R1-Zero by reinforcement learning without any reasoning written by people, neither as examples nor as data for fine-tuning, and rewarded only a correct final answer and a fixed output format. Its responses grew longer through training and began to verify and revise themselves. The released DeepSeek-R1 adds a few thousand written examples of readable reasoning before the same training and keeps this behaviour. So DeepSeek-R1 internalised chain of thought and part of self-criticism, two techniques a prompt had to elicit before, and it sets the length of its reasoning per problem. Sampling several answers and voting stayed outside the model, and the authors note that it complements the long reasoning, since a majority vote over 64 sampled answers still raised DeepSeek-R1 on the mathematics competition benchmark AIME 2024 from 79.8% to 86.7%, at the cost of 64 calls instead of one.

Whether a model reasons, and for how long, then became a parameter of the call. The API of the GPT-5 family, for instance, takes a reasoning effort with every call (OpenAI, 2026a). So the written reasoning and its length, which a prompt controlled in 2022, are set per call today and paid for in the tokens the reasoning consumes.

The window is not used evenly. In the same years, retrieved passages and the steps of a loop made the context of a call long, and the models of that time often used the middle of a long window least. Liu et al. (2024a) gave models a question with 20 documents, one of which contained the answer while the others were search results without it, and moved that document through the positions. Accuracy was highest with the answer at the start or at the end of the window and lowest in the middle, and with the answer in the middle GPT-3.5 answered fewer questions correctly than the 56.1% it answered with no documents at all. Hsieh et al. (2024b) wanted to know whether the models fail to recognise the document in the middle or only attend to it less. They found that models put more attention on the first and the last documents whatever these contain. Their method "Found in the Middle" subtracts that bias from the attention, and with it two open 7B models of 2023 answered 6 to 15 percentage points more of the questions correctly with the relevant document in the middle. So the loss in the middle is tied to a bias in the attention of the model, which reordering the context works around and calibrating the attention reduces.

The part of the window that a model uses well is shorter than the part it accepts, and length alone costs accuracy. Hsieh et al. (2024a) asked whether the context window of a model scales as far as the model claims. On their synthetic benchmark RULER, almost all models lost accuracy as the input grew, and of 17 models that claim at least 32,000 tokens only about half still kept a satisfactory score at that length. Levy et al. (2024) kept the task fixed and lengthened only the input. They padded the reasoning questions of their dataset FLenQA with added text up to about 3,000 tokens, and accuracy fell already there, far below the windows the models accept. It fell with every kind of padding and for every model, even when the padding only repeated the relevant paragraphs, so on these tasks length alone cost accuracy.

Which content fills the window matters as well, and in question answering over retrieved passages, those related to the question that do not contain the answer do the most harm. Cuconasu et al. (2024) placed the passage that answers a question in the window of a question-answering prompt and varied what surrounded it. Passages related to the question that do not contain the answer lowered accuracy most. Random passages from unrelated texts lowered it less or, for some models, raised it, and the authors conclude that such noise can help but leave the reason open. In the tests of Levy et al. (2024), by contrast, unrelated text from books

lowered accuracy more than padding that resembled the task, so which content harms most differed between the two setups.

The loss at length is largest when the question and the evidence are worded differently. On the benchmark NoLiMa (Modarressi et al., 2025), in which the two share almost no words, 11 of 13 models fell to half or less of their short-input accuracy at 32,000 tokens, and on Llama 3.3 70B a longer input lowered accuracy at every position of the evidence for questions that need two steps of inference. Context Rot, a technical report of the vendor Chroma and thus grey literature, repeated such tests on 18 models up to those of mid-2025 (Hong et al., 2025). Accuracy fell with length even on simple tasks, faster the less the evidence resembled the question and further with distractors related to the question, while in its task of finding one hidden sentence the position of that sentence made no notable difference. On questions over long conversations every model answered better from the relevant turns alone, about 300 tokens, than from the whole conversation of about 113,000 tokens. So by 2025 the question had widened from where content stands in the window to which content enters it and how much.

Context engineering for agents. In an agent that question is asked at every step, since the loop of ReAct adds each action and its observation to the context of the next step, so the context is the agent's state. Anthropic calls the design of that state context engineering, the choice of what enters the window of each call and how much of it, across the whole run of an agent (Rajasekaran et al., 2025). Anthropic presents it as the natural progression of prompt engineering.

For agents the practice of context engineering ran ahead of the evidence, and most of it is reported by vendors. Anthropic describes the context as a limited budget of attention and recommends four techniques, the first for every agent and the other three for agents that run long. (1) In just-in-time retrieval an agent keeps identifiers such as file paths and loads the data only when a step needs it, instead of loading everything that might matter into the context in advance. (2) In compaction a conversation that nears the limit of the window is summarised and continued from the summary. (3) In structured note-taking the agent writes notes outside the window and reads them back later. (4) A sub-task goes to a sub-agent with a clean window, which returns a short summary. The start-up Manus reports from four rebuilds of its agent that the context should grow only by appending, since a change to an earlier part invalidates the cache from that point on, which raises cost and latency (Ji, 2025). Manus also keeps failed actions in the context, since in its experience the model is then less likely to repeat them. Both sources are blog posts. They come from teams that build agents in practice, but they report experience rather than scientific tests, and neither compares a technique with the same agent without it. Later studies tested two of these techniques. For just-in-time retrieval, Chaturvedi et al. (2026) found in single runs on six tasks that keeping tool outputs outside the window behind placeholders the agent can restore raised the solve rate of a coding agent with three of five models, while with the other two the solve rate stayed the same and the token use rose. For sub-agents, Sun et al. (2026) found that an agent which hands sub-tasks to branches that return only their result keeps its accuracy with a far shorter window.

Where a model compressed an agent's context by summarising or rewriting it, the agent was not more accurate and the run cost more. Wang et al. (2025b) varied one part of an agent at a time on a benchmark of assistant tasks and compared six kinds of memory for the steps of a run. Keeping the raw actions and observations was both the most accurate and the cheapest, and a memory summarised by the model cost the most without being more accurate, although the authors rate the effect of memory on accuracy as small. Zhang et al. (2026) wanted contexts that improve with use, and they observed that a model asked to rewrite a context as a whole shortens it until it loses what it held. After one such rewrite on a benchmark of agent tasks, the model answered less accurately with the rewritten context than with no context at all. Their method, Agentic Context Engineering (ACE), therefore changes the context by small added

or edited entries instead of rewriting it. So the evidence that exists speaks against having the model rewrite what an agent has seen while it still fits the window, and compaction, which the practice recommends near the limit of the window, was not measured.

What this leaves for a system on a hosted model. A hosted model is adapted through its context, since its weights are not accessible. Most stages of the system in Chapter 3 choose among a fixed set of answers, a route, a label, a score. For the instruction of such a stage, examples in the prompt raised accuracy on most tasks of this kind. Chain of thought and sampling raised it on some of them too, but in a reasoning model the reasoning itself is a parameter of the call, and sampling costs a call for every answer. For the rest of the context four findings carry forward. (1) Length alone costs accuracy, and the part of the window a model uses well is shorter than the part it accepts. (2) Which content enters matters as much as how much, since in question answering over retrieved passages, passages related to the question that do not contain the answer did the most harm, while the effect of position depends on the task. (3) A model finds evidence in a long window worst when the evidence is worded differently from the question. (4) A model's summary or rewrite of what an agent has seen was not more accurate wherever it was measured.

2.2 Natural Language Database Querying

Natural language database querying translates a question into a query in the language of a database, such as SQL or MongoDB's query language (MQL), and is also referred to as text-to-query (Sivasubramaniam et al., 2024). The structured leg of this thesis writes its queries in MQL. The literature on that case is young and grew out of text-to-SQL. Its benchmarks either migrate relational datasets into document collections or copy the relational task design, and its methods are the prompting and agent pipelines developed for SQL. MQL was first measured in 2024 (Sivasubramaniam et al., 2024). The history of this task is therefore the history of text-to-SQL, and it is told below in five periods, (1) the benchmark that defined how the field measures itself, (2) the change from fine-tuning to prompting and the benchmarks that followed it, (3) the stages that became agents, with the first benchmark for MQL at the end of that period, (4) the point where SQL and MQL are measured side by side, and (5) the differences between the two problems.

The benchmark and the fine-tuning era. Today's evaluation of text-to-query systems starts with Spider (Yu et al., 2018). It was the first collection to combine complex SQL with a split across 200 databases, in which a system is tested on databases it never saw in training, so that it has to read a schema, the declared structure of the whole database with every table and its columns, rather than remember one. Earlier collections had either complex queries over one fixed database or tens of thousands of questions that each touched a single table. The best model of 2018 reached 12.4% exact match under the split, and the four years that followed belong to models fine-tuned on Spider.

From that era, schema linking, the stage that selects the tables and columns a question refers to before the query is written, became a named stage and a separate model step, and it has remained one to this day (Shi et al., 2025).

From fine-tuning to prompting. In 2022 a code model with no fine-tuning came close to the fine-tuned systems of two years before. Rajkumar et al. (2022) prompted Codex (Chen et al., 2021) with a Spider question and its schema and measured an execution accuracy, the share of questions whose generated query returns the same result as the gold query, of up to 67.0% on the development set, comparable to the fine-tuned state of the art of late 2020 (Lin et al., 2020). What the prompt contained decided most of the outcome. The question alone gave 8.3%, the schema as `CREATE TABLE` statements 59.9%, the schema with 3 sample rows per table 67.0%,

and past 3 rows accuracy fell. From then on the research focus shifted from training the model to designing the prompt.

The prompting papers that followed treated the prompt as a pipeline of separate calls. DIN-SQL decomposes the task into schema linking, difficulty classification, generation and self-correction, each a prompt of its own (Pourreza and Rafiei, 2023).

Gao et al. (2024) compared, under one protocol, how the question and schema are represented, which examples are selected, and how they are laid out. Their method DAIL-SQL led the Spider leaderboard at the time, and it chooses the examples for each question by the structure of the question and of a draft query rather than by the examples' topic. Examples moved GPT-4 (OpenAI, 2023) from 72.3% zero-shot to 83.5% few-shot on the development set, and GPT-4 did best with examples given as question-query pairs without their schemas. GPT-3.5 and Vicuna-33B gained less from the same examples, and for such weaker models the authors recommend examples that carry their full schema. How much a prompt design adds, and how much schema it has to carry, therefore depends on the model. Every pipeline of the following year (Table 2.1) prompts with examples, so the question that remained was which examples to give.

At the same time the benchmark moved. BIRD holds large databases with dirty values and questions that need knowledge from outside the schema, and the strongest model it tested, GPT-4, reached 55% on it against 93% for humans (Li et al., 2023). Spider had become too easy. Beyond the translation of the question into SQL, a model on BIRD had to work out how the database represents what the question asks about. A value may be stored in another form than the question uses, a column's meaning may be given only in the documentation, and a fact may be missing from the schema.

The benchmark Dr.Spider (Chang et al., 2023) added robustness, with perturbations of the Spider development set, and a fine-tuned model lost half its accuracy when the database content was rephrased. Codex, the one prompted model in the study, lost as much overall but in the other direction, little when the schema was abbreviated and most when the question no longer named the column. For the prompted pipelines of 2024 the two perturbations that cost most are (1) a synonym for the column in the question and (2) rephrased database content (Liu et al., 2025).

Pipelines become agents. In 2024 the pipelines grew more stages, and the stages were renamed agents. MAC-SQL assigns three, a selector, a decomposer and a refiner, and its ablation credits each with a gain (Wang et al., 2025a). CHESS assigns four (Talaei et al., 2024), and two of them carry a measured gain. (1) One looks up every literal the question mentions in an index of the values the database stores, so that the query filters on the stored form of a value and not on the form the question uses, and without it the system loses about 5 percentage points. (2) The other executes the candidate query and, when the database returns an error or no rows, prompts the model again with that result to rewrite the query, and without it the system loses about 7 percentage points, more than without any other stage.

CHASE-SQL led BIRD at 73.0% by spending many model calls on each question (Pourreza et al., 2025). It generates 21 candidate queries, 7 from each of 3 reasoning strategies,[1] repairs those that fail or return no rows, and picks the final one with a fine-tuned selector. On the development set a single query reaches 63%, a majority vote over the candidates 69% and the trained selector 73%. In the pipelines of 2024 the word agent thus denotes a stage, the stages run in a fixed order, and no stage chooses which one runs next.

The word agent had a stricter meaning by then, the loop of ReAct (Section 2.1), which chooses each action from the result of the last. Over a database the actions of that loop are the database's

[1] Divide-and-conquer chain of thought, query-plan chain of thought, and online synthetic example generation.

own operations, reading a schema, executing a query, searching the stored values, and the model chooses which of them comes next. The pipelines of 2024 also run a loop, but over one fixed action, execute the SQL query the model wrote and rewrite it, applied again and again until the model judges the query correct or a cap is reached. A blog post of the vendor Anthropic calls a system a workflow when its steps are fixed in code and the model fills each step, and an agent when the model directs its own process (Schluntz and Zhang, 2024). From their work with customers they recommend the simplest solution possible and more complexity only when it is needed. A workflow suits a well-defined task, since its steps are predictable and consistent, and an agent suits a task whose number of steps cannot be predicted and whose path cannot be fixed in advance. They list routing and a loop in which one call evaluates another among the workflows, so a system stays a workflow when the model only decides whether a fixed step repeats or which of several predefined paths is taken.

Self-Debug established the fixed loop (Chen et al., 2024c), and it is a workflow by that distinction, since the model decides only whether the one action repeats. The SQL query the model wrote is executed and the model is shown the rows it returned. It writes out, line by line, what its query does. Where that description says something other than the question asks, it rewrites the query to do what the question asks. On Spider that added 2 to 3 percentage points over the query written once, and 9 on the hardest questions, while asking the model whether its query was correct without executing it added nothing. CHESS's revise step is the cheaper form of that loop, since it prompts the model again only when the database returns an error or no rows, and stops after 3 attempts.

Huang et al. (2024b) sort the planning of such agents into five families, (1) task decomposition, (2) multi-plan selection, (3) external-planner-aided planning, (4) reflection and refinement, and (5) memory-augmented planning, and draw one distinction inside the first. A decomposition-first planner fixes its sub-tasks before any action runs, so one wrong step fails the task unless a further mechanism adjusts the plan. An interleaved planner chooses the next sub-task from the last result, which makes it tolerant of errors, and on complicated tasks its long runs can drift from the goal. Table 2.1 places the systems of the period in the families, with their backbone and their result. The pipelines are decomposition-first, two of them add a selection among candidates, and every one ends with a step that revises the query, so a system combines two or three of the families rather than belonging to one.

The revision steps differ in the signal they use, and the same year showed which signals work. Huang et al. (2024a) had models critique and revise their own answers with no signal from outside and found that accuracy did not improve and often fell. DIN-SQL's self-correction step is of that kind, since the model is given only its own SQL and asked to fix it. The gains of MAC-SQL's refiner, CHESS's revise step and Self-Debug come from feedback the model did not produce, an execution error, an empty result or the returned rows. Over a database that feedback is available at every step, since an executed query returns either an error or rows.

While the pipelines added stages, schema linking was found to lower the accuracy of the most capable models. Maamari et al. (2024) gave a range of models a schema in which every required column was present and added irrelevant columns, and found that the harm from irrelevant columns falls as the model's capability rises. Real schema linkers pay for their precision in recall, since a linker that keeps irrelevant columns out also drops required ones. For the strongest models every linker they tried lowered accuracy, and their pipeline with no schema linking at all led BIRD. CHESS bounds the finding at a schema of several thousand columns, where its schema selector still adds accuracy at a fraction of the tokens. Maamari et al. conclude that when the schema fits the context window and the model is capable, giving it the whole schema is safer than filtering it. Every added column still lengthens the context, and length alone costs accuracy (Section 2.1).

Table 2.1: Text-to-query systems from 2023 to 2026 and their execution accuracy on four benchmarks.

				Execution accuracy (%)				
						Spider 2.0-lite		
System	Backbone	Planning families	Signal of the revision step	Spider	BIRD	GPT-4o	o3	TEND
Fixed workflows of 2023 and 2024								
DIN-SQL (2023)	GPT-4	decomposition-first; reflection and refinement	none, the model re-reads its SQL	85.3	55.9	1.5	–	–
MAC-SQL (2024)	GPT-4	decomposition-first; reflection and refinement	execution error	86.8[d]	59.6	–	–	–
CHESS (2024)	GPT-4-turbo; Gemini 1.5 Pro	decomposition-first; multi-plan selection at high budget; reflection and refinement	execution error or empty result	87.2	66.7; 71.1	3.8	–	–
CHASE-SQL (2024)	Gemini 1.5 Pro	decomposition-first; multi-plan selection; reflection and refinement	execution error or empty result	87.6	73.0	–	–	–
Self-Debug (2024)	Codex	reflection and refinement	returned rows and the model's own explanation	84.1[d]	–	–	–	–
Free agents of 2025 and 2026								
Spider-Agent (2025)	GPT-4o; o3	interleaved decomposition (ReAct)	every execution result	–	–	13.2	24.9	–
TEND's ReAct agent (2026)	DeepSeek-V4-Flash	interleaved decomposition (ReAct)	every execution result	–	–	–	–	30.8
Fixed workflows of 2025 and 2026								
ReFoRCE (2025)	GPT-4o; o3	decomposition-first; multi-plan selection; reflection and refinement	execution error or empty result	–	–	21.8	36.6	–
TEND's solver (2026)	DeepSeek-V4-Flash	decomposition-first; multi-plan selection; reflection and refinement	gate violation, execution error or empty result	–	–	–	–	35.8

The planning families are the five named in the text before the table, and the tier of each backbone is in Appendix A. Results are on the test set, those marked [d] on the development set. TEND reports a column-tolerant variant of execution accuracy. CHESS's two BIRD values are its low-budget and high-budget configurations. A dash marks a benchmark the system was not evaluated on.

The same year MQL was measured for the first time. SM3 holds one medical dataset fixed and asks the same questions of it in SQL, MQL, Cypher and SPARQL (Sivasubramaniam et al., 2024). Zero-shot, the best open model, Llama 3-70B, reached 21.6% execution accuracy on MQL against 47.1% on SQL. Examples narrowed that gap, and which examples decided how much. GPT-3.5 reached 35.1% on MQL with the same 5 examples for every question and 78.7% with 5 similar examples retrieved for each question, still about 10 percentage points below SQL. The authors relate the zero-shot gap to how much of each language the models saw in training, since zero-shot accuracy followed the number of Stack Overflow posts for each language. The set also nests its MQL queries to an average depth of 3.77 against 0.15 for SQL.

The frontier on the SQL side and on the MQL side. In 2025 Spider 2.0 changed what the task was (Lei et al., 2025). Its tasks run over enterprise databases with hundreds of columns, and an agent works through a sequence of executions and is scored on its final result rather than on one predicted query. The best configuration, Spider-Agent with o1-preview, solved 21.3% of it, where the authors note that the leading systems of the previous benchmarks scored 91.2% on Spider and 73.0% on BIRD. On Spider 2.0-lite, a self-contained variant whose answer is one

SQL query, DIN-SQL and CHESS score far below Spider-Agent when all three run on GPT-4o (Table 2.1).

The authors of Spider 2.0 built a ReAct agent, Spider-Agent, as the baseline system of their benchmark. ReFoRCE, which led the leaderboard in 2025, is instead a sequence of four fixed steps, (1) compress the schema, (2) generate candidates with execution-based repair, (3) vote, and (4) explore the database only for the questions on which the vote is tied (Deng et al., 2025). With the same model this workflow scores 36.6% against the agent's 24.9% on Spider 2.0-lite, at 3.5 model calls per question against about 11. Unlike the fixed workflows of 2023 and 2024, this workflow scored above the agent. Their ablations show what each fixed step adds within the workflow. (1) Compressing the schema is worth about 3.5 percentage points. Of each group of tables whose names share a prefix or suffix, ReFoRCE keeps one in full, and without that step some prompts exceed the context window and fail. (2) The exploration of stored values is worth about 2.5 percentage points. (3) The vote over several candidates is worth about 2 percentage points.

A later benchmark let an agent also ask the user. BIRD-INTERACT makes every task ambiguous and adds a simulated user who answers questions about it (Huo et al., 2026). In its agentic mode a ReAct agent chooses, within a budget, between querying the database and asking that user, and GPT-5 solved 17% of the tasks. In 50 failed cases Huo et al. traced over 80% of the errors to ambiguity left unresolved, since a task held about four ambiguities and the systems asked about once. Success rose steadily the more turns a model was given to ask.

On the document side, two benchmarks of 2025 and 2026 differ in how their documents are shaped. (1) DocSpider migrates Spider into MongoDB one collection per table and one field per column, so the documents are relational in shape, and with the schema in the prompt GPT-4o reaches 71.0% zero-shot on it (Özer et al., 2025). Writing the query in SQL first and translating it into MQL beat writing MQL directly. (2) TEND designs its databases in MongoDB from the start, with array operators in almost every task, and its best solver reaches 35.8% (Lu et al., 2026). The models, the metrics and the schema given to the model differ as well, so the gap is consistent with a difficulty in the document model but does not isolate it. TEND's authors read their results as a problem of reasoning over documents without a declared schema rather than a variant of text-to-SQL in another syntax (Lu et al., 2026), while DocSpider's authors relate their results to what the models learned in pretraining and to how close a query language's syntax is to SQL (Özer et al., 2025).

TEND's baselines say where in the document model the difficulty lies (Lu et al., 2026). (1) With the collection and field names alone the model reaches 0.2%, with three sampled documents per collection 28.6%, since names reveal neither the dynamic keys, the populated paths nor the stored value forms, and the documents do. (2) Writing a SQL query first and translating it into MQL, from the same documents, gives 25.6%, below the direct write. (3) A ReAct agent that explores the databases reaches 30.8%, 2 percentage points above seeing the documents once. (4) TEND's own solver, which grounds paths and values from sampled documents before generation, checks the pipeline against them and repairs it against execution, reaches 35.8%, 5 percentage points above the agent. The authors conclude that TEND can only be solved by grounding in the documents and that writing SQL first is harmful. In their reading, a structure induced from the documents before writing is worth more than an exploration loop that re-derives it for every question, although the loop stayed competitive where documents vary in shape or nest streams of events.

MultiTEND extends the first version of TEND to questions in six languages, German among them. Depending on the method, Qin et al. (2025) measured a gap of 4 to 6 percentage points between English and the mean of the five other languages, and a smaller one for German. Its field names are translated along with the question and its stored values stay English, so part of

that gap is matching a German mention to a German field name. No MQL benchmark pairs a question in another language with field names that stay in English.

Table 2.2: Four public MQL benchmarks and the results they report.

Benchmark	Year	Built from	Documents and schema	Size	Execution accuracy (%)
SM3	2024	synthetic patient records, 408 question templates, one question set for SQL, MQL, Cypher and SPARQL	four collections with embedded entities, gold queries nested to depth 3.8 on average; schema given	10,000 pairs per language	78.7, GPT-3.5, 5 retrieved examples; 21.6, Llama 3-70B, zero-shot
DocSpider	2025	Spider's databases migrated to MongoDB, gold SQL translated to MQL by three LLMs and hand-repaired	flat, one collection per table; schema given	4,043 train and 620 dev pairs, dev evaluated	71.0, GPT-4o, zero-shot; 81.1, GPT-4o, fine-tuned
TEND	2025, revised 2026	document databases designed by experts, values from BIRD	native, arrays, nested paths and dynamic keys in over 90% of tasks; no schema	1,210 tasks, 11 databases, 32 collections	35.8, DeepSeek-V4-Flash, the paper's solver; 30.8, agent; 28.6, sampled documents; 0.2, names alone
MultiTEND	2025	TEND's first version, 154 largely Spider-derived databases, translated into six languages with its field names	nested like TEND's first version, `$unwind` in 60% of pipelines; schema given, translated	101,789 pairs, six languages	59.1, DeepSeek-V3, the paper's method

TEND reports a column-tolerant variant of execution accuracy.

What separates MQL from SQL. Four differences separate the two problems in the results above, whose benchmarks Table 2.2 collects. The first is whether the structure is declared. A relational schema declares its tables, columns and keys, and every text-to-SQL result rests on the model reading that declaration. A document database does not require a schema, its collections accept documents of any shape. TEND removes the declaration by construction, so the structure has to be inferred from the documents themselves, and in its baselines nearly all of the accuracy came from the sampled documents rather than from the names. Where an application declares a schema for its collections, the finding that a capable model does better with the whole schema than with a filtered one (Maamari et al., 2024) should carry over while the schema fits the context window, although it was measured on SQL only. What the declaration omits has to be read from the documents.

The second is the data model. A document database denormalises where it can, so much of what a relational schema expresses as a join it expresses as nesting, the join that remains is an aggregation stage, and the difficulty moves from choosing the join to navigating the path and unwinding the array. SM3's nesting depth and TEND's array operators in almost every task show how common this structure is in the MQL benchmarks. MQL in SM3 scored highest on questions about top-level documents rather than embedded ones (Sivasubramaniam et al., 2024), and Spider 2.0's agent solved 10.3% of the tasks over databases with nested columns against 27.4% of the others (Lei et al., 2025).

The third is the shape of the language, since an aggregation pipeline is an ordered sequence of stages and the same question can be answered by several pipelines that differ as text. Exact match, which compares the text of the generated query with the gold query, therefore

undercounts correct queries in MQL. In MultiTEND, zero-shot queries matched the gold text in under 1% of cases in every language, while 29 to 37% returned the gold result.

The fourth is scarcity, of MQL in what the models have seen and of MQL in what has been measured. SM3's authors count about a quarter as many Stack Overflow posts for MongoDB as for SQL. Public MQL benchmarks number four, one of them a translation of another, with distinct queries in the low thousands, so the measurements are few. Examples in the prompt raised MQL accuracy in SM3 and MultiTEND, and for the larger models in SM3 they added more on MQL than on SQL. SM3's retrieved examples lifted MQL by 44 percentage points over fixed ones against 32 on SQL in the same study, and MultiTEND's retrieved examples added 15 percentage points over zero-shot in English. On the SQL side one of CHASE-SQL's generators has the model write question-and-query pairs for the target schema, unchecked, and uses them as its few-shot examples. On BIRD it scored about 9 percentage points above the baseline prompt without examples with Gemini 1.5 Pro but about 1 with Mistral Large. Whether that holds for MQL is unmeasured. Which of the two scarcities the examples make up for, syntax the model lacks or structure it has to copy, the papers do not separate. The test that would, examples on and off under one model on flat MQL and on nested MQL, has not been run, and it is a gap of this literature.

What this leaves for querying a document database. The methods of the text-to-SQL literature should work for MQL too, since examples, repair and value exploration each raised accuracy where they were tried on MQL. Both the language and the documents make MQL harder than SQL. On flat collections a detour through SQL helped (Özer et al., 2025), while on documents without a declared schema it hurt (Lu et al., 2026), and within SM3 and within Spider 2.0 nested data lowered accuracy. Much of the difficulty therefore lies in the documents, which a model has to read before it writes. A capable model should see the whole schema when it fits the context window, a compressed schema when it does not, and one pruned by a model only at thousands of columns. The stored values have to be read at every schema size, and on both frontiers a fixed workflow with that step built in outscored the agent it was compared with, so the step is what an agent can adopt. For the larger models examples in the prompt added more on MQL than on SQL, and on SQL a strong model could even write its own. Splitting the task into separate calls helps, and on Spider 2.0 the outcome depended on the steps a system had, such as compressing the schema and exploring the values, not on whether their order was fixed in code. Repair helps only with an external signal, the query is written in MQL directly, and candidates with a vote or a trained selector transfer at a multiple of the calls. Open are which of the two scarcities the examples make up for, whether a model can write its own examples for MQL, and what a German question costs against field names that stay English. The evaluation frame transfers from Spider 2.0, an agent's whole run scored by whether it solved the task, which Spider 2.0 calls the success rate. Which of these techniques the design of Chapter 3 adopts, and how the test collection of Section 5.1 borrows the frame, is argued there.

2.3 Information Retrieval

Scouts describe players in their reports, how a player behaves on the pitch, where his strengths and weaknesses lie, and what type of player he is. So when a scout searches for a player of a certain profile, the information that answers him is inside the text of those reports, and the unstructured leg of this thesis ranks players by what the reports say about them. Information retrieval ranks documents by their relevance to a search query and measures a ranking against judgements of which documents are relevant. In its terms, that is entity retrieval, which ranks entities such as people by the documents written about them. The history is told in three periods, (1) term matching, pooled judgements and the first models of people, from 1998 to 2012, (2) dense retrieval, from 2014 to 2021, and (3) language models in retrieval, as rankers, as assessors of relevance and as authors of queries, from 2022 to 2025.

Term matching, pooled judgements and the first models of people. A retrieval system is measured on a test collection (Sanderson, 2010). It has three parts, (1) a set of documents, (2) a set of topics, each an information need with a query written for it, and (3) the relevance judgements that say which documents answer which topic. Judging every document for every topic is impossible past a few thousand documents, so since the Text Retrieval Conference (TREC), the evaluation campaign in which research groups run their retrieval systems over shared topics and submit the rankings, the judgements come from a pool. The top ranks of every system's ranking are merged down to a fixed rank, the pool depth, only that pool is judged by human assessors, and everything outside it counts as not relevant. What pooling gives is a stable answer to which retrieval system is better than which, not the true score of any of them. Voorhees (2000) showed that the order of the systems held even when other assessors judged the same topics again. Zobel (1998) found the pool fair also to systems that had not contributed to it, but estimated that the TREC pools had found at best 50 to 70% of the relevant documents. So a pooled collection said reliably which retrieval system was better but could not say how much any of them had missed.

How a ranking is scored changed in the same years. Järvelin and Kekäläinen (2002) introduced normalised discounted cumulated gain (nDCG), which scores a ranking in three steps. (1) Each document gets a gain by its relevance grade, so a highly relevant document counts more than a marginal one. (2) The gain is divided by the logarithm of the rank at which the document appears. (3) The discounted gains are summed and divided by the sum of the ideal ranking, which sorts the topic's judged relevant documents by grade. A relevant document that no assessor judged is missing from that ideal ranking and counts as zero where a system retrieves it, so nDCG is only as reliable as the judgements it rests on.

The judgement itself was split into parts in the same decade, and the reason was redundancy, since a ranking of ten relevant documents that all say the same thing serves a user no better than one. Zhai et al. (2003) wanted retrieval that covers a need. So they split the topic into subtopics, the sub-questions a full answer covers, and compute recall over subtopics rather than over documents, which they call subtopic recall (S-recall). Clarke et al. (2008) wanted a measure that rewards such coverage, since systems are tuned to their measures. So they split the answer into nuggets, the single statements a relevant document can contain, and judge a document by which nuggets it does contain, in a measure they called α-nDCG. So a subtopic is a heading and a nugget is a statement under it. Pavlu et al. (2012) wanted relevance judgements that do not depend on the pool, since the pool holds what the systems of one year retrieved and later systems retrieve differently. So their assessors extracted nuggets from a sample of relevant documents, and the relevance of any other document was inferred by matching its words against those nuggets. On TREC 8 the inferred judgements agreed with the assessors' own on most documents and found relevant documents the assessors had missed. So relevance decomposes into units, and a document's grade follows from the units it contains. In all of these collections the lowest grade a document can get is zero. The grade measures how much a document says about the topic, not whether what it says is favourable. A document that lacks a unit gets nothing for it, and no unit exists whose presence counts against a document or removes it from the ranking.

The direction of an opinion, its polarity, entered the judgements in the opinion retrieval task of the TREC Blog track (Ounis et al., 2006). Its assessors judged first whether a post was about the target and then whether it held an opinion and in which direction, and the direction was recorded without deciding relevance, so a critical post was as relevant as a favourable one to a user asking what people think. The systems of 2006 did not retrieve posts of one direction clearly more often than posts of the other. Only the polarity task of 2008 made the direction a condition, ranking positive and negative posts apart and leaving out posts that held both (Ounis et al., 2008). So in this literature the direction of an opinion is a property of a document, recorded beside its relevance or used to sort posts into two rankings.

The scoring of these collections was the term model, and its standard form has held for two decades. Robertson and Zaragoza (2009) consolidate BM25, a ranking function derived from one question, how probable it is that a document is relevant given which query terms it contains and how often. It needs no training and is robust to its two parameters, which is why it became the baseline every retrieval system in this section is measured against, and why it still beat most trained systems out of domain in 2021. Its scores order the documents of one query and nothing more, since the formula keeps only what changes that order, so scores from two different queries cannot be compared. And the model is over terms, so what counts as a term, how text is cut into tokens and tokens reduced to stems, sits outside it and differs by language.

Since scores do not compare across systems either, the rankings of several systems can be merged by their ranks alone. Cormack et al. (2009) took the rankings that several systems returned for the same query and summed, for each document, the reciprocal of its rank in every ranking, offset by a constant, which they called reciprocal rank fusion. On the rankings submitted to four TREC tasks the fused ranking beat the best single submitted ranking in three, and in the fourth only a ranking made with a person in the loop scored higher. So fusing the ranks of many systems almost always beat the best of them.

Not every retrieval task returns documents. In an organisation someone asks who knows about a topic, an area of work such as a standard the organisation maintains, and wants a ranked list of colleagues, not of documents. Nobody has written down who knows what, so the information sits in the organisation's emails and web pages, and those documents are the evidence that a colleague knows the topic. The literature calls this task expert finding, and Balog et al. (2006) gave it its two models. A candidate model gathers everything associated with a person into one profile and ranks the profiles against the query. A document model ranks the documents against the query and propagates each document's score onto the people associated with it. Balog et al. scored the rankings by mean average precision, the precision at the rank of each relevant person averaged over a topic's relevant people and then over topics. On the TREC 2005 Enterprise collection, with documents linked to people by exact name matches, the document model reached 0.19 against 0.13 for the candidate model and scored higher on nearly all topics. Its lead was significant for every way of linking documents to people except matching e-mail addresses, which linked the fewest documents.

Expert finding was set on its own footing a few years later, and the survey by Balog et al. (2012) adds three things. (1) It recasts expert finding as entity retrieval, because a person is only the first kind of entity the field ranked. (2) It advises a practitioner to start from the document model. Across the studies it compares, the document model was more effective than the candidate model and less sensitive to its parameters. It also needs nothing beyond an ordinary document search engine, since the whole model is one run of that engine followed by adding each document's score to the people linked to it. (3) A list of the documents that answer a topic is not a list of the documents that show who knows it. An expert-finding system first retrieves documents for the topic and then gives each person the scores of the documents that name him. One study the survey reports replaced the first step by the assessors' own list of documents relevant to the topic, expecting the best possible expert ranking from the best possible documents, and got a worse ranking than with an ordinary retriever. The assessors' judgements were made for document retrieval, whose question is which documents a reader of the topic should open, and the study had borrowed them as the ground truth for a task whose question is who knows the topic. So the documents that show who knows a topic include documents no reader would open, and a system that filters them out by document relevance before it counts evidence per person loses that evidence. So a collection for an entity retrieval task has to judge its documents for the entity retrieval question, whether a document shows that this person knows this topic, and not for the document retrieval question.

A related task arose for other entities, with reviews as the documents. Ganesan and Zhai (2012) wanted to rank hotels and cars by what their reviewers wrote, for a query that states preferences

on several aspects, such as clean rooms and a good location. They represented each entity by the text of all its reviews, as the candidate model represents a person. Scoring each aspect of the query on its own and averaging the scores ranked slightly better than one search for the whole query, above all for queries with more aspects. No collection existed for the task, and judging how well an entity matches a preference would have meant reading all of its reviews for every query. So they took the ratings that reviewers gave each aspect beside their text as the judgements. The relevance of an entity to a query was the average of its ratings on the aspects of the query, so a rule derived the relevance of every entity and no pool was needed. A low rating lowered a hotel's grade, and since the grade averaged over the aspects of a query, a hotel rated poorly for its location could still rank high for its clean rooms.

Dense retrieval. The term model compares words, and dense retrieval compares vectors. The vectors come from a transformer used in one of two shapes. A cross-encoder reads a query and a document together and scores the pair. A bi-encoder embeds each text on its own, so that the similarity of two texts is the cosine of two vectors computed once. The first is the more accurate of the two and the second is the one that scales. Reimers and Gurevych (2019) trained the pretrained transformer BERT as a bi-encoder, which they called Sentence-BERT.

Sentence-BERT compared sentences, and Karpukhin et al. (2020) trained a bi-encoder to retrieve, dense passage retrieval (DPR), with one encoder for the question and one for the passage, trained on question-passage pairs so that the vectors of a pair lie close. The retriever of open-domain question answering had been BM25, which misses a passage that answers the question in other words, since it matches tokens. On the question set Natural Questions (Kwiatkowski et al., 2019), DPR placed the answer passage in its top 20 for 78.4% of the questions, against 59.1% for BM25. So a bi-encoder trained for retrieval beat the term model in the domain it was trained on, while its transfer to a different corpus was not measured.

In the same year a third shape between the cross-encoder and the bi-encoder matched the cross-encoder's ranking in a small fraction of its time. Khattab and Zaharia (2020) wanted the word-level matching of a cross-encoder with the documents encoded ahead of time, as a bi-encoder encodes them. So their retriever ColBERT keeps one vector per token of each text and scores a pair by summing each query token's similarity to its most similar document token. The literature calls this architecture late interaction. The collection MS MARCO is built from real Bing search queries. ColBERT reranked BM25's top 1,000 passages for those queries as well as a BERT cross-encoder did.

The next question was whether a bi-encoder works on a corpus it was not trained on, and Thakur et al. (2021) measured that it does not. Their benchmark, BEIR, gathers 18 existing test collections into one format. All 18 are scored with nDCG at rank 10. DPR had been trained on Natural Questions, and none of the other 17 collections was in the training data of a tested retriever. Every bi-encoder averaged below BM25 there. In domain the order had been the opposite, since every bi-encoder scored above BM25 on the collection it was trained on. A cross-encoder that reranked BM25's results, by contrast, scored above BM25 alone on 16 of the 18 collections (Table 2.3). So a bi-encoder of that time worked in the domain it was trained on and averaged below BM25 outside it, and BM25 remained a strong baseline for a collection no system was trained on.

The same paper found that the pools of those collections, built by the lexical systems of their day, score the dense retrievers too low. The share of unjudged documents in the top 10 was 6% for BM25 and 31% for DPR. Judging those documents raised the scores of the dense retrievers. So a pooled collection undercounts the systems that retrieve differently from those that built the pool.

How relevance is judged changed in the same years as well, on its own and not because of the embedders. Two papers let a retrieved document count against the system, one in existing

collections and one in a new one. Gienapp et al. (2020) found that assessors had already given junk documents a negative label in the TREC collections of several years, and that the standard evaluation tool, trec_eval, maps that label to zero. Keeping the label as a negative gain instead of mapping it to zero changed the order of the systems in three of five track years, in some of them at the first rank, so that the best system may depend on the choice. They normalised nDCG between its lowest and highest possible value so that a negative gain fits, and found this measure more reliable than mapping the label to zero. Clarke et al. (2021) built a new collection, in the TREC Health Misinformation track, for a case where harm is the point, search results with health advice. Each useful document is judged for whether it supports or dissuades from a treatment, and one that contradicts the stance fixed for the topic counts as harmful and against the system. Since nDCG would ignore the harmful documents, they scored each submitted ranking by its compatibility with the helpful documents and, separately, with the harmful ones. Compatibility scores a ranking by its similarity to the closest of the ideal rankings (Clarke et al., 2020). So a harmful document became measurable, once by repairing the measure in existing collections and once by building a collection and replacing the measure.

Language models in retrieval, as rankers, as assessors of relevance and as authors of queries. Language models were first used in retrieval at the query. A query and the passage that answers it are phrased differently, so an encoder has to learn to place a query next to its answer, and that learning needs labelled pairs of a query and its answer. Gao et al. (2023a) wanted retrieval on a new corpus that has no such pairs. So in their method, HyDE, they have the model write a hypothetical document that would answer the query, embed that document in place of the query, and match document to document, since the written answer is already phrased like the documents that answer the query. Matching document to document narrowed the gap between an unsupervised dense retriever and fine-tuned ones without any relevance label. On some tasks the unsupervised retriever matched the fine-tuned ones. The gain was largest where labels are scarcest.

Fusing the rankings of several phrasings of one question raised effectiveness on average at a risk for single topics. Benham and Culpepper (2017) wanted to know how often such fusion harms single topics, which an average score hides. Reciprocal rank fusion over phrasings written by people raised average precision on the news collection Robust04 from 0.254 to 0.331, while about 30% of the topics got worse than with a single phrasing. They weighed the losses with a risk measure. Reciprocal rank fusion still beat the single phrasing when each loss counted twice and no longer differed significantly from it when each loss counted six times. So whether fusion pays depends on how heavily a loss on a single question weighs. Rackauckas (2024) wanted one query to reach the passages that its single phrasing misses, so in RAG-Fusion a language model writes the rewrites, each is retrieved and the rankings are merged by reciprocal rank fusion. The answers got more complete, but people judged this by reading them rather than measuring it on a benchmark, and the method failed in one way. If a rewrite the model wrote means something the user did not, the passages retrieved for it enter the merged list and pull the answer toward that meaning. The risk was measured with BM25, where a new phrasing matches different words. With an embedder the phrasings of one question stay close in meaning, and whether that lowers the risk and the gain alike is unmeasured in these two studies.

In RankGPT (Sun et al., 2023), a language model reranked the top 100 of BM25 listwise, that is, it was given the candidates as a list and answered with a new order for them instead of scoring each candidate on its own. With GPT-4 this listwise instruction reached 75.6 nDCG at rank 10 on TREC Deep Learning 2019, against 73.2 for the best of the cross-encoders trained on labelled query-passage pairs. The listwise instruction has one cost. It improves the order it was handed rather than producing one, so it inherits the retriever's order, since given the same candidates in random order the GPT-3.5 version fell from 65.8 to 25.2.

As the assessor of a collection's relevance a model agrees with the human assessors at a level that depends on its prompt. Faggioli et al. (2023) measured GPT-3.5's agreement with the TREC assessors at a Cohen's kappa of 0.26, half the 0.52 an earlier study had found between two sets of human judgements, and Thomas et al. (2024) measured GPT-4 above that level with one prompt and below it with another.

The risk of a model as assessor is a circularity, and it remains at every level of agreement. Faggioli et al. (2023) name it. A model good enough to assess relevance could also rank, and a collection it labelled would rate that model too high, possibly as perfect. Systems that estimate relevance in another way, such as BM25, could be penalised. Faggioli et al. add that models trained on similar web corpora may give correlated judgements. Two studies of 2025 measured that correlation. A model judging other models scored higher the models whose mistakes resembled its own, also when they came from another family (Goel et al., 2025), and most of all those from its own provider or with its own architecture (Kim et al., 2025a). Kim et al. found that on a benchmark of questions with four answers, two models that were both wrong chose the same wrong answer about 60 % of the time on average. Picking at random among the three wrong answers would give 33 %, so the models agreed almost twice as often as chance. In both studies, the more accurate the models, the more alike their errors, even across providers.

Dense retrieval crossed languages in the same years, and the literature keeps two problems apart. Retrieval over a corpus with documents in several languages is called multilingual retrieval. Retrieval with a query in one language over documents in another is called cross-lingual retrieval, and it asks the embedding to align across languages rather than to handle each language on its own. Chen et al. (2024a) wanted one embedder for every language and for dense and sparse matching alike, since the embedders before it offered one or the other. So they trained BGE-M3, one encoder for more than 100 languages with three outputs, (1) a dense vector for the whole text, (2) a learned weight per token for sparse matching, which scores the tokens query and passage share as BM25 does, and (3) one vector per token for late interaction as in ColBERT. The benchmark MIRACL asks questions in 18 languages, each against passages in its own language. On MIRACL BGE-M3 reached 71.5 nDCG at rank 10 with its three outputs combined and 69.2 with the dense vector alone, against 66.6 for multilingual-e5-large (Wang et al., 2024), the strongest embedder Chen et al. compared it with. The cross-lingual benchmark MKQA asks questions in 25 languages other than English against English passages. Neither benchmark searches a corpus that mixes languages.

With embeddings, scoring each quality of the query on its own and combining the scores again beat one search for the whole query where each document spoke of one quality, as it had by a small margin with the term model (Ganesan and Zhai, 2012). Korikov et al. (2024) wanted to rank entities that are described by many short documents, for a query that asks for several qualities at once. They call each such quality an aspect. Since no dataset of such documents existed, they generated recipe reviews with GPT-4, which let them control how the aspects are spread over the reviews. They compare two levels. First, one embedding of the whole query against retrieving per aspect. Second, where retrieval is per aspect, combining the scores against combining the rankings. Where each review mentioned only one aspect, retrieving per aspect and averaging the scores beat one embedding of the whole query, with a mean average precision at rank 10 of 0.56 against 0.41 for the dataset's labelled aspects and 0.46 against 0.41 for aspects a language model extracted. Where every document covers every aspect the two were level. The mean average precision of taking the minimum over the aspects fell to near zero once more than one review per aspect was averaged. Averaging the per-aspect scores also beat combining their rankings. So evidence spread over many documents is retrieved per part and its scores combined onto the entity, as the document model did.

Three benchmarks of 2024 and 2025 measured that similarity stops being relevance where relevance needs (1) a negation, (2) an exclusion or (3) an inference step, and Table 2.3 places the retrieval architectures on them. Weller et al. (2024) wanted to measure what negation does

to neural retrieval, since the earlier work on negation in retrieval was written before the neural models and nobody had measured it since. So NevIR pairs two documents that differ by one negation and asks a retriever to rank the pair for two queries. The score is the share of pairs ranked right for both queries. No bi-encoder and no late-interaction model reached the chance level of 25%, and only cross-encoders exceeded it. Zhang et al. (2025a) wanted to measure whether a retriever honours a query that rules something out, since users routinely say what they do not want and no benchmark had measured it. So ExcluIR puts an exclusion into the query. Every architecture trained on standard data scored between the chance level of 50% and 71%. The term models fail because a model that matches terms scores the excluded entity high on the very document it rules out.

The third benchmark moved relevance one step further, from a word to an inference. Su et al. (2025) wanted a benchmark on which relevance needs reasoning rather than shared vocabulary, since on the existing benchmarks the relevant document usually shares words with the query. So they built BRIGHT, 12 datasets of queries taken from StackExchange posts, coding problems and theorem-based problems in science and mathematics. The embedder that led BEIR at 59.0 nDCG at rank 10 scored 18.3 on BRIGHT. BM25 scored as much as the smaller dense models. The cross-encoder that BRIGHT tested was trained on MS MARCO. Reranking with it scored below BM25 alone. A language model as the reranker gained 3 points over the BM25 ranking. A model reasoning about the query before retrieval gained up to 12 points over it. So similarity fails where relevance needs a negation, an exclusion or an inference step. A model that reads query and document together recovers part of it. On negation that model is a cross-encoder and on inference a language model as the reranker. A model that reasons about the query before retrieval recovers part of it too.

Table 2.3: Retrieval models by architecture on four benchmarks.

Model	BEIR nDCG@10	BRIGHT nDCG@10	NevIR Pairs (%)	ExcluIR Queries (%)
Term model				
BM25	42.3	14.5	–	53.5
Bi-encoder				
DPR	23.7	–	6.8	58.8
all-mpnet-base-v2	–	14.9	8.1	69.0
GritLM-7B	–	21.0	–	–
Late interaction				
ColBERT	43.1	–	19.7	60.0
Reranker				
MiniLM cross-encoder	47.6 (+5.3)	13.1 (−1.2)	–	63.4 (+9.9)
monoT5 3B cross-encoder	–	–	50.6	70.7 (+17.3)
GPT-4	–	17.4 (+3.1)	–	–

nDCG@10 is the mean nDCG at rank 10 over BEIR's 18 collections, computed from the paper's per-collection table, and over BRIGHT's 12 datasets. NevIR counts the document pairs ranked right for both queries, where chance is 25, and ExcluIR the queries whose wanted document ranks above the excluded one, where chance is 50. A value in brackets is a reranker's difference from BM25 alone in the same experiment, where BM25 scores 14.3 on BRIGHT. A model is listed when at least two benchmarks tested it, with two exceptions, GritLM-7B, which stands for the embedders with billions of parameters, and GPT-4, because only BRIGHT tested a language model as reranker. GritLM-7B is from Muennighoff et al. (2025), monoT5 from Nogueira et al. (2020), and the MiniLM and mpnet checkpoints from the Sentence-Transformers library (Reimers and Gurevych, 2019). A dash marks a model the benchmark did not test.

What this leaves for ranking players by their reports. Four strands of the retrieval literature bear on ranking players by their reports. In the terms of information retrieval the players are the entities, and the scouting reports are the documents that describe them. The first strand concerns what is scored. In expert finding, scoring each document about a person on its own

and then combining the scores ranked people better than one profile built from all of that person's documents. Scoring each quality of a query on its own and combining the scores onto the entity ranked better than one search for the whole query, clearly with embeddings where each document spoke of one quality and slightly with the term model. The second strand concerns how the reports are found. Outside their training domain the bi-encoders of 2021 scored below BM25, but on the newer benchmarks the strongest embedders score above it and the smaller ones match it (Table 2.3). BM25 remains the reference an embedder is measured against.

The third strand concerns which relevance a retriever can see. A retriever scores how much a report is about the quality asked for, which the literature calls topical relevance, and whether the report affirms or denies that quality does not enter the score (Table 2.3). So relevance that needs a negation, an exclusion or an inference step lies outside topical relevance, and a bi-encoder can rank a report that denies a quality through a negation as high as one that affirms it. Once the scores are combined, such a report can wrongly raise the player's score for that quality, and reranking with a cross-encoder or a language model reduces this failure without removing it. Whether current retrievers fail on polarity as they fail on negation is unmeasured. The fourth strand concerns how such a ranking is measured. Pooled judgements give a stable order of systems but leave unknown what share of the relevant reports a system never found, and both pools and model assessors favour systems like the ones that produced the judgements. Deriving the relevance of every entity by a rule from labels of single reviews avoids the pool, but in the collection that did so for hotels and cars the labels were ratings beside the text, and its average let a strong aspect make up for an opinion against another. So measuring whether a denying report wrongly raises a player's score needs a collection built for that question, and RQ1 asks how such a collection is built where the full set of relevant reports is not knowable. Chapter 3 argues which of these findings the unstructured leg adopts, and Section 5.1 how its collection is built and scored.

2.4 Hybrid Systems over Structured and Unstructured Data

A question to a scouting database can need both kinds of data at once, a player's age and club from the database and a quality that only the reports describe. Structured data is read by a query that a model writes (Section 2.2), and text is read by retrieval (Section 2.3). Retrieval-augmented generation, in its broad sense, is any system that adds information fetched from an external store to what a model generates from, and Gao et al. (2023b) count a database among those stores. This section calls such a store a source, and the sources that matter here are a database, which holds structured data, and a collection of documents, which holds unstructured text. A system that reads both kinds of data for one question is called hybrid in this section. In information retrieval, however, a hybrid system is one that combines sparse and dense retrieval (Section 2.3), and that sense is not meant here. The section follows four periods, (1) retrieval-augmented generation over text, (2) the decision per question what to retrieve and from where, (3) agents and what they cost, and (4) the systems that combine a database with a corpus.

Retrieval as a source of knowledge. A language model holds what it learned in its parameters, where that knowledge is hard to inspect, to update or to trace to a source. Lewis et al. (2020) wanted a model whose knowledge could be read from outside, so they coupled a retriever with a generator, trained the generator together with the retriever's encoder for the question, and named the combination retrieval-augmented generation (RAG). The retriever was DPR over Wikipedia (Section 2.3), and the generator wrote its answer from the passages it was given. On Natural Questions RAG, whose generator had 400 million parameters, reached 44.5% exact match against 34.5% for T5 with 11 billion parameters answering from its parameters alone. After the release of the chat model ChatGPT in late 2022 the techniques multiplied, and Gao et al. (2023b) ordered them into three paradigms, (1) retrieving once before generating, (2)

adding steps before and after retrieval, such as rewriting the search query and reranking the passages, and (3) composing such steps as interchangeable modules.

The decision per question. Retrieving for every question costs time, and a passage retrieved for a question the model could answer alone adds noise. Jeong et al. (2024) put the decision in front of the model. A fixed strategy either sent simple questions through expensive multi-step retrieval or gave complex ones a single retrieval step that did not suffice. So they trained a classifier to predict a question's complexity from its text and to pick one of three strategies, (1) no retrieval, (2) one retrieval step or (3) several. The classifier was a small T5 model. Its training labels came from running all three strategies on each question and taking the simplest one that answered it correctly. A question that no strategy answered took the strategy its benchmark was built for, one retrieval step for a benchmark whose questions one passage answers and several for a benchmark whose questions need several passages. Its classifier agreed with the automatically assigned strategy for 55% of the questions. The authors let three models of different size write the answers, and with the smallest, Adaptive-RAG reached an F1 of 46.9 against 48.9 for multi-step retrieval in less than half the time. So a cheap classifier in front of the expensive step kept most of the quality at a fraction of the cost, and such a classifier is called a router. Its choice was among ways of retrieving text.

The router later chose among sources. Bai et al. (2026) built a rule-driven router that sends each question to a relational database, to documents, to both concatenated, or to no retrieval. They first sent every question of one question set down each of the four paths. Some questions were answered only from the database and others only from the documents, and concatenating both made questions fail that one source alone answered, since the redundant evidence distracted the model. Their router is a language model that follows a list of rules in its prompt. It sees only the question and the rules. It sends each whole question down one path, so a question that needs both sources gets both in one prompt, and the language model has to combine them. With the backbone Qwen2.5 the router beat each of the four naive baselines, which send every question down the same path, on three question sets with tables and text. On the two sets where cost was measured, concatenating both sources also used more tokens than routing. Their database path finds tables with BM25 over a short text per table and adds records extracted from them to the prompt as text. So the router pattern of Adaptive-RAG carried over from strategies to sources, with the database read as one more retrieval source.

Pipelines become agents, here too. As in text-to-query (Section 2.2), the pipelines of RAG became agents. By the distinction of Section 2.2 the routers above are workflows, since code fixes the paths a question can take, and Schluntz and Zhang (2024) list routing among the workflow patterns, a classification of the input that selects a specialised next step. They recommend routing where the inputs fall into distinct categories that are better handled separately and can be classified reliably.

What agents cost became a question of its own. Kapoor et al. (2025) wanted agents that are useful in practice and not only accurate on benchmarks, since an agent's accuracy can be raised by calling the model again and the papers proposing agents reported accuracy without cost. On the 164 programming problems of HumanEval (Section 2.1), they called GPT-4 again, up to five times until its program passed the example tests that come with each problem. The retried model solved 93.2% of the problems and the best of the three published agents they re-ran 93.3%, a difference the authors found not significant, and the retried model cost less than half as much as that agent. They concluded that agent evaluations must control for cost, and left open whether reflection and debugging help on harder tasks.

Kim et al. (2025b) compared a single agent with four ways of coordinating several agents across nine models. Coordination raised accuracy where a task split into independent parts and lowered it where each step depended on the one before, and its gain shrank once a single agent already

solved about 45% of a benchmark's tasks. The authors read this as a trade-off. A single agent keeps the whole history in one context, while several agents explore more but split that context and spend turns coordinating, and once a single agent already does well that cost outweighs the gain. The study "Efficient Agents", on assistant questions that take several steps of web search, found that within one agent the backbone had the largest effect, and that sampling several actions and keeping the best added cost for a marginal gain in accuracy (Wang et al., 2025b). So on HumanEval and on web search the simpler design came close to the more elaborate one at a lower cost, and several agents raised accuracy only where a task split into independent parts.

Structured and unstructured together. The systems of this period answer one question from both kinds of data, and they differ in where the two kinds are combined. Liu et al. (2024b) found that 80 of 162 crowdsourced questions and conversation turns for a restaurant assistant needed both the structured data about the restaurants and free text such as their reviews, and they argued that a classifier sending each whole question to one source cannot answer these. So they kept text-to-query and extended the query language instead. Their language SUQL is SQL with a function that passes a text column, such as a restaurant's reviews, to a language model together with a question. A language model translates each question into one SUQL query, in which a quality that stands in a column becomes an ordinary condition and a quality that stands only in the text becomes a call of that function. The compiler evaluates the conditions on the columns first, such as the cuisine, and calls the language model only for the restaurants that meet them. Of the restaurants SUQL returned for 100 questions asked outside a conversation, 93.8% met every condition of the question, against 57.0% for a system that wrote no SQL. That system turned each database row into text and retrieved the row whose embedding was closest to a search query that a language model wrote from the question.

Whereas SUQL extended the query language of one database, Xia et al. (2025) gave every source the same small query language in their system ER-RAG. So a database, Wikipedia and a web page are all read in the same way. A query consists of GET calls, each of which reads entities from one source, and JOIN calls, which pass the values that one GET returned into the filter of the next. The system translates each GET into the source's own access, SQL for a database table and retrieval for text. A selector picks one source or a combination of sources per question, and a fine-tuned model then writes the query. On the benchmark CRAG, with each answer scored by a language model, choosing the sources reached 58.3 against 55.2 for putting every source's results into the context. A query across the database and Wikipedia for every question scored 51.2, below both settings but above every single source. The authors read this as a sign that a query whose steps read different sources helps. The model learned to write queries across sources from synthetic questions that need two steps, and only a synthetic test set consisted of such questions alone.

What this leaves for a question over a scouting database and its reports. Four findings of this literature bear on answering one question from the structured data of a scouting database and from the text of its reports. (1) Questions that need both kinds of data are common, and neither a query nor retrieval alone answers them. (2) Choosing a path for each question before the expensive step keeps most of the quality at a fraction of the cost. The earlier routers, such as Adaptive-RAG, chose among ways of retrieving text, and the later ones between a database and a collection of documents. (3) Deciding for each question which sources it needs answered better than always giving the model both and better than always querying across two sources, although on CRAG such a query still beat every single source. (4) The systems combined the two kinds of data in three ways, inside the query language of one database, in one prompt to the answering model after a router had sent the whole question to both sources, or in a query language shared by all sources, in which one query passes the values of one source into the next. No system in this literature combined a document database with text.

2.5 Open Questions from the Literature

Read across the four sections, two principles recur. The first is that, of the content a call is given, content on the call's topic that lacks what the call needs can do the most harm. In question answering over retrieved passages, passages related to the question that do not contain the answer lowered accuracy more than unrelated text did, although in reasoning tasks padded with text, unrelated text lowered it more (Section 2.1). A document that denies a quality is about that quality, so a retriever that scores topical relevance does not separate it from one that affirms it (Section 2.3). Hybrid systems that decided for each question whether it needs the database, the documents or both answered better than those that always gave the model both (Section 2.4). In text-to-query, by contrast, what a call was given was either related to the task, examples of queries, which raised accuracy when they were chosen by the structure of their query, or unrelated to it, columns the query does not need, whose harm fell as the model's capability rose (Section 2.2). So in text-to-query, what a call was given helped or did no great harm.

The second principle is that the steps a system has decide more than who orders them, and among the steps that paid are those that read what the model cannot know in advance. Practice recommends that an agent read data when a step needs it rather than load it in advance (Section 2.1). Over a database, sample rows, a lookup of the values a question mentions and a structure inferred from sampled documents each raised accuracy, and on the enterprise benchmark the outcome depended on steps such as compressing the schema and exploring values, not on whether their order was fixed in code or chosen step by step (Section 2.2).

Two findings do not fit together at first sight. The first concerns steps that repeat the work. Voting over several answers added accuracy where answers can be compared or executed (Sections 2.1 and 2.2). A model retried until its program passed the example tests of the problem matched the best of three published coding agents at less than half the agent's cost, whereas sampling several actions and keeping the best added little for an agent answering questions by web search (Section 2.4). So repeating the work paid where a correct answer can be recognised while the system runs, by tests, by execution or by agreement between answers, and without such a check it added cost and little accuracy (Sections 2.1 and 2.4). The second concerns where the evidence stands in the window. Position mattered on the models of 2023. On later models, however, its effect depended on the task, and on the one model tested for both, length weighed more than position for questions that need two inference steps (Section 2.1). So the effect of length is the one most likely to hold for current models.

The findings of this chapter were measured on models of several generations, whose capability rose fast from one to the next (Appendix A), and often on small or synthetic test sets. So what can be expected to hold for current models is the direction of an effect, not its size. What the literature settles and leaves open differs by research question.

Test collection (RQ1). Pooled judgements order systems reliably but leave unknown what a system never found, and pools and model assessors favour systems like the ones that produced the judgements. A collection for entity retrieval judges each document for whether it shows that the entity has the quality (Section 2.3), and in text-to-query an agent's whole run is scored by whether it solved the task (Section 2.2). Relevance derived by a rule from labels of single documents avoids the pool, but the collection that did so rests on ratings beside the text and on an average in which one aspect makes up for another (Section 2.3). Since no study has tested current retrievers on polarity, measuring whether a denying document wrongly raises an entity's score needs a collection built for that question (Section 2.3). Chapter 5 takes this up.

Architecture (RQ2). Questions that need structured and unstructured data are common, and the hybrid systems combined the two kinds of data in three ways (Section 2.4). On the database side a capable model does better with the whole schema than with a filtered one while the schema fits, stored values have to be read before the final query, and for the larger models

examples weigh more on MQL than on SQL (Section 2.2). Open are which way of combining transfers, since no system combined a document database with text, and what a question in another language costs against field names that stay English. Chapter 3 takes up the first, and Chapter 5 measures the second.

Context engineering (RQ3). The studies agree on the direction of four effects, (1) length, (2) content in question answering over retrieved passages, (3) the wording of the evidence and (4) a model's summary or rewrite of an agent's context (Section 2.1). Open is which recommendations of Table 2.4 transfer to an agent that queries a scouting database, since none of the techniques of practice was tested on an agent that queries a database. Chapter 3 takes them up, and Chapter 5 validates a selection.

Table 2.4 collects the recommendations on context engineering. R1 to R4 concern what a single call is given, R5 to R7 how an agent's context is kept across its steps, R8 the schema and R9 a context that improves with use. Two of them conflict for an agent that queries a database, since R5 loads data only when a step needs it, whereas R8 gives a capable model the whole schema in advance while it fits.

Table 2.4: Recommendations of the literature on context engineering from Sections 2.1 and 2.2.

	Recommendation	Sources	Model years	Buys	Costs
R1	Put what matters at the start or at the end of the window	Liu et al. (2024a); Hsieh et al. (2024b)	2023	accuracy where position matters, on later models for some tasks only (Modarressi et al., 2025; Hong et al., 2025)	a fixed order of the prompt
R2	Give a call what its decision needs and no more	Levy et al. (2024); Hsieh et al. (2024a)	2023–24	the same accuracy at a lower cost	for a capable model, more accuracy lost to dropped content than gained (Maamari et al., 2024)
R3	Keep out passages on the question's topic that lack the answer	Cuconasu et al. (2024); Hong et al. (2025)	2023; 2023–25	the accuracy such passages cost	a judgement that costs a call and can drop evidence
R4	Select the evidence before the call, above all when it is worded differently from the question	Modarressi et al. (2025); Hong et al. (2025)	2024–25; 2023–25	the accuracy lost at length	evidence the selection misses never reaches the call
R5	Load data through identifiers when a step needs it, not all in advance	Rajasekaran et al. (2025); Chaturvedi et al. (2026)	none; 2026	a short window per step	one more step per load, and with two of five models more tokens than without it at the same solve rate
R6	Inside a loop keep the raw actions and observations, failed ones included	Wang et al. (2025b); Ji (2025)	2025; none	the cheapest of six kinds of memory and slightly the most accurate, and fewer repeated failures	a window that grows with every step
R7	Give a sub-task its own clean window and return only its result	Rajasekaran et al. (2025); Sun et al. (2026)	none; 2025	the caller's window stays short	the caller cannot check what the sub-agent saw
R8	Give a capable model the whole schema while it fits, and shorten only very large ones	Maamari et al. (2024); Talaei et al. (2024); Deng et al. (2025)	2024–25	no required column is filtered out	the whole schema in the tokens of every step
R9	Change a lasting context by small edits, not by rewriting it as a whole	Zhang et al. (2026)	2025	what the context held does not collapse	a context that keeps growing

The model years are the release years of the models the evidence was measured on, and none marks a source that measured no model. Rajasekaran et al. (2025), Ji (2025) and Hong et al. (2025) are grey literature, the first two blog posts and the third a vendor's technical report.

3 System Design

This chapter describes the system that the thesis built. It answers RQ2 at the level of the design and sets out the context-engineering decisions that RQ3 examines. It starts from what scouts ask and from the data the answers come from, gives an overview of the system and then takes its components in turn. For each component it states the design, the finding of Chapter 2 the design rests on and which earlier version the design replaced and why. Limits, budgets and cuts appear here as rules, and Section 4.3 gives the settings the system runs with.

3.1 Requirements and Overview

Scouts search for players by what a database records about them, such as age, position or club, and by qualities that scouts describe in the text of their reports, which is unstructured data. A question can ask for facts only, as in “left-footed centre-backs under 23 from the Bundesliga”, for a quality only, as in “players with an explosive first step”, or for both, as in “left-footed centre-backs under 23 from the Bundesliga with an explosive first step”. The last kind needs structured and unstructured data together, and it is the kind of question the system is built for.

The core requirement is that the system answers questions of all three kinds as scouts phrase them and returns the results most relevant to each question. What makes this hard is the data each club creates itself. A club’s database combines a shared base of general football data, such as players, clubs and transfers, with its own data, such as watchlists, tags and scouting reports, which hold free text and rating fields the club defines. The club’s own data can differ widely from one club to the next, and the system has to work with whatever a club has created. Within one club, the facts of a question often span several collections, whose references to each other are stored in more than one form. The club’s data is also written in different languages. Football is global, and the scouts of one club write their reports in their own languages, so a question in one language has to find reports in all of them and be answered in its own. The club’s own data usually carries read permissions as well, so not every user sees the same, and the system has to respect what each user may see. A scout usually follows a search with questions about its result, so a question also has to be able to build on the answers before it in the same chat session.

Figure 3.1 shows the system built to meet these requirements, a routing workflow (Schluntz and Zhang, 2024) with one sub-agent. A router reads the question and tells apart two kinds of condition. A fact is something the data states, and a trait is a quality that has to be judged from what scouts wrote and rated. A quality for which the question itself names a threshold, such as a grade of at least 4 for heading, counts as a fact (Section 3.2). The router assigns the question one of four routes, (1) a fact route for questions that ask only for facts, (2) a trait route for questions that ask only for traits, (3) a hybrid route for questions that ask for both and (4) a clarification route that asks the user back. On the hybrid route the router also splits the question into its facts and its traits. Fixed code, the dispatch, then runs the parts in order. The fact part runs first, because the players it returns narrow the trait search. It goes to the MQL sub-agent, an agent that the workflow calls for the fact part and that writes queries in MongoDB’s query language (MQL) step by step. The MQL sub-agent is the only component that plans its own steps. The traits go to the trait search, a fixed procedure over the report texts and the report fields. A single synthesis call writes the answer. Two memories carry a chat session forward.

The chat history reaches the router and the synthesis call, and a session memory keeps what the MQL sub-agent found for its next run.

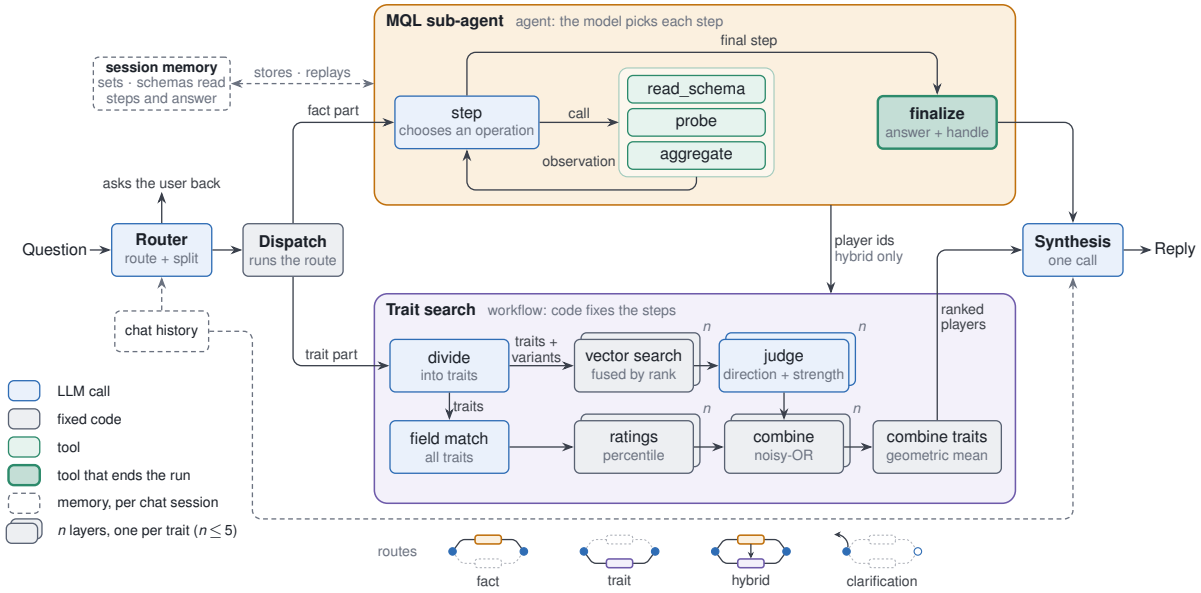


Figure 3.1: The system as a routing workflow with one sub-agent. The router assigns a route and splits the question into facts and traits, the dispatch runs the fact part first and passes its players to the trait search, the MQL sub-agent is the only component that plans its own steps, and one synthesis call writes the answer. Dashed boxes are memory, replayed into the next call of the same chat session, and the small diagrams below the trait search show which parts each route runs. The steps inside the two large boxes are described in Sections 3.3 and 3.4.

Structured data and text, which the hybrid question needs together, are combined in the system on two levels, one nested in the other. At the top, the router and the dispatch combine the MQL sub-agent with the trait search. Inside the trait search, the text of the reports is combined with their structured rating fields, so what the trait search hands to the top level already combines both. The following sections describe the router and the dispatch, the MQL sub-agent, the trait search and the synthesis in turn.

3.2 Router and Dispatch

3.2.1 Facts and Traits

The system answers from two sources. The database records facts about players, clubs and competitions in its fields, and its reports hold text in which scouts describe what they saw. The first design made both sources accessible as structured and unstructured data, in the way of the hybrid systems that decide for each question which sources it needs (Bai et al., 2026; Xia et al., 2025). A question went to the database, to a search over the report texts or to both, it was divided into a part for each source, and the results were combined.

Building the search over the report texts changed this picture, because it raised the question of what the search is for. The answer was not the texts but one concept the reports cover, the trait. A question about a quality asks for a trait, so the search has to serve that concept and not a kind of data. For a trait the text of the reports is not enough, since scouts also record the same judgement in the rating fields of their reports, which are structured data. The search over the report texts therefore became the trait search, which reads both.

Like the first design of this system, SUQL (Liu et al., 2024b) answers each condition of a question either from a column or from the text. That works where each kind of data holds its own kind of information, so that the kind of data already tells which part of a question it answers. In scouting

data it does not. The judgement of a trait stands in the text of the reports and in part also in their rating fields, so structured data serves the same information as the text without holding all of it. A question therefore has to be divided by the information it needs, a fact that the data states or a trait that has to be judged from the evidence, and not by the kind of data that holds it. The combination of structured data and text remains, but on the two levels named in Section 3.1, inside the trait search and around it.

The line between facts and traits lies in whether the data states what a condition asks for, not in how a question words it. Every condition that a query over the stored data can answer is a fact, even when the question words it differently from how it is stored or asks in another language. A question for "players we have recommended signing" sounds like a judgement, for instance, but the recommendation could be stored in the database as a tag. How such a condition is stored is worked out later by the MQL sub-agent, when it writes the query. A trait asks for a verdict that the data does not state. Reports carry grades for heading, but whether they make a player "good in the air" is stored nowhere. Which fields count toward that, how the text weighs against them and where good begins are left open by the question, and deciding them is a judgement of its own. The rating fields are therefore evidence for a trait, together with what scouts wrote. A question that names a grade and a threshold, such as "a grade of at least 4 for heading", has made that judgement itself and goes to the fact route. A question that names only the quality leaves the judgement to the system, which makes it by a fixed procedure in the trait search and does not leave the MQL sub-agent to set its own threshold for every question.

3.2.2 Routing

Once a question is divided this way, the conditions it holds determine which retrieval components answer it. Facts go to the MQL sub-agent and traits to the trait search, and where a question has both, the facts decide which players are candidates, and the traits decide which of them reach the answer and in what order. This decision could also be left to the MQL sub-agent, if it held the trait search as one more tool and chose step by step which tool a condition needs. Such an agent could revise a choice once it noticed that the choice was wrong, without the user having to step in, and it could decide for itself how to combine the two parts. It would need at least one more step for every question, since it has to see a result before it knows what to do next. Its context would also have to hold the distinction between facts and traits at every step, beside its work on the database. A router decides once, and the sub-agent after it can concentrate on the database. Schluntz and Zhang (2024) recommend the simplest solution possible. They name routing for inputs that fall into distinct categories, which are better handled separately and can be classified reliably. The three kinds of question are such categories, so the system puts a router in front, as Adaptive-RAG put a classifier in front of retrieval (Jeong et al., 2024). The router decides once for each question, from the question alone and before any retrieval runs, and the components that run afterwards do not revisit that decision.

Besides the route, the router divides a hybrid question so that each part receives only the conditions it can answer. A trait in the fact part would send the sub-agent looking for a stored value that does not exist, and a fact in the trait part would dilute the search, since the reports do not hold that information. Bai et al. (2026) observed a related cost on the side of the answer, where questions that one source alone answered could fail once the model received the evidence of both sources, because the redundant evidence distracted it. SUQL and ER-RAG likewise give each source only its own part of the question (Liu et al., 2024b; Xia et al., 2025). Route and division come from one output because both need the same reading of the question, and one output keeps them consistent, so each route comes with exactly the parts it runs on. Each condition goes into exactly one part, and numbers, names and comparisons are kept as the user wrote them. A role named as a type, such as "a number 6", is split into the position it is played in and the traits the role implies. A follow-up question is rewritten into an instruction that stands on its own and names the earlier result it refers to, so the components after the router receive an instruction and not a conversation.

The router's context holds its rules, examples of the division for each route, a list of what the database holds and the current date. The list is generated from the collections the MQL sub-agent can query, so the router knows which kinds of things the database holds, but not which values, and it can take a stored value that sounds like a judgement for a trait. The chat history reaches the router as a dialogue, as the earlier turns of its own conversation with the user, since resolving a follow-up is a task about that conversation. The fourth route lets the router ask the user back when a question cannot run as it stands, for instance a follow-up whose earlier result is missing. Asking back helps where a question leaves something open, since most failures of the agents on BIRD-INTERACT came from ambiguity they left unresolved, and their success rose with the number of turns in which they could ask (Huo et al., 2026). The evaluation switches this route off (Chapter 5). Scoring the route would need a person or, as in BIRD-INTERACT, a simulated user to answer the system's question. A question back is also rarely wrong in itself, since how often a system should ask is a matter of degree and of style rather than of right and wrong.

3.2.3 Dispatch

On the hybrid route the fact part runs first, and the trait search ranks only the players it returns. This is where the system combines structured data and text at the top level. The dispatch, fixed code that runs the route, asks the MQL sub-agent for every player who matches the facts, as a set of candidates and not as the answer, and passes the players' ids to the trait search in code, without entering the context of a model. If the fact part returns no usable players, because none matches or because no set of players came back, the trait search does not run. The hybrid route still ends in the synthesis call (Section 3.5), which in that case says what happened.

The literature combined the two kinds of data in three ways, (1) evaluating the conditions on the columns first, as SUQL does (Liu et al., 2024b), (2) passing the values one source returns into the query of the next, as ER-RAG does (Xia et al., 2025), and (3) handing the results of both sources to the model in one prompt, as the rule-driven router of Bai et al. (2026) did for a question that needs both. This combination takes from the first two. ER-RAG passes the values in a query language of its own, in which a query can read the sources in either order. A new language needs an executor for every source and examples from which the model picks it up. In this system, the fixed code behind the router passes the values, which needs neither. The third way puts both results into the model's context and leaves the model to combine them. Passing the model only a reference to each result would keep that context small, but the two results would then still have to be joined, by code or by a further model call.

This system combines the two parts in one direction only, from the players that the facts admit to the trait search that ranks them, since that is the combination a hybrid question usually needs. Fixed code gives up the flexibility of a query language. A question whose fact part depends on a trait, such as the average market value of the best pressers, cannot be answered in one run, since the trait search never passes its players back to the MQL sub-agent. A wrong route, or a condition that the router puts into the wrong part, cannot be repaired later either, since no loop returns to the router.

Chapter 5 tests how much accuracy the router costs each part behind it.

3.3 MQL Sub-Agent

The MQL sub-agent answers the fact part of a question, and its design follows two principles. The first is that the model sees as little as possible and as much as a step needs, which decides how the schema, the values, the sets and the results reach it. The second is that code guarantees what the model cannot be relied on to get right, which decides the joins, the access rights, the end of a run and what the memory keeps.

3.3.1 Planning Approach

The fact part of a question asks for whatever a club's data states, and the answer can lie in any of the 31 collections the system can query. Many questions need several collections, such as a question about the reports on the players of a watchlist. Which collection holds a fact and in what form becomes clear only by reading the collections, so a first query finds where a fact lies and how it is stored, and only the next query can use what it found. Within one question each query therefore usually depends on the result of the one before, and the sequence of queries can rarely be fixed before the first one runs. This is why the fact part goes to an agent that decides its next query after each result, in the manner of ReAct (Yao et al., 2023), which Huang et al. (2024b) call interleaved decomposition. The same dependence argues for one agent rather than several, since coordinating several agents lowered accuracy where each step depended on the one before (Kim et al., 2025b). Interleaved decomposition fits these questions better than a decomposition that fixes every sub-task in advance, because such a plan would have to guess what the data holds. Huang et al. also name multi-plan selection, which generates several plans or answers and chooses one of them, as self-consistency does by a majority vote (Wang et al., 2023). In text-to-query it raised accuracy at the price of many candidate queries for every question (Pourreza et al., 2025; Deng et al., 2025), and in an agent it added little while the tokens rose (Wang et al., 2025b). Even run in parallel, the samples would lengthen the wait for an answer, since the answer waits for the slowest run and the choice adds a step of its own. Multi-plan selection thus costs too much and takes too long for a scout waiting on an answer, so the agent does without it.

The text-to-query literature also asks whether such questions need an agent at all. In a fixed workflow, code sets the stages and their order, and the model only fills each stage. On Spider 2.0 and on TEND such a workflow did better than the agent it was compared with (Deng et al., 2025; Lu et al., 2026). Both workflows prepared the structure of the data before they generated a query, and both voted over several candidates, which each paper credits with part of its gain. Each paper, however, compared one workflow with one free ReAct agent that had no steps prescribed, and both papers are preprints, so neither comparison shows that a workflow is better than an agent in general. The MQL sub-agent therefore stays free to choose its next step, but its instructions tell it to read the structure of a collection before it queries it and to check the stored values before it answers. The agent thus takes the reading of the structure from the workflows but not the vote, and it takes the freedom to choose its steps from the ReAct agents.

3.3.2 Agent Loop

A run of the agent is a sequence of steps, and in each step the model chooses one of four operations, (1) reading a schema, (2) probing, (3) aggregating and (4) finalising. Each step consists of one sentence of reasoning and the operation with its arguments, and the result returns as the observation that the next step reads. Every step stays in the context of the run, failed ones included. The agent is told to read the schema of a collection before it queries it and to probe before it finalises, but it reads another schema or probes again whenever a result shows that it has to. The operations are described below along one run, shown in Figure 3.2, which answers the question "SV Werder Bremen squad for the 2020/2021 season, with each player's appearances".

Reading a schema. Reading a schema gives the agent the declared structure of the collections it names, and one step can name several collections at once. For each collection the agent receives its fields with their types, the references that link them to other collections and the fields that carry an index. Notes on how the collection is used and worked examples that show queries over it step by step come with the schema. In the example the agent reads the schemas of `teams`, which holds the clubs with their youth and reserve sides as separate entries, of `performanceSummaries`, which holds one row of season statistics for each player, team, competition and season, and of `players`, which it needs to name the players. The notes

Question: "SV Werder Bremen squad for the 2020/2021 season, with each player's appearances."

1 `read_schema`

I'll check how historical team membership and season appearances are recorded.

```
read_schema(
    collection=["teams", "players", "performanceSummaries"]
)
-> Mongoose schema for 'teams':
     externalId:  { type: String, index: { unique: true, sparse: true } },
     name:        { type: String, required: true },
     search_name: { type: String, required: true, default: '' },  ...
   Related collections:
     Referenced BY: ..., performanceSummaries.teamId, players.membership, ...
   Notes for querying 'teams':
     'search_name' is the FULL official name, lowercased and ascii-folded ...
   (likewise for players and performanceSummaries; about 11,500 tokens in all)
```

2 `probe`

I'll identify the senior Werder Bremen side so the season figures exclude its reserve and youth teams.

```
probe(
    collection="teams",
    filter={"search_name": {"$regex": "^sv werder bremen"}},
    fields=["name", "externalId", "ageCategory", "league"]
)
-> {"count": 9, "ids_ref": "teams#1", "sample": [
     {"_id": "teams@1", "name": "SV Werder Bremen Youth", ...},
     {"_id": "teams@2", "name": "SV Werder Bremen U19", ...},
     {"_id": "teams@3", "name": "SV Werder Bremen U17", ...},
     {"_id": "teams@4", "name": "SV Werder Bremen", "ageCategory": "Senior",
      "league": "Bundesliga", "externalId": "86"},
     {"_id": "teams@5", "name": "SV Werder Bremen U19 II", ...}]}
```

3 `aggregate`

I'll total each player's 2020/21 appearances across Werder Bremen's recorded competitions and link the figures to the players.

```
aggregate(
    collection="performanceSummaries",
    pipeline=[
        {"$match":  {"teamId": 86, "season": "2020"}},
        {"$group":  {"_id": "$playerId",
                     "appearances": {"$sum": {"$ifNull": ["$matchesPlayed", 0]}}}},
        {"$sort":   {"appearances": -1}},
        {"$lookup": {"from": "players", "localField": "_id", "foreignField": "playerId",
                     "as": "player", "pipeline": [{"$project": {"_id": 1}}]}},
        {"$unwind": "$player"},
        {"$set":    {"pid": "$player._id"}}
    ],
    columns=[
        {"field": "pid", "header": "Player", "kind": "ref", "collection": "players"},
        {"field": "appearances", "header": "Appearances 2020/21", "kind": "number"}
    ]
)
-> {"collection": "performanceSummaries", "total_rows": 31, "rows": [
     {"_id": "260053", "appearances": 39, ..., "pid": "players@1"},  /* 9 rows */
     "... 11 more rows, kept in the stored result ...",
     /* 9 rows */  {"_id": "196813", "appearances": 0, ..., "pid": "players@31"}],
   "rows_sampled": "only the first 10 and last 10 of 31 rows are shown, ...",
   "agg_ref": "agg#1", "ids_refs": {"pid": "players#1"}}
```

4 `finalize`

The season records provide each player's appearances and are ready to display.

```
finalize(
    answer="Werder Bremen's 2020/21 squad has 31 players with season records. Their
            appearances across all recorded competitions are shown below, including
            players with zero appearances. If you meant Bundesliga appearances only,
            let me know.",
    attach=["agg#1"]
)
```

Figure 3.2: A live run of the MQL sub-agent (GPT-6 Sol). Reasoning in grey, the model's input in blue and observations in black, quoted from the trace and shortened where marked. One of the worked examples the model received covers this question.

on `performanceSummaries` show that its rows name a team by the id that the data provider assigns, so the agent knows that it has to find this id for the club first.

Probing. A probe runs a filter that the model writes on one collection and shows the agent what the filter matched. It returns the exact number of matching documents and a few of them as a sample, which the model can restrict to chosen fields and order by a field it names. A handle such as `players#1` stands for the ids of all matching documents, as many as the count states, while the sample shows only a few of these documents. In the sample, the id of each document appears as a token such as `players@3`, which stands for that one document. A later filter refers to a handle to continue from all matching documents or to a token to continue from a single document, so a question about the reports on the players of a watchlist can be answered with three probes, one on the watchlist, one on the players it lists and one on the reports about those players. A filter can exclude a set in the same way, for instance the players already on a watchlist. In the example the agent probes `teams` for names that begin with "sv werder bremen". The count of 9 shows that the filter also matched the youth and reserve sides, and the sample shows which entry is the senior side and which provider id it carries.

Aggregating. An aggregation computes what a filter cannot, such as a count per club or a ranking. The model writes it as a pipeline of read-only stages, and a join to another collection returns only the fields the model asks for. When the model declares the columns of the result, the rows become a table that can be attached to the answer. A column that holds ids is declared as a reference to its collection, so the table names the players or clubs it lists, and a column of players can pass them on to the trait search on the hybrid route. The other columns are typed only so that the interface can show them directly, such as a number with the right decimal places. The result is stored under a handle as well, and a later filter can use the values of one of its columns, for instance the clubs at the top of a ranking. In the example the aggregation selects the rows of the senior side for the season 2020/21, sums the appearances of each player and sorts the players by them. A join to `players` adds each player's id in the database, and the columns declare this id as a reference to `players` and the appearances as a number. The 31 rows are stored under the handle `agg#1`.

Finalising. Finalising ends the run with an answer to the fact part of the question, which goes to the synthesis call of Section 3.5. The answer states what the agent found, and where it found nothing, it says what the agent tried and what the user would have to specify. The agent attaches the handle of the set or of the table the answer refers to, so the players and rows themselves never pass through the agent's model. In the example the answer states that the squad counts 31 players with season statistics, says that the appearances are counted across all competitions and offers to narrow them to the Bundesliga. The agent attaches `agg#1`, which the interface shows as a table of the players and their appearances.

Code checks each step before it runs, and the agent repairs a step when an error comes back. An error reaches it in one of two forms. The first form is a hint written in advance for a known mistake. Code catches some of these mistakes before the query runs, such as an operator the system does not allow. Others it recognises when the database aborts the query, such as a query that exceeds its time budget, and the hint then says to rewrite the query around the indexed fields. Each hint says how to correct the call, as Aizawa (2025) advise for the errors a tool returns. The second form is the database's own message for every other error, which says what failed but rarely how to correct it, so code catches as many mistakes as it can before a query reaches the database. Figure 3.3 shows an observation of each kind. In both cases the error is the observation of that step, and the next step can repair it. The agent relies on these signals because a model that revises its own output without an outside signal does not improve it (Huang et al., 2024a), while the result of executing a query is such a signal (Chen et al., 2024c).

(a) A hint written in advance, returned before the query runs: an aggregation joins `performanceSummaries` to `teams` on two fields that store different types.

```
-> {"error": "$lookup performanceSummaries.teamId (Number) -> teams.externalId (String): the key
   fields store different types, so the join would match nothing. Use a key pair that stores
   the same type, or join in two steps: probe teams into a set handle and filter with
   {"teamId": {"$in": "<that handle>"}}."}
```

(b) A hint written in advance, returned when the database aborts the query: an aggregation exceeds the time budget.

```
-> {"error": "the aggregation pipeline exceeded the 60s time budget and was aborted
   server-side. Work from the indexed fields -- marked ‘index: true‘ in the schema above -- and
   rewrite the query around them. Where the condition you need sits on an unindexed field, swap
   in an indexed one that means the same, or pair it with an indexed condition that cuts the
   set down first -- selective enough to leave thousands of rows rather than tens of thousands,
   or it buys nothing. Put the $match before any $lookup or $group. ..."}
```

(c) The database's own message, framed by code: a `$group` stage names a field without an accumulator.

```
-> {"error": "MongoDB rejected the aggregation pipeline: The field 'reports' must be an
   accumulator object. Fix it and retry."}
```

Figure 3.3: Three error observations as the model receives them, all generated by the system's code. Observation (c) carries a message of the kind MongoDB returns. Shortened where marked.

A run ends when the agent finalises or reaches a step limit. The limit guards against a loop that does not end. It also ends a long run, which on complicated tasks can drift from its goal (Huang et al., 2024b). A question rarely needs many steps. The questions of the evaluation (Section 5.1) need between three and six steps in a run without a single error, one to read the schemas, one for each collection the answer draws on and one to finalise. The limit is a high ceiling, which leaves room for corrections, so the agent does not work to it as a budget. Chapter 5 records how many steps successful runs need under this ceiling and shows whether a lower limit would cost little accuracy for the steps it saves. A run that reaches the limit gets one more call, which finalises with what the run has established so far.

3.3.3 Context Composition

The agent reads the schema of a collection when a step needs it, rather than receiving the whole schema in its prompt. The schema of the club's database is declared in code, so the agent can read it, whereas TEND's solver has to infer the structure from sampled documents (Lu et al., 2026). The first design put the whole schema into the prompt. With the notes and examples the agent receives for each collection, the whole schema of the 31 collections comes to about 45,000 tokens, and a given question needs little of it. Since that still fits into the context window, the text-to-query literature would give the whole schema, because for capable models the columns a schema linker wrongly dropped cost more accuracy than leaving out the irrelevant ones gained (Maamari et al., 2024). Studies of long contexts show that accuracy still falls as the input grows (Levy et al., 2024; Hsieh et al., 2024a). The design follows the long-context findings, since the product also has to stay cheap, and Chapter 5 tests whether the accuracy given up for that is noticeable.

The worked examples come with a collection rather than with a question, and the agent receives them even when they answer a different question. They show how a collection is used, such as which collection links two others. A correct query probably depends on that more than on the pattern of the query, since much of what makes MQL hard lies in the documents (Lu et al., 2026). Examples selected for each question, by the similarity of the question (Sivasubramaniam et al., 2024) or of the question and its expected query (Gao et al., 2024), worked better than fixed ones. Chapter 5 tests whether the agent answers fewer questions correctly without the examples and whether examples retrieved for each question serve it better than examples tied to a collection.

Knowing the structure does not tell the model how values are stored, and a club's own tags and rating fields differ from one club to the next. The literature first put a few sample rows of every table into the prompt (Rajkumar et al., 2022). Such rows cost no step, but they are not chosen for the question, so the value a question names is rarely among them. The agent probes instead, and a probe shows only a few documents. A count of tens of thousands shows at once that a filter is too broad, and a count of zero that the filter or the value it names may be wrong, and the sample shows how the values are actually written. Reading stored values before writing the final query is a step that the strongest systems on recent benchmarks shared (Talaei et al., 2024; Deng et al., 2025; Lu et al., 2026), and the agent is told to take it before every answer. This costs little, since a probe is often also the query that answers the question, and its match can be attached to the answer directly. How a value is written also depends on its language, and the question reaches the agent in the user's language, since nothing translates it. The shared football data is stored in English apart from a few fields in German, such as a player's preferred foot, and a club's own data can be in any language. The agent's instructions state this, and the notes of a collection name its German fields. Many facts are also stored in more than one collection, and these copies are not always updated together. The matches a player played, for instance, stand both in his own performance history and in the records of the matches, and the records of the matches miss one more often than his history does. The notes of a collection therefore say which copy to trust, and its worked examples show it by the copy they use.

A broad question can match thousands of documents, and putting all of them into the context would fill it with documents that no later step needs. The answer to such a question is the set of matching documents, which code can deliver without the model reading it, so a probe keeps its whole match in code under a handle and shows the model the count and a sample, with every id the database assigned replaced by a token. A handle compresses a result in the restorable way that Ji (2025) describes for the Manus agent, since it drops the content from the context and keeps a pointer to it. Handles and tokens follow the recommendation to keep identifiers in the context and load data only when a step needs it (Rajasekaran et al., 2025). Chaturvedi et al. (2026) found that restorable placeholders for the tool outputs of a coding agent raised its solve rate with some models and left it unchanged, at more tokens, with the others.

Of each result the model sees only the part its task needs. A generator in retrieval-augmented generation likewise answers from the passages that matter. Beyond 20 retrieved passages, further ones raise its accuracy little (Liu et al., 2024a), and passages related to the question that do not contain the answer lower it (Cuconasu et al., 2024). A probe's sample serves to check the filter, so a few documents beside the exact count suffice. The rows of an aggregation are the result the agent reports, and a row of a declared table holds only its columns, far fewer than the fields of a document, so a short result is shown whole, which keeps a ranking of the usual length complete. A longer result is shown as its first and last rows with the total and a note that the rows between them are left out, since a sorted result has its most telling rows at the top and at the bottom. The rule does not depend on the form of a result, so any further kind of result the system might return could be shown to the model in the same way, by a few telling parts and its total. A time series, for example, could be shown by its first, last, lowest and highest value.

A second limit, on length, catches what the limits on samples and rows leave too long, since they bound how many documents the model sees but not how long they are. A probe that asks for many full reports can still produce an observation longer than a display budget, a fixed share of the model's context window. Such an observation is not cut off inside a document, since the model could take a cut-off document for a complete one. Instead it is shortened in the same form as a long aggregation, to fewer whole rows at both ends or to the first documents of a sample, with a note that the budget left out the rest. When not even one row or document fits, the observation keeps only the exact count, the handle and a note on how to narrow the query. This follows the advice of Aizawa (2025) to say when a result is cut and to point the agent to a tighter query.

A scout usually follows a question with others about its result, so the context of a run also holds a memory of what earlier runs in the same chat session found. For each earlier run, the memory keeps the sets the run created, the schemas it read, and a note that records the task with the steps it took and the answer. The note keeps the steps as they were taken rather than a summary of them. On a benchmark of assistant tasks, the raw record of earlier steps within one run served the next steps at least as well as summaries of them and cost less (Wang et al., 2025b), and the note carries that finding over to later runs. The memory covers the whole session, since a follow-up can refer to any earlier answer. Condensing the memory of a session that outgrows the window lies outside the scope of this thesis, although a memory of only the relevant turns may serve a follow-up better than the whole session (Hong et al., 2025).

3.3.4 Deterministic Safeguards

Of the things code guarantees, two need safeguards of their own, (1) joins between ids stored in different forms and (2) access rights. A field that refers to another document stores that document's id, depending on the collection, either as an ObjectId, the database's own id type, or as a string. A report, for example, names its player by a string, while the player document holds its id as an ObjectId. The application reads the data through Mongoose, a JavaScript library that converts between the two forms in most of its queries by the declared schema, so the inconsistency rarely shows there. The agent's queries reach the database without that library, and the database never treats a string as equal to an ObjectId, so a filter or a join across the two forms matches nothing without raising an error. Such an empty result looks the same as a question that has no answer.

Code therefore takes over the conversion that the library does for the application. A handle or a token in a filter is expanded into each of its ids in both forms, so the filter matches whichever form the field stores. For a join, code reads the declared types of the two key fields and converts the key of the collection the join starts from into the form of the joined collection, so the model writes the join as if both stored the same form. Ids from data providers, which some collections store as numbers and others as strings, are not converted, and code returns the hint of Figure 3.3(a) instead.

Access rights follow from the requirement of Section 3.1, and the model's filter is never trusted to include them. Code adds the restriction to what the user may see to every query, including every join to another collection. Sensitive fields, such as passwords or the email address of a report's author, are removed from the collection schemas for every user, so the model never sees them. A survey of security in text-to-SQL systems reports that language models do not reliably keep to the access rights of a user, and it names as the most lasting defence a design in which the model can reach only what the user may see, the principle of least privilege (Bui et al., 2026). A restriction composed in code therefore holds even for a query the model gets wrong. Since the conversion of id forms and the restriction of access are code rather than behaviour of the model, they are tested by tests of the code rather than in the evaluation.

3.4 Trait Search

The trait search receives the trait part of a question and, on the hybrid route, the players that the fact part returned. It returns players ranked by how well the reports support each trait, together with the evidence for each player. The ranking serves exploration. A scout uses it to find every player who fits a profile before he decides whom to look at more closely, so it should leave none of them out. He can check a player who is shown by reading his reports, but not a player who is left out. The evidence lies in the scouting reports. A report consists of the written report itself and of report fields that the club defines. Some report fields hold text, as the written report does, and others hold a number, such as a grade for heading. The text and the numbers can both speak about a trait, and the trait search reads both.

Reading both raises the questions that Section 3.2.1 left open, (1) which fields count toward a trait, (2) how the text weighs against them and (3) where good begins. The trait search settles all three by a fixed procedure and does not leave them to the MQL sub-agent, since a threshold that the agent set for each question could differ between questions and between runs, so that the same evidence could rank a player differently from one question to the next. The steps of a trait search are also the same for every question, and for such well-defined tasks Schluntz and Zhang (2024) recommend a workflow, which is predictable and consistent where an agent is flexible.

3.4.1 Design History

The first version of the trait search was plain document retrieval. The trait part of the question went into one search, which returned the reports that matched it best. Since every report is about one player and is linked to him, a ranked list of reports seemed to be a ranked list of players already. The version also read only the text, although scouts record part of the same judgement in the rating fields of a report, so that the evidence for a trait lies in text and in structured data at once. The rating fields joined the search later (Section 3.2.1), and three problems of the text search led to its present form. Each of these problems shows how ranking players by traits differs from retrieving documents.

The first problem was the unit of the result. A list of reports ranks each report on its own, while the evidence for one player lies spread over the reports of different scouts, so a player with one well-matching report could rank above a player whom many reports described. Ranking people by what documents say about them is the task the retrieval literature calls expert finding, a form of entity retrieval in which the results are people (Balog et al., 2012). Its document model scores the documents on a topic and adds the scores up for each person associated with them, and it outperformed the alternative of putting all documents about a person together into one text and matching the question against that text (Balog et al., 2006). So the search began to group the reports it found by player and to add up their scores, and no single report had to hold all of the evidence for a player.

The second problem was the direction of the evidence, since the search ranked a report by how well it matched the topic of the question. A question for players who are "good in the air" asks what scouts say about that quality, and a report that calls a player weak in the air is about the same topic as one that calls him strong. Retrieval scores how much a text is about the quality asked for, which the literature calls topical relevance. Topical relevance is the standard case in retrieval, and even the opinion retrieval task of the TREC Blog track judged a post relevant whatever the direction of its opinion (Ounis et al., 2006). Retrievers barely tell a text apart from its negation (Weller et al., 2024). The search therefore split the work into two steps. Retrieval only finds the chunks that are about a trait, which is what it is built for. A language model, the judge, then reads each chunk together with the trait and decides whether the chunk speaks for or against the trait and how strongly. A report that calls a player weak in the air thus counts against him rather than for him. In this, the judge's task is closer to aspect-based sentiment analysis than to a ranking by relevance. Aspect-based sentiment analysis does not give a text one sentiment as a whole but determines the sentiment the text expresses toward each aspect it speaks about (Pontiki et al., 2014). A trait is such an aspect of a player, and the judge determines the sentiment of a chunk toward that one trait. So a report that praises a player in general still counts against his heading if it says that he rarely wins a header.

The third problem appeared with questions for several traits, and it arose from the first. A question for players who are fast and good at pressing rarely finds a report that speaks about both, and it does not need one, since a player can show his pace in one report and his pressing in another. Because the evidence for a player crosses reports, a question with several traits must not be answered by the reports that cover all of them. The version at that point searched each trait with query variants of its own, but it fused the result lists of all variants by rank into

one list of reports and grouped that list by player only afterwards. Its judge read each report against all traits together, which is a harder task than judging one trait, and it judged each report on its own rather than the evidence across a player's reports. Since rank fusion rewards what several lists agree on, the top of the fused list held reports that mentioned several traits at once rather than the players strongest on each.

The present version therefore divides the trait part into its traits first and runs the same search for each trait on its own, with its own grouping by player and its own judge, which reads the chunks against that one trait alone. The query variants remain, but they now only rephrase a trait and no longer separate the traits. The traits are combined only after that, for each player rather than for each report, so a player can be strong on one trait in one report and on another trait in a different report. Korikov et al. (2024) found that scoring each aspect on its own and combining the scores beat one search for the whole query when each review spoke of one aspect, as a short chunk of a report mostly speaks of one trait. One model call divides the trait part of the question into its traits. It keeps the traits as distinct from each other as possible and merges aspects for which the same evidence speaks, such as heading and winning aerial duels, since one quality divided into two traits would weigh twice in the ranking and the traits would no longer count evenly.

Ranking players by traits thus differs from retrieving documents in four ways. (1) The results are players, whose evidence is added up across reports. (2) What counts is the sentiment of a chunk toward a trait and not only whether the chunk is about the trait. (3) A question holds several traits, which are searched separately and combined for each player. (4) The evidence for a trait lies partly in text and partly in rating fields, so both are read for the same trait.

3.4.2 Text Evidence

The trait search embeds the text of a report, the written report and each text field on its own, in short chunks of one or two sentences. In a scouting report a trait is mostly described in a single sentence, and scouts say much in little text, so a longer chunk would mostly add statements about other traits. A single dense vector also represents a longer text less faithfully, and retrievers that embed a text as one vector lost accuracy as the texts grew longer (Luan et al., 2021). Smaller units retrieved about as well on the data a retriever was trained on and clearly better on data it had not seen (Chen et al., 2024b). Scouting reports are such unseen data for any public embedder. The rating fields are not part of this search, since they are the second kind of evidence (Section 3.4.3) and would otherwise count twice for a trait.

The scouts of a club write in their own languages, so a question in one language has to find reports in the others (Section 3.1). BM25 could match them only with a separate index for each language, since its tokenisation and stemming depend on the language of a text (Robertson and Zaragoza, 2009), and only after the question had been translated into every language present in the reports. A multilingual embedder places the texts of all languages in one space, so one index serves every language and a question needs no translation. BGE-M3, a multilingual embedder, was evaluated on search within one language and on questions in one language against passages in another, and neither benchmark mixes languages in one corpus (Chen et al., 2024a). That one index also ranks reports in different languages against each other is therefore an assumption, and the evaluation tests only German questions against English reports (Section 5.3.6). The chunks are searched in that space as approximate nearest neighbours over a navigable graph of the chunks (Malkov and Yashunin, 2020), which finds the most similar chunks without comparing the question with every chunk. Combining BM25 with the embedder would probably retrieve better than the embedder alone, as adding the learned lexical score of BGE-M3 to its dense score did when question and passage shared a language. The combination would, however, bring back the separate indexes and the translation, so the trait search uses the embedder alone. The embedder therefore has to be multilingual, trained to follow a retrieval

instruction and small enough to run on the servers of the deployment, and Section 4.2 describes the one chosen.

On the side of the question, the embedder receives an instruction that asks for report passages which discuss the qualities a scout asks for, and not for passages which praise them, so that chunks speaking against a trait are found as well and the judge can count them against a player. The embedder was trained with such instructions and needs them to retrieve well (Microsoft, 2026a). Each trait is also rewritten the way a scout would write about a player, since a text in the register of the corpus can retrieve better than the question itself (Gao et al., 2023a). For an embedder trained with instructions this rewriting is an addition rather than a necessity, and it costs only a few more output tokens of the call that divides the question. That a rewritten trait reads like praise should not hide the chunks that deny it, since retrievers barely tell a statement from its negation (Weller et al., 2024) and a chunk that denies the trait in its own wording is still about the trait.

The call that divides the question rewrites each trait twice, once close to the words of the question and once in other words. These two query variants follow RAG-Fusion, which searches several versions of a question that a language model writes (Rackauckas, 2024), and their result lists are fused by rank (Cormack et al., 2009). A chunk c receives the score

$$s(c) = \sum_{v} \frac{1}{k + \mathrm{rank}_v(c)}, \tag{3.1}$$

summed over the query variants v of the trait, where $\mathrm{rank}_v(c)$ is the place of the chunk in the result list of v and a list that misses the chunk adds nothing. The constant k keeps the first places of one list from deciding alone, so a chunk that both variants place well rises above one that a single variant places first. Scouting prose lies outside what public embedders were trained on, so a small embedder is uncertain there, and a second wording smooths that uncertainty. Fusing query variants raised term-based retrieval on average but lowered it for some questions (Benham and Culpepper, 2017). Cormack et al. fused the rankings of many retrieval systems, and Benham and Culpepper fused phrasings that people wrote for a term-based retriever. Fusing two phrasings that a model wrote for one embedder is therefore a transfer of their technique. Since an embedder keeps two wordings of one trait close in meaning, the two variants probably gain less than in the study of Benham and Culpepper and risk less as well. Chapter 5 tests whether the second variant helps against a search with the first variant alone, and reports beside it how gains and losses balance across the questions.

The judge is one model call for each trait. It receives the trait and the retrieved chunks as one numbered list and scores every chunk on its own. The scale runs from -5 for a decisive weakness through 0 for a chunk that says nothing about the trait or mentions it without leaning either way to $+5$ for an outstanding strength, and each step carries a description. So each chunk gets a direction and a strength, and an order of the chunks would give neither. A listwise reranker such as RankGPT returns such an order and inherits the order it was handed (Sun et al., 2023), so the judge receives no retrieval rank. The list also names no player, club or report, so that what the model knows about a name cannot replace what the chunk says. A chunk from a text field carries the heading of its field, since a sentence under the heading "Weaknesses" can describe a weakness without naming it as one. Keeping one trait in one call lets the judge score all chunks of that trait against the same reading of the scale. Chapter 5 tests the judge by switching it off.

The judge reads only what retrieval places at the top. The players are ordered by the summed score from (3.1) of their m best chunks, and for each trait the judge receives these chunks of the first N players. This budget spreads one call over as many players as possible, since retrieval has already ordered the chunks by how much they are about the trait. A player behind the first N, or a chunk beyond a player's best m, is never judged, even where it speaks clearly against the trait. The trait search therefore depends on retrieval placing the chunks about a trait first.

That is the topical relevance embedders are built for, and the judge decides only the direction. Evidence against a trait can thus remove a player only if retrieval places it within this budget.

For each player the scores of the judged chunks are added up, as the document model adds up the evidence of all documents about a person (Balog et al., 2006), so more evidence counts more. The sum places the player in one of three states for the trait. A sum of at least $+1$ is (1) evidence for the trait, a sum of 0 or -1 is (2) no evidence, and a sum of -2 or less is (3) evidence against it. The line lies at -2 because a single hedged doubt, which the judge scores -1, cannot reach it alone. Only the chunks scored above zero remain as the evidence shown for the player. The text is read on this fixed scale, while the rating fields are read by a player's standing among the rated players (Section 3.4.3), because a scout who writes about a player already judges him against the players of his position. A winger called good in the air is good by the standard of wingers, not of centre-backs, so the comparison group that a grade needs is already part of the sentence. Section 3.4.4 describes how the three states change the ranking.

3.4.3 Rating Evidence

The rating fields work on the same division into traits as the text, and they are of two kinds. Ordinal fields hold a value on an ordered scale, mostly a grade, such as a field "Heading" graded from 1 to 6. Nominal fields list options without an order, from which a scout ticks those that apply, such as a field "Strengths" or a field "Weaknesses" with the options "Heading", "Pace" and "Passing". One model call matches, for every trait of the question, the fields and options that rate it from their labels. The model is instructed to leave a field out when it is unsure, since a missed field costs little while the text still covers the trait, whereas a wrong field ranks every player by something that does not measure the trait. For "good in the air" it takes the ordinal field "Heading" as a whole, since an ordinal field declares whether high or low grades are better. It also takes the option "Heading" of the field "Strengths", and a player's evidence there is the share of his reports with a filled field "Strengths" in which the scout ticked it. The search counts a ticked option toward the trait, since it is built to find the players who are good at it, so the model is instructed to take only options whose tick says that the player is good at the trait. A tick at "Heading" under "Weaknesses" is evidence against the trait, and since the trait search removes players on evidence against them (Section 3.4.4), it would belong in the search as well. It is not used, which is a gap of the present design. Closing it would need the model to name also the options that count against a trait and the search to treat a tick at such an option as evidence against the player.

Once the fields of a trait are known, each of them gives a player one value. On an ordinal field this value is the average of his grades, and on a nominal field it is the share of his reports with a filled field that tick the option. An average assumes that the steps between grades are equal, which a scale that only orders does not promise, and Stevens (1946) accepts it nonetheless as a pragmatic sanction where it leads to useful results.

Such a value is the least reliable when it rests on few ratings, since extreme values come most often from few observations (Gelman and Price, 1999). Each player is therefore ranked on a field by the cautious end of an interval around his value rather than by the value itself, so that consistent evidence from many reports outranks a perfect value from one. For the share of an option, the interval is that of Wilson (1927), which serves small samples well (Brown et al., 2001). In return the ranking favours players who were scouted often.

For a grade the interval lies around the player's average, and it narrows the more his grades agree and the more of them there are. Agreement on a grade is never quite exact, however. A grade forces a whole number, so a scout who sees a player somewhere between 5 and 6 writes the step closest to his view, and the grade records that step rather than the view itself. Two scouts who both write a 6 may therefore see the player differently, one at 5.6 and one at 6.3, and the grade hides that difference. Taken as it stands, the agreement of two reports with a 6 each would show no variation at all, the interval would vanish, and the two reports would

count as certain. The variation of a player's grades is therefore never taken as smaller than the choice of the closest step alone causes, which for views spread evenly within a step is about three tenths of a step (Widrow et al., 1996). With n grades of average $\bar{g}$ and standard deviation σ on a field whose smallest gap between two grades is Δ, a player is therefore ranked by

$$b = \bar{g} - z \, \frac{\max(\sigma, \Delta/\sqrt{12})}{\sqrt{n}}, \tag{3.2}$$

where z sets how cautious the end is, and on a field on which lower grades are better the sign of the second term is reversed. A player with few grades thus keeps an honest interval even where all his scouts agree.

Whether a value is good is a further question. A grade has no known neutral point, so a 4 may be routine praise on one field and a concern on another, and what counts as good depends on the player's group, since wingers are fast as a rule, and a grade for pace that is ordinary among wingers would stand out among defensive midfielders. Good and bad are therefore read from where a player stands among the players rated on a field, as scouting practice compares a player with the players of his position (Sports Reference LLC, 2026; Whitmore, 2023) and as player ratings were found meaningful only between players of similar roles (Pappalardo et al., 2019). On a hybrid question the group is the set of players that the fact part returned, so a question for wingers compares wingers with wingers. A player's standing on a field is his percentile in this group, 1 for the best and 0 for the last, taken over the cautious values of all rated players. Percentiles also put every field on the same scale, whether it holds a grade from 1 to 6 or a share between 0 and 1, so that the fields of a trait can be combined.

A trait often matches several fields, and their standings are combined into one standing for each player and trait, an average weighted by the number of his ratings on each field, or of his reports that hold the field for a nominal field. A field rated forty times thus weighs more than one rated twice. The match of the model is a yes or a no, so a field that is only related to a trait is left out entirely. A natural extension would let the model grade how closely each field measures the trait, so that a related field counts a little instead of not at all and a closely matching field counts more. The search does not grade its matches yet.

The combined standing decides whether the ratings speak for or against the trait, and the cut is made only after combining, since a cut for each field would let the number of matched fields decide how often a player risks removal, with every further field another chance to fall into its bottom part. A combined standing above an upper cut is evidence for the trait. Evidence against needs a second reading, over the ordinal fields alone, below a lower cut, since a scout who does not tick a strength does not claim that the player is weak at it. A low share of an option is therefore no evidence against a player, whereas a low grade is a scout's actual judgement. As on the text, evidence against needs a stronger signal than evidence for, for the reason given in Section 3.4.4.

3.4.4 Evidence Combination

Section 3.1 named two levels at which the system combines structured data and text, and the two are nested. At the top, the router and the dispatch combine the MQL sub-agent with the trait search, and what the trait search hands over already combines both kinds of data. That inner combination happens here, where the text evidence of Section 3.4.2 and the rating evidence of Section 3.4.3 are combined.

The hybrid systems of the literature choose a source for each question or each condition, or they hand the results of both sources to the model. SUQL answers each condition of a question either from a column or, through a call to a language model, from the text (Liu et al., 2024b), and Bai et al. (2026) route a question to the database, to the documents or to both, because the two sources complement each other. The system here divides a question by what a condition asks for rather than by where its data is stored (Section 3.1). A trait is a quality that the scouts both

write about and rate, so the text and the rating fields speak about the same quality. Choosing one source per condition or handing the results of both to a model therefore does not transfer, and the trait search needs a rule of its own that combines the two as evidence for one trait.

The combination inside the trait search runs first within each trait and then across the traits of a question, which yields one ranking of players. Each kind of evidence places a player in one of three states for a trait, (1) evidence for it, (2) no evidence and (3) evidence against it, and the states decide how he enters the ranking. The TREC Health Misinformation track labels the stance of a document with the same three states, supporting, neutral and dissuading (Clarke et al., 2021), whereas here they describe a player and one trait. A question with several traits describes a profile, and the search decides whether a player is worth considering for it. No evidence on a trait does not say that the player lacks the trait, only that the reports do not show it. He may still fit the profile, so he stays in the ranking as long as he meets more than half of its traits, although below the players whose evidence covers more of them. Evidence against a trait does say that he lacks it. A fast player whose reports call him weak in the air does not fit a profile of fast players who are good in the air, so evidence against any trait removes the player from the whole result.

Evidence against a trait can be outweighed only by evidence about the same trait. Where one kind of evidence speaks against a trait and the other speaks for it, the two cancel to no evidence, since a contradiction between the scouts' words and their grades leaves open whether the player has the trait. Across traits nothing is balanced, and however good a player is on his other traits, evidence against one of them removes him.

For the text evidence as for the rating evidence, evidence against a trait needs a stronger signal than evidence for it, because the two errors cost differently. A wrong verdict of no evidence costs one kind of evidence for one trait, and the player can still be found through the other. A wrong verdict of evidence against takes the player out of the whole result. In the literature, evidence against a document only lowers the score of a ranking that retrieves it (Gienapp et al., 2020; Clarke et al., 2021), and a low rating of an entity only lowers its grade (Ganesan and Zhai, 2012). None of these collections removes an entity on negative evidence, whereas the trait search removes a player on it.

Within a trait, the text evidence and the rating evidence are two views of the same question, and two views may only add to each other. Only players with evidence for the trait on at least one view enter this step, those whose text passed the judge's threshold and those whose ratings passed the upper cut. Each view orders the players it admitted for the trait, the text by the sum of the judge's scores and the ratings by their combined standing, and gives each player a rank score by his place in that order, with 1 for the best. Every rank score stays above 0, since a low place is still positive evidence for the trait, only less of it than the others have, and does not mean that the player is poor at the trait. The two rank scores are joined by noisy-OR,

$$s_T = 1 - (1 - t)(1 - r), \tag{3.3}$$

where t is the player's rank score from the text and r the one from the ratings, and a view without evidence counts as 0. Clarke et al. (2008) derived this form for probabilities, the relevance of a document from independent units of evidence. The rank scores are places in an order rather than probabilities, and the form is used for three properties it has for any values between 0 and 1. (1) A missing view adds nothing and costs nothing, which an average does not offer. (2) One strong view suffices, and (3) two agreeing views raise the score above either alone, so a player at 0.6 on the ratings who also has text evidence at 0.5 scores 0.8, above a player at 0.6 on the ratings alone, which the maximum does not offer. Since every rank score is positive evidence, a second view always adds to the first, but it can only close part of the gap the first leaves.

Across traits the rule is the opposite, since the traits of a question are separate requirements and a missing one has to cost. Players who meet more than half of the traits are ranked by the

geometric mean of their trait scores,

$$S = \Big(\prod_{T} \max(\varepsilon, s_T)\Big)^{1/|\mathcal{T}|}, \tag{3.4}$$

taken over the set $\mathcal{T}$ of the question's traits, in which a trait without evidence counts as the small constant ε, below every rank score, so the weakest evidence on a trait still counts more than none. The geometric mean is used for two properties. (1) A strong trait cannot make up for a missing one, so a player at 0.9, 0.9 and without evidence on a third trait ranks below a player at 0.6 on all three, which the arithmetic mean does not offer. (2) Every trait still counts, so players who miss a trait are still ordered by their other traits, which the minimum does not offer, since it reads only the weakest trait. Korikov et al. (2024) found that combining aspects by their minimum fell to near zero once more than one review per aspect was averaged, while the arithmetic, the geometric and the harmonic mean scored alike. Their results thus do not decide between the means, and the geometric mean is chosen for its two properties.

3.5 Answer Synthesis

Neither the fact part nor the trait part returns a reply, only results. The MQL sub-agent states what it found without phrasing it as a reply, so that the rules on how a reply speaks stay out of its context, where they would not help it choose its next query, and the trait search returns a ranking of players. One synthesis call therefore writes the reply on the fact, the trait and the hybrid route. The router answers the clarification route itself, since only it knows what is unclear and nothing was retrieved that a reply could present, and it follows the same rules on voice and language as the synthesis call.

A reply can fail in three ways that do not depend on the route, and the call's instructions guard against each. (1) The question is a small part of the call, beside instructions in English and results that are often in another language or full of foreign names, so a reply can drift into the language of the data. The router therefore names the language of the question, and the call is told it explicitly. (2) The model knows many players from its training, and what it added from there would read like a result. Every claim therefore comes from what the call received, and counts stay as the results state them, since the reply is read next to the table that shows them. (3) An empty result looks the same whatever caused it, while the user can only rephrase the question once the cause is known. The call knows the cause only from what it received, the sub-agent's account of what it tried or a trait search that found no evidence, so the reply says that and guesses nothing beyond it.

What the reply says depends on the route. The interface always shows the players and records below the reply, as tables or as cards. The card of a player found through the reports also carries the excerpts and ratings that speak for him. The reply therefore points to them rather than retelling them. On the fact route the reply restates the sub-agent's answer in the user's language without changing its content. When the sub-agent ends by asking the user to narrow the question, the reply passes that question on. On the trait route the reply says what was searched and how many players matched, and points to the cards. The reply does not rank or judge the players itself, since the call never sees the excerpts the trait search found, and so it cannot attribute an excerpt to the wrong player either. On the hybrid route the call receives the results of both parts, and it is the last place where structured data and text meet. It does not decide how they are combined, since the dispatch has already narrowed the players to those that match the facts (Section 3.2.3). What remains is how the combined result is told. The number of players that match the facts is the first stage, and the players with evidence for the traits are the answer. Where the fact part returned no usable players, the call receives a note that the trait search did not run, and the reply says so.

4 Implementation

Chapter 3 describes the system by its rules. This chapter describes the environment the system runs in and what it is built with. It then takes the components in turn, how the stages of each are instructed, how each is implemented where that is not obvious and which settings it runs with.

4.1 Environment

The system is built to run inside the SCOUTASTIC platform, and the platform gives every club an installation of its own. A club's installation runs on a virtual server of its own, typically with 8 processor cores, about 30 GB of RAM and no GPU, the processor that neural networks usually run on. The server already runs the platform, as containers that Docker Compose starts, among them the platform's web service, a MongoDB 8.0 database with the club's data, a cache, monitoring and backups. The agent runs beside them as one more container, built from the same container image as the platform's service, and uses the same RAM and processor cores as they do.

Everything that reads the club's data runs on that server, the database, the backend with its agent, the trait search and its embedder. Only the calls to the language models leave it, to the models that Microsoft Azure hosts. Such a call carries what the stage needs, the question, the schemas and the documents and rows the MQL sub-agent reads, and the chunks of reports the judge scores. The reports and ratings a club stores are personal data under the European data protection law, the GDPR, so the models are served from Azure data centres in the European Union.

Without a GPU the embedder runs on the server's ordinary processor, where a neural network runs much slower, and that bounds its size. The embedder of the system has 0.6 billion parameters, about 2.4 GB of RAM in 32-bit numbers, and on a typical club server about 18 of the 30 GB are already in use, and more under heavy load, such as large batches of requests to the platform's API. The embedder is therefore loaded on the first trait question and unloaded after five idle minutes, so a server that answers no trait question holds no model in RAM. A club's archive ranges from a few hundred to about 200,000 reports, and embedding all of them on the processor takes long. The first start of the backend therefore embeds the reports of the last three years in one long run. After that a scheduled job embeds only the reports written or changed since its last run and removes the chunks of deleted reports. Each chunk records the settings by which the report was split into chunks and embedded, and a change of these settings embeds every report again.

The agent was first built as a prototype outside the platform, with a chat interface of its own and a copy of a club's database in a local MongoDB that includes its search engine, and this prototype is the system the thesis describes and evaluates. The agent was then built into the platform, where its answers appear in the platform's own chat page, with the platform's cards for players, clubs and transfers.

4.2 Stack

Every model call of the system goes to a language model that Microsoft Azure hosts. Of the models Azure offers, two of one family are considered for this system, `gpt-6-sol` and the smaller `gpt-6-luna`, both in their versions of 22 September 2026. The rest of the thesis calls them Sol and Luna. Both are served in the European data zone of Azure, and both lie on the frontier of

capability against the cost to run the Intelligence Index of Artificial Analysis (Artificial Analysis, 2026), which is why they are considered for the system. Artificial Analysis is an independent company that runs the same set of evaluations on every model itself, and the Stanford AI Index takes its prices of language models from it (Maslej et al., 2025). Its measurements are not peer reviewed, but no peer-reviewed source measures the models of one release on capability and cost together. Both come from the same release, so a comparison between them compares a larger and a smaller model of one version. Each model has a context window of one million tokens. The configuration sets for each stage which of the two serves it, and every stage runs at a high reasoning effort.

The trait search uses harrier-oss-v1-0.6b (Microsoft, 2026a), which is level with the strongest embedders of its size on the multilingual retrieval benchmark MTEB(Multilingual) (Enevoldsen et al., 2025) and with one more than ten times its size, and which is small enough to run on the servers of the deployment (Table 4.1). The slightly smaller geevec-embeddings-1.0-lite is level with it, but its card says nothing about how it was trained (GeeVec, 2026). harrier-oss-v1-0.6b builds on Qwen3 (Microsoft, 2026b), whose embedding models and their training a technical report describes (Zhang et al., 2025b), and it scores above the Qwen3 embedder of the same size.

Table 4.1: Embedders on the 18 retrieval tasks of MTEB(Multilingual, v2).

Model	Parameters (billions)	Retrieval mean
geevec-embeddings-1.0-lite	0.4	70.9
Qwen3-Embedding-8B	7.6	70.9
harrier-oss-v1-0.6b	0.6	70.8
Qwen3-Embedding-0.6B	0.6	64.7
GritLM-7B	7.2	58.3
multilingual-e5-large-instruct	0.6	57.1
BGE-M3	0.6	54.6

The retrieval mean is the mean of the tasks' main scores, as on the benchmark's public leaderboard, and the main score is nDCG at rank 10 on 16 of the 18 tasks. The values are computed from the benchmark's results repository (Enevoldsen et al., 2025) at commit fe1eb05. The table lists the two leading embedders with at most 1 billion parameters, Qwen3-Embedding at two sizes, GritLM-7B from Table 2.3, BGE-M3, and multilingual-e5-large-instruct, the instruction-tuned version of the multilingual-e5-large that BGE-M3 was compared with on MIRACL. Embedders not listed reach up to 79.9. BGE-M3 is from Chen et al. (2024a) and multilingual-e5-large-instruct from Wang et al. (2024).

The database of the system is the club's MongoDB. One database holds the club's data, the chunks of the reports with their vectors and the chat sessions. The vectors are searched by MongoDB's own vector search, which builds the navigable graph of Section 3.4.2 over the chunks. The backend reaches the database through MongoDB's Python driver PyMongo.

The backend is written in Python with the web framework FastAPI and runs the router, the dispatch, the MQL sub-agent, the trait search and the synthesis call. These are written directly on the OpenAI SDK and use no agent framework such as LangGraph, so the router, the dispatch and the loop of the sub-agent are plain Python. Every output schema is a Pydantic model, from which the JSON schema passed to the provider is generated and against which every answer is validated. The embedder runs with the library sentence-transformers. The MQL sub-agent reads the collection schemas from the Mongoose files of the scouting platform, which the backend holds as a copy, so it reads the structure the platform declares.

The chat interface is an Angular application. It keeps the chat sessions of a user, shows the players and tables below each reply, and offers an inspector that shows the transcript of every run exactly as the model received it.

4.3 Components

Every stage of the system is a call to a language model, and each is instructed through two texts the model reads together, a system prompt and the descriptions of the fields in its output schema. The system prompt states the task, its rules and its examples, and the output schema states the shape of the answer, which field holds what and when a field stays empty. The system passes each output schema to the provider as the required format of the answer, and the provider holds the answer to it by constrained decoding (OpenAI, 2024), so an answer cannot leave out a field or add one. The steps of the MQL sub-agent are the one exception, since a query's filter has no fixed shape (Section 4.3.2). Within the system prompt the examples stand in a block of their own, apart from the rules. Appendix B names the file that holds each prompt in the code repository.

The code follows the same division. Each component has a folder, and each stage that calls a model is a folder of its own with the same four files, `prompts.py` with every text the model reads, `llm_schema.py` with the output schema, `config.py` with the settings and `core.py` with the logic (Figure 4.1). A stage whose result has a type of its own declares that type in `result_types.py`. The MQL sub-agent is the one stage with a loop, so its logic is split into the loop in `agent.py` and the single step in `planner.py`.

The subsections below also give the settings each component runs with, the fixed numbers in its configuration, such as how many steps a run may take or how many documents a sample shows. The thesis asks how the system is designed and not which settings serve the design best, so the settings are not a subject of their own. None of them is tuned against the evaluation either, since the only questions with known answers are those of the test collection (Section 5.1) and settings tuned on them would fit the system to the questions it is scored on. A setting therefore comes from one of three sources, (1) a default left unchanged, (2) a limit of the software, or (3) a choice made for the design of this system, which was not measured against other values.

4.3.1 Router and Dispatch

Prompt. The router is one call whose output holds the route, the fact part, the trait part and the language of the question, so route, division and language come from the same reading of the question. The router sees only the question and none of the data, so its prompt has to give it a sense of which questions belong to which route. It defines each route by the kind of question it takes and names the questions that look like that route but belong to another. Each route is followed by a block of examples that show the filled output, with the fact part and the trait part beside the route, since the route alone does not show how a question is divided. Examples in the prompt raised accuracy over an instruction alone on most of the tasks GPT-3 was tested on, many of them a choice among fixed answers (Brown et al., 2020), and they cost the router few tokens, so it keeps them without a test of its own (Section 5.1.3).

Section 3.2.1 draws the line between a fact and a trait by whether the data states what a condition asks for. The prompt carries out that line by defining a trait narrowly, as a quality of a player that the system has to judge from what scouts wrote and rated, and counting every other condition as a fact. A trait is the specific kind of condition and a fact the broad one, so the prompt spends its care on recognising traits, and a rule for doubtful cases sends a question that stored fields alone could answer to the fact route. Whether a question can be answered at all depends on what the database holds. The prompt therefore holds a summary of the schema at the level of its collections, one line each, without their fields. The summary is generated from the collections the MQL sub-agent can query, so the router's picture of what can be answered matches what the sub-agent can reach.

Dispatch. The dispatch is a function per route. On the hybrid route it has to turn the sub-agent's answer into a set of players. Where the answer attaches a set of players, that set is the list of candidates. Where it attaches a table, as a ranking does, the candidates are the players in the table's column that refers to players. The candidates' ids then become the ids of their reports,

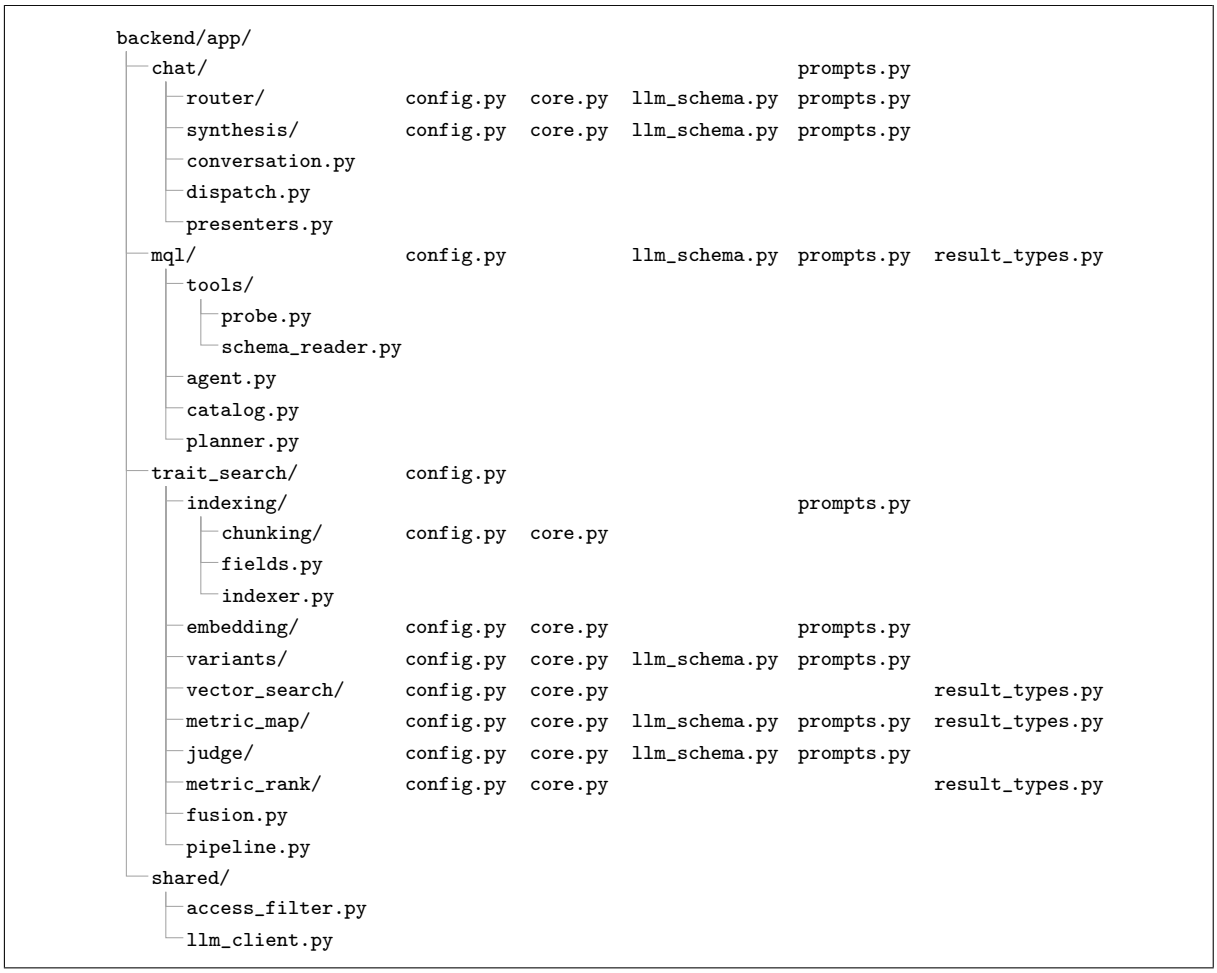


Figure 4.1: The folders and files of the backend, without the files `__init__.py`. The five files a stage can have stand in fixed columns on the row of their folder, so an empty place marks a file the folder does not have. Every other file is a leaf of the tree. `variants` holds the call that divides the trait part, `metric_map` the matching of the fields and `metric_rank` the rating evidence. Left out are the folders `api` with the web routes, `db` with the access to the database, `models` with shared data types and `services` with two helpers, the file that retrieves the worked examples for one configuration of the evaluation, and the files that start the application. The copy of the platform's Mongoose schemas, which `schema_reader.py` reads, lies beside `app/` in `backend/schemas/`.

and the vector search is restricted to these reports before the nearest chunks are chosen, so a chunk of another player cannot take a place among them. The search engine accepts about a thousand values in such a filter, so a longer list of reports is searched in batches whose results are merged.

4.3.2 MQL Sub-Agent

Prompt. What the MQL sub-agent needs to know depends on the collections a question touches, so its prompt is split into two parts. What holds for every run stands in the system prompt, and what concerns one collection arrives with a schema read of that collection. The system prompt holds the four operations, the rules on how to query at all and a catalog with one line for each of the 31 collections, its name and a short description. The catalog sorts the collections into seven categories by theme, such as the scouting evaluations with the reports and the fields scouts rate, or the market with transfers and rumours. One of these rules is that stored text is matched by a pattern and never by its exact spelling, since the model does not know how a value is stored (Section 3.3.3). A schema read returns five things for the collection,

(1) its declared schema, (2) the references to and from other collections, (3) the fields that carry an index, (4) the notes on the collection and (5) the worked examples tagged on it. The code that registers the collections lists for each one the fields the model must never see, such as the passwords and email addresses of the users. These fields are removed from the declared schema and from every document the model is shown, and a filter that names one is rejected. The notes state what neither the declared schema nor a sample of documents shows, such as that the season statistics store the season 2025/26 as "2025". Since the notes come with the schema read, such a rule reaches only the runs that read that collection. A worked example is a whole run, from the first schema read to the answer, and it is tagged on every collection whose use it shows, such as the collection that links two others. An example tagged on several collections is shown once per run, however many of them the run reads. None of the worked examples answers a question of the test collection.

Code around each step. With this prompt the model writes the steps of a run, and code adds to each step what the model must not decide or cannot see. Figure 3.2 shows the steps of one run as the model wrote them. On the probe of step 2, code adds clauses that leave out deleted documents and restrict the documents to those the user may see. It then runs two queries at the same time, one that counts the matching documents and one that reads the five documents of the sample. The ids of all matching documents would need a third query, which for a broad filter returns up to 100,000 ids where the sample returns five. Code therefore stores only the filter under the handle `teams#1` and runs that query only when a later step uses the handle or the answer attaches it. In the sample, code replaces every id the database assigned with a token. On the aggregation of step 3, code places the access clauses at the start of the pipeline, appends a limit and replaces each player id in the rows with a token. A foreign key in the rows, such as a report's player id, is labelled with the collection it refers to, which code takes from the declared schema and not from the type the model wrote. The model knows from the schema where the key points, and the token states the same, so a result stays consistent with what the schema told the model about the key. On finalising in step 4, code delivers `agg#1` as a table and resolves its column of player ids to the players' names, linked to the players.

A join to another collection needs the most from code, since it is where access rights and the different forms of an id meet. The model writes a standard MongoDB join whose inner pipeline has to project the joined documents to the fields it needs, since whole documents, such as reports with their full text, would fill the rows the model reads. Code puts its own stages in front of the model's, as Figure 4.2 shows for a join from players to their reports. It converts the player's id to the form in which a report stores it, restricts the joined reports to those the user may see, leaves out deleted ones and removes the fields the model must never see.

A step carries a MongoDB filter or pipeline, a document whose fields cannot be listed in advance, while the provider holds an answer only to an output schema that lists every field, and only to the whole of it. The output schema of a step therefore reaches the model only as guidance, unlike those of the other stages, and code validates the step after the call. Code also checks a rule that no output schema can express, since the arguments a step needs depend on the operation it names, such as that an aggregation needs a pipeline and a final answer takes no filter. If a step does not fit its output schema or the arguments its operation needs, it goes back to the model with the error, and the model writes it again, at most two more times. If it still fails, the error becomes the observation of that step, so an invalid step never reaches the database. What a step that passes asks of the database is checked before it runs (Section 3.3.2), and a step that breaks one of those rules comes back as an error observation, which the next step repairs.

Session memory. What a run leaves for the next run of the same session is kept in the session memory. For each earlier run it holds the task, each query step as the model wrote it with the error of a step that failed, the answer and the handles with their sizes. The ids behind the

As the model writes it

```
{"$lookup": {"from": "reports", "localField": "_id", "foreignField": "playerId",
             "as": "reports", "pipeline": [{"$project": {"createdAt": 1}}]}}
```

As it reaches the database

```
{"$addFields": {"_mqlJoinKey0": {"$convert": {"input": "$_id", "to": "string",
                                              "onError": null, "onNull": null}}}},
{"$lookup": {"from": "reports", "localField": "_mqlJoinKey0", "foreignField": "playerId",
             "as": "reports",
             "pipeline": [
               {"$match": {"$and": [
                 {"deleted": {"$ne": true}},
                 {"gender": "male"},
                 {"$or": [{"userId": "u1"}, {"sharedWithUsers": {"$in": ["u1"]}}],
                  "gender": "male"}]}},
               {"$project": {"reporter": 0}},
               {"$project": {"createdAt": 1}}]}}
```

Figure 4.2: A join from players to their reports, as the model writes it and as code rewrites it, for a user who may see only the reports they wrote or that were shared with them. `_mqlJoinKey0` is a field code adds to hold the player's id as a string, the form in which a report stores it. Code's `$match` with the access clauses and its `$project` that removes the hidden fields are always stages of their own, placed in front of the model's stages, which stay unchanged. Generated by the system's rewrite (`_rewrite_lookups` in `backend/app/mql/tools/probe.py`).

handles stay in code. Of an attached set the memory keeps the ids, since they were fetched for the answer and a follow-up refers to exactly the players the user was shown. Every other handle keeps only its filter, since its ids are fetched only when a step uses it, and fetching them for the memory would cost a query for every handle of a run. A later run of the same session that reuses such a handle runs its filter again.

Settings. The settings of the sub-agent bound the length of a run and what the model sees in it. A run ends after at most 20 steps, a ceiling chosen from a few sample runs. A probe shows 5 documents as its sample, and the model may ask for up to 100 where the documents themselves are what it has to read, such as the latest reports on one player whose content it summarises. An aggregation shows up to 20 rows whole, which keeps a top-twenty ranking complete, and of a longer result its first and last 10 rows. An observation is measured in characters against a display budget of 90,000 characters, about 2.5% of the context window, and rows or sample documents are left out whole until it fits. A query runs for at most 60 seconds.

4.3.3 Trait Search

Indexing. The chunks are cut when a report is indexed. The written report is cleared of the formatting some clubs store with it, split into sentences and packed into chunks of 100 to 300 characters, which is one or two sentences. Each text field is cut in the same way, and each of its chunks begins with the label of its field and of the field's category. Every chunk is stored with its text, its vector, the id of its report and, for a chunk from a text field, the id of that field.

Prompts. A trait question passes through three stages that call a model, and the prompt of each is short and carries one decision. The call that divides the trait part returns at most five traits with two query variants each and shows worked divisions as examples. Every trait runs its own retrieval and its own judge call, so each further trait adds to the cost of a question, and the prompt bounds their number. The bound is an instruction to the model and is not enforced in code, since a sixth trait only costs more, while cutting the list would drop a trait the user asked for.

The call that matches the fields receives the labels of the club's ordinal fields and the options of its nominal fields as a list in its prompt. A club's report templates repeat fields under the same

label, so the list names each label once, and code maps a label the model returns back to every field that carries it. The model therefore returns each label exactly as the list spells it.

The judge's prompt is the anchored scale of Section 3.4.2 with each step described. It also names three things that do not decide a score, (1) how much room a chunk gives the trait, (2) whether the chunk names the trait explicitly or only implicitly and (3) what else the chunk praises. A scout can show that a player is good in the air without using the words, and such a chunk can still reach the highest step. Where a chunk lies between two steps, the judge takes the one closer to zero. The chunks reach the judge numbered as C1, C2 and so on, so its scores are matched back to the players in code.

Parallel execution. These calls and the computations around them run at the same time wherever they do not depend on each other. The call that matches the fields runs while the chunks are retrieved. The rating evidence is computed while the judge reads the text evidence, and the judge's calls for the different traits run in parallel. The rating statistics are computed in the database, by one aggregation per field that returns for each player the mean of his grades, their number and their standard deviation, or for an option the number of his reports that tick it. Every step after these statistics, from the rating evidence of Section 3.4.3 to the final ranking of Section 3.4.4, needs no model call.

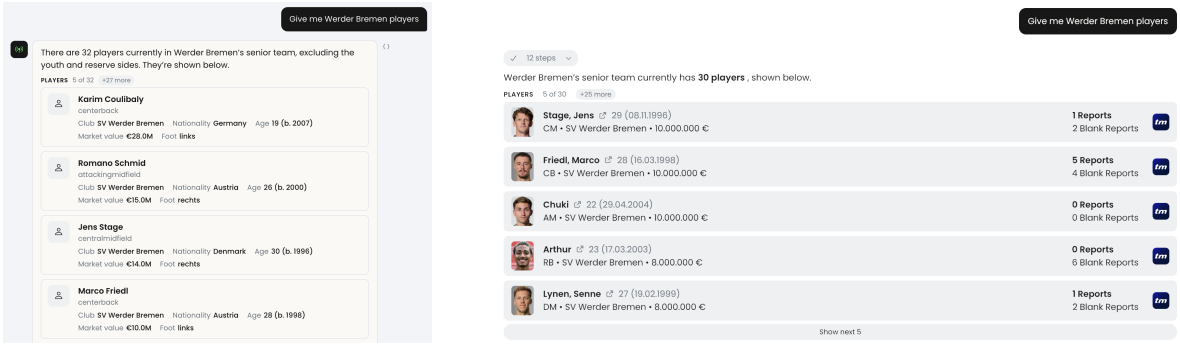
Settings. Of the settings the trait search runs with, those of retrieval are set for recall. For each query variant the vector search explores 10,000 candidates, the most it accepts. Rank fusion uses the constant $k = 60$, which Cormack et al. (2009) set in a pilot study and kept throughout. How far down a ranked list to read is a trade-off between signal and noise, and it arises twice. The judge receives the 5 best chunks (m) of each of the first 50 players (N) of a trait, at most 250 chunks, and the 50 is a choice between the cost of its call and the players it covers. A player beyond the first 50 counts as having no evidence on that trait at all, which costs most on a common trait such as pace, whose deserving players outnumber 50 while those of a rare trait do not. Retrieval can read further at little cost, since the judge filters what it lets through. Each variant returns twice as many chunks as the judge reads, $2 \cdot N \cdot m = 500$, so that a player whose reports describe the trait in several chunks can reach the first 50 above a player with a single close chunk, even when each of those chunks ranks only in the middle of the lists that the trait's two query variants return. The depth thus grows with the judge's budget. Where a ranked list stops carrying signal is a known problem of retrieval, ranked-list truncation, which this thesis does not take up (Chapter 7).

On the rating fields a single grade suffices to rank a player, since the floor of a grade's spread, $\Delta/\sqrt{12}$ (Widrow et al., 1996), gives even one grade a margin. The cautious end lies at $z = 1.96$, the conventional end of a 95% interval. A combined standing, a player's weighted mean percentile on the fields of a trait, of 0.60 or more is evidence for a trait, and one below 0.30 on the ordinal fields is evidence against it. Mapped onto the same range from 0 to 1, the judge's scale puts its lines of $+1$ and -2 (Section 3.4.2) at the same two points, so evidence against needs the stronger signal on both kinds of evidence. A trait without evidence counts as $\varepsilon = 0.02$, a small value above zero, since zero would set the geometric mean of every player who misses a trait to zero. A player at place r of the n players a view admits gets the rank score $\varepsilon + (1-\varepsilon)(n-r+1)/n$, from 1 for the best down to just above ε for the last, so the last admitted player still counts more than a trait without evidence. A player has to meet more than half of the traits, which reads a question as a profile that a player may fit in most respects (Section 3.4.4). Requiring every trait would be as defensible and would return fewer players, and which of the two fits is a preference of the user. Each excerpt and rating on a player's card names the trait it speaks for, so the wider result shows the user which players fit only part of a profile.

4.3.4 Answer Synthesis

Prompt. In the prompt of the synthesis call a note on what the user was already told comes first, then the question, then the results as material that carries no instructions. The question stands before the results because, with the question last, the model answered the results, and a database answer can end with a question to the user (in line with R1 in Table 2.4). The rules on language, dates and how the user is addressed are one block of text that the router's prompt and the synthesis prompt both end with, so the two calls that write to the user speak the same way.

Results and cards. What the synthesis call receives depends on the route, and it is a short form of what the interface shows. On the fact route it receives the sub-agent's answer, the first rows of a table the sub-agent attached and the names and the number of the players or records shown. On the trait route it receives the number of matched players, the names on their cards and whether the cards carry excerpts from reports, averages of ratings or both. On the hybrid route it receives both. The cards themselves are built in code from the ids the parts returned. The prototype's cards follow the components the scouting platform's interface uses to show a player, a club or a transfer, and the agent inside the platform uses those components themselves (Section 4.1). Figure 4.3 shows one question in both.



(a) The prototype **(b)** The agent inside the platform

Figure 4.3: The question "Give me Werder Bremen players" (a) in the prototype, with cards of its own, and (b) in the agent inside the scouting platform, with the platform's cards. The prototype runs on the frozen copy of a club's database on which the thesis is evaluated, the platform on current data, so the two answers count different numbers of players.

5 Evaluation

This chapter evaluates three components of the system, the MQL sub-agent, the trait search and the router, and tests the design decisions of Chapter 3 that concern each of them. Section 5.1 describes what the three evaluations share, which are the questions with their gold, the frozen database and the runs. Sections 5.2 to 5.4 then take one component each, with how its output is scored, which hypotheses it is tested on, and what its results and its cost on both models are. The router comes last, because its test uses the scores that Sections 5.2 and 5.3 report.

5.1 Evaluation Setup

5.1.1 Question Set

The evaluation uses 154 questions, of which 65 ask only for traits, 43 only for facts and 46 for both. The router is scored on all 154, the MQL sub-agent on the 43 fact questions and the trait search on the 65 trait questions. Each question carries the gold for every component it reaches. For the router the gold is the route and the division of the question into a fact part and a trait part (Section 5.4). For the MQL sub-agent it is a gold query written by hand, whose result on the database is the list of players the sub-agent has to return (Section 5.2). For the trait search it is the traits the question asks for, from which Section 5.3 derives the relevant players. Figure 5.1 shows how each of the three golds is built. Appendix F lists all 154 questions with the IDs by which the text names them.

The 46 hybrid questions are scored only on the router's route and division, and their answers are not scored, which is the largest gap of the evaluation. On a hybrid question the trait search ranks only the players that the fact part returns (Section 3.2.3), and the test collection holds reports on only 70 players. A fact condition therefore leaves a handful of them, between 6 and 18 in six hybrid questions chosen to span the widest range, and a trait search over six players adds little to reading their reports. The trait search is built to rank many players, and on a club's full database, where far more players have reports, a fact condition leaves enough of them for a ranking to matter. Scoring the answers of the hybrid questions would therefore measure a case that the test collection creates and a club's database does not.

Scoring the components on their own still covers what decides the answer to a hybrid question. The router's division is scored on the 46 hybrid questions, the MQL sub-agent on the 43 fact questions and the trait search on the 65 trait questions. The dispatch between them adds no decision of its own, since it runs the fact part first and passes the ids of its players to the trait search in code. If the router divides a question correctly and both components answer their parts correctly, the answer to the hybrid question is therefore correct in most cases as well. Two things can still make it wrong, and neither is measured. (1) The ratings are read by a player's standing among the players the fact part returned (Section 3.4.3), so in a small group a standing rests on few players. (2) An error of the fact part carries into the trait part, since a player the fact part wrongly leaves out cannot be ranked at all. Chapter 6 discusses what this leaves open.

Every question also exists in German, so that the MQL sub-agent and the trait search are tested on questions in both languages. The German phrasings are translated by meaning rather than word for word, in the expressions German scouts use. Each German phrasing asks for the same facts and traits as its English one, so the two share one gold and count as one pair of phrasings rather than as two questions. What a German question costs each of them is measured in Sections 5.2 and 5.3.

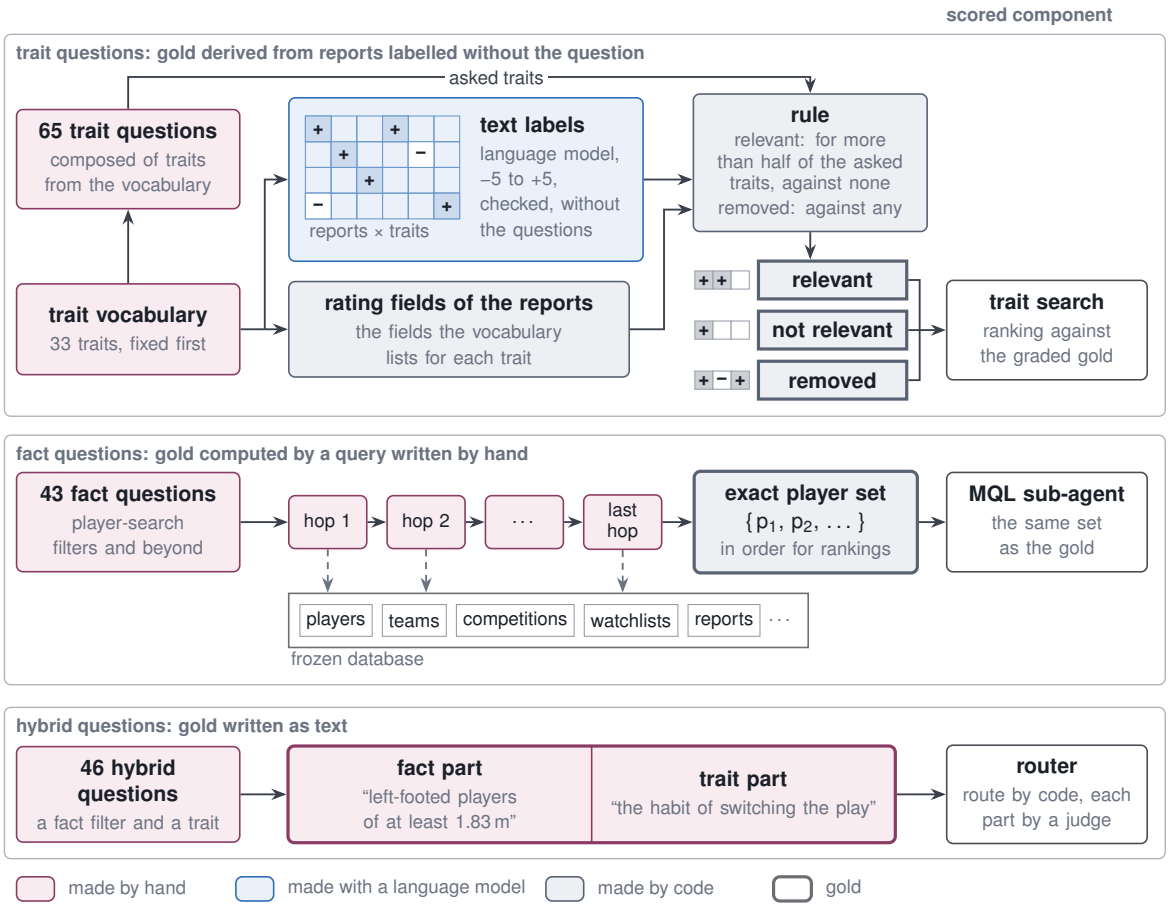


Figure 5.1: How the gold of each kind of question is built. For a trait question it is derived from reports labelled without the question, for a fact question it is computed by a query written by hand, and for a hybrid question it is the question's division, written as text. Each gold scores one component. Every question is also asked in German with the same gold, and every query reads the one frozen database.

5.1.2 Frozen Database and Runs

All runs read the frozen copy of a club's database that the prototype of Section 4.1 runs on. Each question also fixes the date the system takes as today, so a question for players "under 23" keeps its gold however long after the test collection was built it is asked. The data of the copy has three origins. (1) The players, clubs and competitions with their attributes are real and unchanged, so the names and stored forms of values that the MQL sub-agent has to find are those of production. (2) The text of the reports is real prose by scouts (Section 5.3). (3) The rating fields of the reports with their values and the rest of the club's own data, such as watchlists and scouting statuses, are made for the evaluation. No single club's data holds every case the evaluation has to test, such as a rating field that records weaknesses beside one that records strengths. The data of several clubs together would hold more of them, but the evaluation runs on one database, so this data is made to hold the cases one club's data lacks. They are made with deliberate variety, so the database holds an empty watchlist beside a large one, lists that overlap and players on none, and questions that combine or exclude lists are not trivial.

All runs ran on one laptop, an Apple MacBook Pro with an M3 Max processor of 14 cores and 36 GB of RAM. The frozen copy ran there in MongoDB 8.3 with its search engine, in the container image that MongoDB provides for local use (Atlas Local). The container used 8 processor cores and about 10 GB of RAM. The backend ran in Python 3.11 outside a container. The embedder ran on the processor, and only the calls to the language models left the laptop, to the

deployments on Azure of Section 4.2, so the times in the results include the network and the load of the provider.

The MQL sub-agent and the trait search are each run in two modes. In the isolated mode the component receives its part of the question from the gold, and the router does not run. In the end-to-end mode the question enters at the router, and the component receives the part the router extracted from the question. Any drop from the isolated score therefore comes from the router, and the share of its isolated score that a component keeps end to end is what the router's mistakes cost it (Section 5.4). Without the isolated mode a low score could not be told apart from an error of the router. The router itself is scored directly, by comparing its route and its division of each question with the gold, with the clarification route switched off (Section 3.2.2).

The larger GPT-6 Sol and the smaller GPT-6 Luna of Section 4.2, called Sol and Luna below, each answer every question in every configuration. For the hypotheses the two models replicate each other, and a hypothesis counts as supported only if its test rejects the null hypothesis on both models (Section 5.1.3). The findings of Chapter 2 were measured on models of several generations and carry over as directions, not as sizes, so the hypotheses predict at most a direction, never a size. The two models are also compared with each other, though not by a test, since the larger model is expected to answer better. The comparison shows for each component how much the larger model gains against how much more it costs, since Luna costs a twentieth of Sol per token (Artificial Analysis, 2026). The results of each component set its score against its cost on both models, from the tokens every run records. Cost is priced at OpenAI's list price for the two models, which serves as the reference price (OpenAI, 2026b). At this price a token read from the cache costs a tenth of an input token and a token written to the cache a quarter more. In use, the provider would read much of a prompt from its cache, so the cost assumes the share of cached tokens that a single question would get. In the runs, the provider served from its cache 92 and 87 % of the prompt tokens of the MQL sub-agent on Sol and Luna, 31 and 21 % of those of the trait search and 99 % of those of the router. The cost assumes about the same shares for the MQL sub-agent and the router but only about 6 % for the trait search. The assumed shares are lower than those of the runs because each question ran five times, so the provider had already cached the parts of a prompt that belong to the question itself, which a single question in use would not find, most of all in the judge calls of the trait search. Since the provider caches every prompt of at least 1,024 tokens, the uncached part of such a prompt is priced as written to the cache (Microsoft, 2026c).

5.1.3 Hypotheses and Tests

Each component is scored on the measures of Table 5.1 in every configuration it runs in, and the evaluation tests the eight hypotheses of Table 5.2.

Table 5.1: The measures each component is scored on, in every configuration it runs in.

Component	What is scored	Measures	Questions
Router (3.2.2)	How often the router gives the right route and divides the question as the gold does	route accuracy, parts the same	all 154
MQL sub-agent (3.3.2)	How often the sub-agent answers correctly, and in how many steps	success rate, steps	43 fact
Trait search (3.4)	Whether the trait search finds all the relevant players, how well it ranks them, and whether it keeps out the players whose reports speak against a trait	recall, nDCG@10, harmful inclusion, discounted harm@10	65 trait
Every component	What a question costs on each model	tokens, price, time	all 154

Table 5.2: The hypotheses of the evaluation, each on a design decision of Chapter 3 or on a condition the system has to handle.

Tested	Hypothesis	Compared	Tested on	Questions	RQ
The sub-agent receives worked examples (3.3.3)	Without the worked examples the sub-agent answers fewer fact questions correctly	*w/o examples* and *Standard*	success rate	41 fact	RQ2
Examples are tied to a collection (3.3.3)	One way of giving the examples performs better than the other	*Retrieved examples* and *Standard*	success rate	41 fact	RQ2
Schemas are read when a step needs them (3.3.3), R5 against R8 of Table 2.4	Reading schemas on demand and loading every schema up front differ in how many fact questions they answer correctly	*Preloaded schemas* and *Standard*	success rate	41 fact	RQ3
Questions in German (3.3.3)	Fact questions in German are answered less accurately than the same questions in English	*German* and *Standard*	success rate	41 fact	RQ2
A judge reads the sign of each chunk (3.4.2)	Without the judge, players whose reports speak against a trait stand higher in the result	*w/o judge* and *Standard*	discounted harm@10	53 trait, with a player whom the text of his reports removes	RQ1
Two query variants per trait (3.4.2)	Two variants fused by rank order the players better than the first variant alone	*One variant* and *Standard*	nDCG@10	59 trait	RQ2
Questions in German (3.4.2)	Trait questions in German rank the players less well than the same questions in English	*German* and *Standard*	nDCG@10	59 trait	RQ2
A router divides the question first (3.2.3)	Behind the router each component scores lower than on its own	*Via router* and *Standard*	success rate, nDCG@10	41 fact, 59 trait	RQ2

Standard is a component as designed, and each other name in italics changes one of its settings and names that configuration in the tables and figures.

Every question is answered five times by each configuration on each model. A model can answer the same question differently and at a different cost in each run, so a single run would leave both to chance. Kapoor et al. (2025) ran each agent they compared five times and reported its mean accuracy and mean cost with intervals over the runs, since a result reported from one run leaves its variance unknown. Each question enters a test with its mean over the five runs. For a graded measure such as nDCG this is the mean score, and for a question that is answered either correctly or not it is the share of the runs that answered it correctly.

Each hypothesis is tested on the questions of its component and on one measure, the one Table 5.2 names. For question i let d_i be the difference between the mean scores of the two configurations, taken in the direction the hypothesis predicts, so that a positive d_i agrees with it. The null hypothesis is that the mean difference μ_d over the questions is at most zero, and the alternative is that it is positive. Both configurations answer the same questions, so the null hypothesis is tested with the one-sided paired t-test at a significance level of 0.05. Its statistic is $t = \bar{d}/(s_d/\sqrt{n})$ with $n - 1$ degrees of freedom for n questions. Each mean difference is reported with the lower bound of its one-sided 95 % confidence interval, which lies above zero exactly when the test rejects. Two hypotheses predict no direction, the choice of the worked examples and the reading of schemas. Both are tested two-sided, with the null hypothesis that the mean difference is zero, and each difference is reported with a two-sided 95 % interval, whose sign

shows which way works better. In the text, an interval stands in brackets after its value, and every interval is at the 95 % level. Three hypotheses predict that a design decision costs some of the score, the German questions of both components and the router. For them the bound says how much the decision costs at least or, when the test does not reject, how much it could cost at most.

The paired t-test makes two assumptions. (1) The differences d_i are independent, so that the difference on one question says nothing about the difference on another beyond what the configurations change for all questions. A weaker configuration lowers every score, which moves the mean and is what the test measures. Questions that ask for the same trait can still depend on each other, since a weakness of a configuration on that trait shows in each of them, and the test then counts them as more evidence than they are (Chapter 6). (2) The mean difference $\bar{d}$ is approximately normally distributed. The scores themselves cannot be exactly normal, since a normal distribution has tails without bound and the scores lie between 0 and 1. The test does not need them to be, because by the central limit theorem the mean of many independent values is close to normal whatever their distribution. How many questions are enough was tested by Urbano et al. (2019) for nDCG, the measure of the trait search, on TREC data. From 25 topics up, the t-test found a difference between two equally good systems in no more than 5 % of the cases, as a significance level of 0.05 requires.

On the fact questions the t-test alone can give too small a p-value. A fact question whose share of correct runs is the same in both configurations has a difference of zero, so only the questions that changed move the mean difference. When only a few of the 41 questions change, the mean difference can take only a few values, with four changed questions for instance only five, and is far from normal. The whole chance of the observed value then belongs to the p-value, but the normal curve of the t-test spreads it over a slope on both sides of the value and counts only the part beyond it, so for four changed questions that all improved it gives 0.022 instead of 0.0625. The fact questions are therefore also tested with a sign-flip permutation test (Smucker et al., 2007), which assumes no normal distribution. If a configuration made no difference, each changed question would be as likely to have gained as to have lost, so the test gives each of the k changed questions either sign, in all 2^k combinations. Its p-value is the share of the combinations whose mean difference is at least as large as the observed one, or for a hypothesis without a direction at least as far from zero. Even when all k questions gained, only one combination reaches the observed mean, or two without a direction, so the test can reject at 0.05 only from five changed questions on, since $1/2^5 \approx 0.031$, and without a direction from six. A hypothesis on the fact questions counts as supported only when both tests reject, so that its p-value does not rest on a normal distribution, and the t-test gives the bound on how large the difference is at least.

A rejected null hypothesis supports the claim that the configuration the hypothesis favours scores better on average, by at least the lower bound of its confidence interval. The claim holds for questions like these, on this database and with these models, and not for each single question or for the data of another club. When a null hypothesis is not rejected, the upper bound of the interval is reported as well, since it shows how large an effect could at most be. Each hypothesis is tested on both models, since it states a property of the design and not of one model. It counts as supported only when the test rejects on both, as in the intersection-union test of Berger (1982). This keeps the probability that a false hypothesis counts as supported at 0.05 at most. Counting a rejection on either model would raise it, since on each model the test can reject by chance with a probability of up to the significance level of 0.05. A comparison in the results that no hypothesis names comes from an analysis chosen after the runs and is therefore exploratory, since it can suggest an effect for a later test but supports none.

Some decisions of Chapter 3 are argued there and not tested. (1) Whether the router needs its worked examples is not tested, since they cost few tokens and would stay whatever the result. (2) A single agent with every tool is not tested, since it would be a second system to

build and evaluate, which lies outside the scope of this thesis. (3) The MQL sub-agent is not compared with a fixed workflow that takes the same steps in a fixed order, such as reading the schemas, probing the stored values and writing one query that joins what the question needs. Its stages would need prompts and examples of their own, since the worked examples show runs of the loop, so it would be a second system as well. (4) Whether the judge should read more than 50 players per trait is not tested, because the test collection holds only 70 players and the judge already reads most of them. The question matters on a club's full database instead, where far more players have reports that speak of a trait than the judge reads. The judge reads only the first 50 players in the order of retrieval. A player further down gets no text evidence, but he can still reach the result through his ratings or the other traits of the question. A relevant player can therefore be lost although his reports speak for the trait, and a player whose reports speak against it can stay in the result, since the judge never reads that report. How the budget could adapt to each trait lies outside the scope of this thesis and is taken up in Chapter 7. (5) Within the trait search, the rule that weighs text against ratings and joins the traits of a question is not compared with other rules. The relevant players of a question are derived from the text labels by this same rule (Section 5.3), so a comparison would favour it by construction. Testing the rule would need relevance assessed for each question as a whole, for instance by scouts. (6) The conversion of ids and the access rights are guaranteed by code and checked by its tests (Section 3.3.4). (7) That the MQL sub-agent sees handles and tokens instead of ids (Section 3.3.3) is not compared with showing it the ids themselves. It is a design decision argued there, and the ids themselves would cost far more tokens on every step that works with a set of players. (8) The synthesis call is not scored, since the results it presents are scored at the components and are shown beside the reply (Section 3.5). What it adds is mainly how the reply reads, which a study with users would have to judge.

5.2 MQL Sub-Agent

5.2.1 Fact Questions

The fact questions start from the player search of the product, since the sub-agent has to answer at least what that search can already filter. The search was built together with clubs, so its filters are the facts by which scouts look for players. Of the 41 questions, 11 can be answered by the search itself, with its sort and its filters on basic data, debuts, performance data, report ratings and the club's tags, watchlists and scouting statuses. Since the sub-agent writes its own queries, it can answer more than these filters. The other 30 questions therefore go beyond the search, in what they combine, such as the players on one list but not on another, or in what they ask about, such as the injuries of a player. The number of questions follows from this coverage rather than from a target, and the 41 questions span 17 kinds of attribute, such as the age, the market value or the club. The questions are written to leave nothing open that would call for asking back, since the system answers an open question by asking the user (Section 3.2.2), which the evaluation does not score.

Two problems recur in the literature on natural language database querying, and 35 of the 41 questions put the sub-agent in front of at least one of them. The first is finding how a value is stored. A question names a value in its own words, and the database may store it in another form (Li et al., 2023). A question for "Brazilians", for instance, has to find the players whose nationality is stored as `Brazil`. The sub-agent meets this problem by probing the stored values before it answers (Section 3.3.3), and 27 questions name such a value. The second is the nested data model, in which a document holds lists of further documents that a relational database would keep in separate tables. In SM3, MQL was most accurate on questions about top-level documents rather than embedded ones, and on Spider 2.0 an agent solved fewer tasks over databases with nested columns than over the others (Sivasubramaniam et al., 2024; Lei et al., 2025). Of the 41 questions, 24 need data from such a list of embedded documents, such as the clubs a player belongs to. The data adds a problem of its own, since it stores many facts

more than once, such as the matches of a player, which stand both in his own performance history and in the records of the matches, and the records of the matches miss one more often than his history does. The sub-agent then has to know which copy to trust, which the notes of a collection state and its worked examples show by the copy they use (Section 3.3.3).

Each gold query follows the path the sub-agent is built to take, not the shortest path that an author who knows the database could write. It is a sequence of hops, each a query, and it starts from what the question says. A named club, competition or list is therefore first found in its own collection, and a later hop filters the players by it. A shorter path would have to guess an id or the stored form of a name, which the instructions of the sub-agent rule out, since they tell it to look up stored values before it answers (Section 3.3.3). The 41 questions need between 1 and 4 hops, and each gold query returns between 5 and 36 players.

5.2.2 Measures and Tags

A run answers a fact question correctly when the players the sub-agent attaches to its answer are exactly the players of the gold, compared as a set. Only the result is compared and never the queries the sub-agent wrote, just as execution accuracy compares the result of a predicted query with that of the gold query (Li et al., 2023). Spider 2.0 likewise scores only the output of an agent's whole run and calls the share of solved tasks the success rate (Lei et al., 2025). In the five questions that rank the players over the whole result the order counts as well, and players with the same value may stand in either order. The success rate is averaged over the 41 questions whose gold query returns players, which form the main population. Two further questions ask for facts the database does not hold, such as players with a pilot's licence, and are reported one by one.

Beside the success rate, each run is scored on how many steps it takes beyond the shortest path its gold query implies. The shortest path takes at least 3 steps, (1) one to read the schemas, (2) one for each hop and (3) one to finalise (Section 3.3.2). A longer run repairs an error, repeats a step or reads a collection the question does not need.

The steps of the correct runs also show whether a step limit below 20 would cost accuracy. On the tasks it solved, the agent of Spider 2.0 took 9.0 steps on average and between 6 and 17 (Lei et al., 2025). A run that takes far more steps than the correct ones has most likely gone in a wrong direction, and it costs the scout time and the product tokens, while a follow-up question from the scout could set it on the right path. The recommended limit is therefore the 95th percentile of the steps of the correct runs, which at most one correct run in twenty would exceed. The longest correct run would be a poor limit, since one unusual run would then set the limit for all.

A wrong run is assigned one of seven outcomes, so the results show how the sub-agent fails and not only how often. (1) No answer means the run reached the step limit, whose last call answers in words but attaches no player, and (2) an empty answer means the sub-agent finalised by itself without a player. (3) Too few means it attached only right players but not all of them, which points to a filter that is too strict. (4) Too many means it attached all right players and others besides, which points to a filter that is too loose. (5) Partly right means it attached right and wrong players and missed some right ones, (6) none right means it attached only wrong players, and (7) wrong order means it attached the right players in the wrong order. TEND divides its runs into outcomes of this kind, so that the correct runs and the outcomes sum to one (Lu et al., 2026). Its outcomes for invalid syntax and for execution errors have no counterpart, because the sub-agent repairs such errors within its run.

Each wrong run is also assigned the cause of the first error that its answer inherits, so the results show where a run goes wrong and not only how its answer ends. A step that the sub-agent corrects later, such as a schema it reads and then leaves unused, counts as exploration and not as an error, since unlike a query written in one call, a run can still change course after it. The

causes follow the classes of errors that DIN-SQL found in the queries a model wrote wrongly (Pourreza and Rafiei, 2023), such as schema linking, a wrong join or a wrong aggregation. Invalid syntax is again left out, because the sub-agent repairs such errors within its run, and three causes are added. (1) A wrong stored value means the run read the right field with a value the question does not mean. (2) A failure of the run's process means that no earlier step went wrong, but the run lost a right result in its last steps or reached the step limit. (3) A stated reading means the run took a reading that the wording of the question allows and said so in its answer or asked back about it.

The success rate is also given separately for five tags, so the results show which kinds of question the sub-agent answers and which it fails, as text-to-query benchmarks report their results. From Spider the tags take what makes a query hard, the constructs it uses (Yu et al., 2018), and from Spider 2.0 the way of reporting them, one rate for each task attribute (Lei et al., 2025). (1) The reasoning-shape tag groups the questions by whether answering needs an aggregation such as a count (SQL's `GROUP BY`), a ranking such as the five most valuable players of a club (`ORDER BY` with `LIMIT`), or set logic such as the players on one list but not on another (`EXCEPT` or `UNION`). In MQL these constructs become a `$group` stage, `$sort` with `$limit`, and `$nin` or `$or` over a set of players. (2) The hop tag groups them by the number of hops of their gold query. (3) The stored-value tag marks the questions that name a value only the documents hold, and (4) the array tag those that need a list of embedded documents. (5) The search tag marks the 11 questions the player search of the product can answer itself, so its rate shows whether the sub-agent reaches what the product already does. Table D.7 defines each tag and gives the number of questions that carry it.

5.2.3 Hypotheses

The first hypothesis tests the worked examples. The sub-agent receives its 36 worked examples with the schema of each collection whose use they show (Section 3.3.3), so it can receive an example for a different question than the one it answers. In SM3, examples in the prompt raised the accuracy of MQL more than that of SQL for the larger models, and examples chosen for each question worked better than fixed ones (Sivasubramaniam et al., 2024; Gao et al., 2024). The hypothesis expects that without the examples the sub-agent answers fewer fact questions correctly, and the test on the success rate over the 41 questions of the main population shows how much the examples add. The steps and the tokens of each run are reported beside it, since every schema read carries the examples of its collection and a run without them may take more steps.

The second hypothesis tests whether examples tied to a collection and examples retrieved for each question differ, without assuming a direction, since both have a reason to work better. Examples chosen for each question show a query close to the one the question needs, and they worked better than fixed ones where the literature compared them (Sivasubramaniam et al., 2024; Gao et al., 2024). Examples tied to a collection show how that collection is used and how it links to others, which matters where the difficulty lies in the documents, as TEND's authors read their results (Lu et al., 2026). SM3's authors relate the gap between MQL and SQL to how little MQL the models had seen in training (Sivasubramaniam et al., 2024), which examples of either kind address. The retrieving configuration uses the embedder of the trait search to find the 5 examples closest to the question and places them before the task, as SM3 retrieved 5 examples for each question (Sivasubramaniam et al., 2024), and a schema read then returns the schema without its examples. The two sets differ in what they are chosen by. With examples tied to a collection, a schema read brings every example of that collection, whatever the question asks, while retrieval looks at what the question asks and can draw on any collection. A question that counts the players of each club can therefore be closer to an example that counts the reports of each scout than to an example that lists the players born after 2000, although only the latter reads the same collection. The retrieving configuration thus changes which examples the sub-agent receives and when it receives them, since examples chosen by the question are

known before any schema is read, and giving examples with a schema read makes sense only when they belong to its collection. The comparison therefore decides between two designs and does not show what the choice of the examples alone is worth. The hypothesis is tested two-sided on the success rate over the 41 questions, and the sign of the difference shows which way of giving the examples works better.

The third hypothesis tests reading the schemas when a step needs them, against loading every schema up front. The sub-agent reads a collection's schema on demand, which keeps its context short and the product cheap, and Section 3.3.3 argues that choice against the finding that capable models do better with the whole schema than with a schema a linker has filtered (Maamari et al., 2024). In the preloaded configuration every collection's schema stands in the system prompt from the start with its notes and examples, and the schema read is withheld. Since the short context and the whole schema each argue for one configuration, the hypothesis predicts no direction and is tested two-sided. The interval then shows whether the cheaper design gives up accuracy or gains it, and how much (RQ3). Tokens, steps and time are reported beside it, because the two configurations trade them. Reading on demand costs a step for each read and with it time, and preloading costs tokens in every call. The tokens of the preloaded schemas probably overstate their cost a little, since the provider caches a system prompt that stays the same for every question.

The fourth hypothesis tests German fact questions against field names that stay English. The field names and almost all stored values of the database are English, so matching a German mention to a field or a value is cross-lingual. MultiTEND measured a loss for questions in other languages than English, with the field names translated along with the questions (Qin et al., 2025). A question that words a field or a value differently from how the database stores it is answered less accurately even in English (Chang et al., 2023), and a German question widens that difference, so a loss is expected here all the more. The hypothesis expects German fact questions to be answered less accurately than English ones, and it is tested on the success rate over the 41 pairs of phrasings.

5.2.4 Results

How well the MQL sub-agent answers. The MQL sub-agent answers most fact questions correctly on both models, and it answers most of them alike in all five runs. In the standard configuration, which reads the schemas on demand, gives the worked examples with their collections and asks the questions in English, its success rate is 89.8 % on Sol and 82.0 % on Luna (Table 5.3). Most questions come out the same in every run, since only 5 questions on Sol and 7 on Luna are answered correctly in some runs and wrongly in others. The two questions that no player in the database matches are answered correctly, with no player, in every run of every configuration on both models.

The configurations change the cost of a run in the way their settings imply, but hardly its success rate (Table 5.3). Preloading the schemas saves the schema reads and costs tokens, so a run takes about one step fewer while its fullest prompt grows from 21 to 54 and 55 thousand tokens. Without the worked examples the prompt is shorter and a run takes about half a step more, and the retrieved examples, 5 instead of the median of 10 that come with the schema reads, lower the input tokens. The success rate, by contrast, spans less than 5 percentage points between the configurations on each model. The two models differ more, since Sol answers 7.8 [1.1, 14.6] percentage points more questions correctly than Luna. A question costs 0.10 dollars on Sol and 0.0044 on Luna, 24 times as much. This exceeds the price ratio of 20 because Sol wrote 45 % of its steps again, against 6 % on Luna, since a progress message it sent before its answer made the answer not parsable.

The hypotheses. None of the four hypotheses on the MQL sub-agent is supported, and none holds even on one model, since neither the t-test nor the sign-flip test rejects on either model,

Table 5.3: The MQL sub-agent on the 41 fact questions in each configuration on GPT-6 Sol and GPT-6 Luna.

Configuration	Model	Success rate (%)	Steps	Tokens (k) Input	Tokens (k) Output	Fullest prompt (k)	Time (s)
Standard	Sol	89.8 ± 8.5	6.6	191.4	2.9	21.1	33.2
	Luna	82.0 ± 10.8	6.1	116.6	3.0	21.1	24.5
w/o examples	Sol	88.3 ± 9.1	7.1	190.1	3.2	**18.9**	34.8
	Luna	**83.9** ± 11.0	6.6	113.4	4.2	**19.0**	28.6
Retrieved examples	Sol	88.8 ± 9.3	6.4	**178.2**	3.1	19.3	35.9
	Luna	79.0 ± 11.3	6.0	**108.2**	**2.9**	19.4	**22.1**
Preloaded schemas	Sol	88.8 ± 9.4	**5.4**	341.5	**2.3**	54.1	**26.6**
	Luna	82.9 ± 10.2	**4.9**	279.1	3.6	55.2	23.1
German	Sol	**91.2** ± 7.1	6.5	186.3	2.9	21.3	34.1
	Luna	**83.9** ± 10.7	5.8	111.6	3.1	20.4	26.0

The success rate is given with the half-width of its 95 % confidence interval over the questions. Input tokens are the total over all calls of a run, including the part served from the provider's cache. Output tokens include the model's reasoning. The fullest prompt is the largest prompt of a run, and the time is the median over the runs. Within each model, bold marks the best value in a column, although no difference of the success rate is significant.

with p-values of 0.28 and above (Figure 5.2). In each comparison only 5 to 9 of the 41 questions changed their share of correct runs, and some of them gained while others lost, so that no mean difference reaches 3 percentage points. Taking the larger bound of the two models, the worked examples raise the success rate by at most 5.6 percentage points and German questions lower it by at most 3.5. The retrieved examples lower it by at most 10.3 points and raise it by at most 4.5, and the preloaded schemas lower it by at most 5.9 and raise it by at most 6.1. Appendix I gives the p-values and bounds of every test. German questions were even answered correctly slightly more often than English ones on both models, against the prediction, and Appendix H discusses possible causes in the wording of the two languages. Section 6.1 discusses why the worked examples and the schemas changed so few questions.

Where the MQL sub-agent fails. Across all configurations and the end-to-end runs, most wrong runs still return some of the right players. They miss others or add players the question does not ask for. Their most frequent cause is an answer taken from the wrong collection or field, which DIN-SQL calls schema linking, in 67 of the 125 wrong runs on Sol and 94 of the 222 on Luna (Table D.11 in Appendix D). On Luna the next most frequent cause is a wrong condition. A wrong join, by contrast, occurs in 2 runs and a wrong aggregation in 14, all on Luna. The runs whose process failed after correct steps mostly lost a ranking. In 4 runs on Sol the sub-agent had computed the right order of the players and then fetched them sorted by market value, as the general rule of its prompt prescribes for a list (Appendix B), so a stored order replaced the computed one.

The wrong runs gather on a few questions, 14 on Sol and 21 on Luna, whereas every other question is answered correctly in all of its 30 runs (Figure 5.3). Only one question is wrong in every run on both models, the one for players who scored in more than half of the games they played (M31), since its runs mostly answered a neighbouring question that the stored season totals answer directly, goals per game. The question on playing time after a transfer separates the models most (M40), with 25 of its 30 runs correct on Sol and none on Luna, which in most runs filled the end of the summer window, which the question leaves open, with an assumption of its own that cut it too early. The success rate per tag therefore depends on which tags the failing questions carry (Table D.10 in Appendix D). The success rate of the 12 questions that

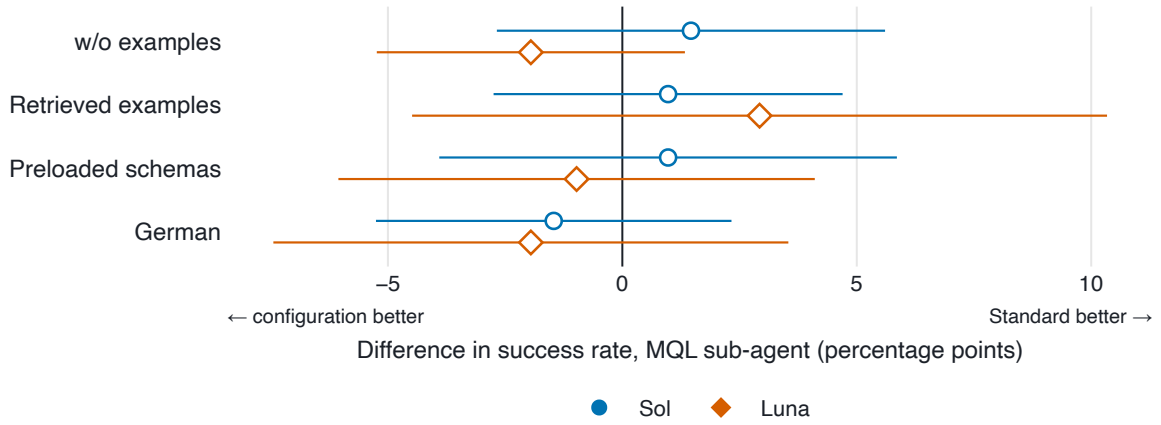


Figure 5.2: None of the four hypotheses on the MQL sub-agent is supported, since no test rejects on GPT-6 Sol or on GPT-6 Luna. Each row gives how many percentage points more of the runs *Standard* answers correctly than the configuration the row names, as the mean over the 41 fact questions. For a hypothesis with a direction the bar spans the lower and the upper one-sided 95 % confidence bound, and for the two without a direction the two-sided 95 % confidence interval. Every marker is hollow, since no test rejects.

need an aggregation lies below the average on both models, whereas that of the 11 questions the product's own player search can answer lies above it.

A correct run of the standard configuration takes a median of 6 steps on Sol and 5 on Luna, and the 95th percentile of the steps of the correct runs lies at 12 and 10 steps. Over all its runs, correct or not, the sub-agent takes about 2 steps more than the shortest path its gold query implies. A limit at the 95th percentile, as the rule of Section 5.2.2 recommends, would have cost 9 correct runs on Sol and 6 on Luna. Above that percentile, correct runs even outnumber wrong ones on Sol, 9 against 1, since the long correct runs were mostly searching the stored values, such as the name of a scouting status. The assumption that a long run has gone in a wrong direction therefore does not hold, and the runs do not fix a limit either, since across all configurations only 9 runs reached the 20 steps, all on the question on playing time after a transfer (M40). The 7 of them on Luna had gone wrong before, whereas the 2 on Sol had made no wrong step yet, so a higher limit could have saved at most these two.

Further looks at the runs. Preloading every schema trades the schema reads for tokens, since every call then carries all schemas, and a run processes about 1.8 times as many tokens on Sol and 2.4 times as many on Luna without answering more questions (Table 5.3). The provider served 98 % of the prompt tokens of the preloaded runs from its cache on both models, which makes the extra tokens cheaper than their number suggests. The fullest prompt of a preloaded run still filled only about 5 % of the context window of 1 million tokens, so a schema that fills the window was not tested.

Of the 5 examples retrieved for a question, only about half come from a collection whose schema a run then reads, whereas a run of the standard configuration receives a median of 10 examples with the schemas it reads. For four questions none of the 5 retrieved examples belongs to a collection that their gold query reads. On these four the unrelated examples cost no correct run, since the retrieved examples gave as many correct runs in total as no examples at all and no fewer than the examples tied to a collection, on both models.

Without the worked examples the sub-agent takes about half a step more per run on both models beyond the shortest path of its gold query, while its success rate changes by at most 2 percentage points. Most of the extra steps explore the data, and the runs of about 25 of the

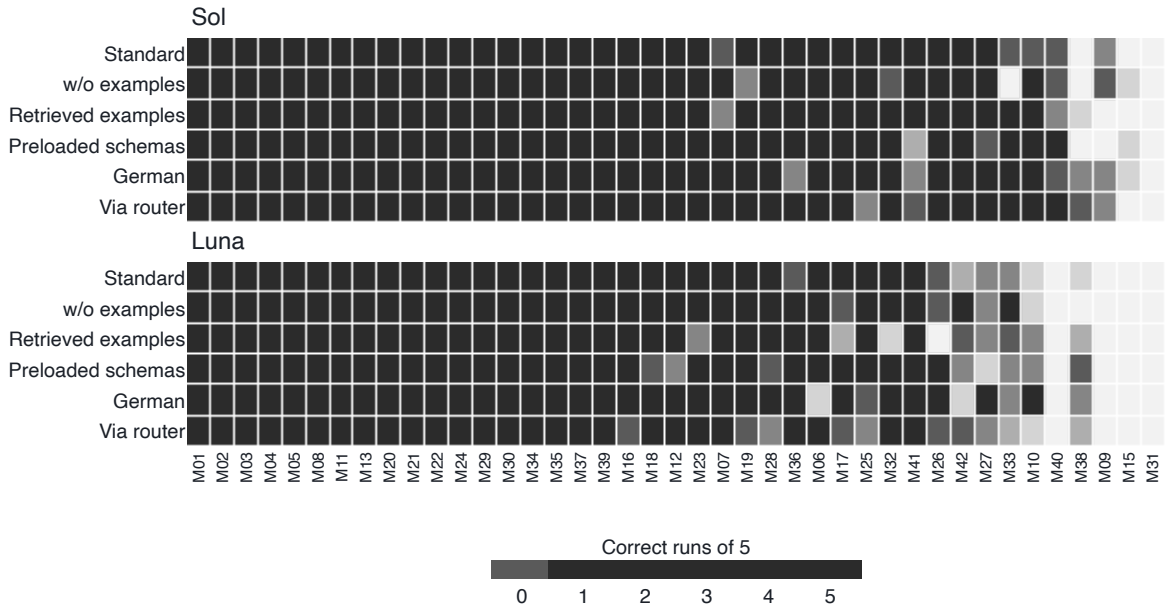


Figure 5.3: Most fact questions come out the same in every configuration and on both models, GPT-6 Sol and GPT-6 Luna. Each column is one of the 41 fact questions, sorted by its correct runs over all rows, and each cell gives the correct runs out of five of one configuration on one model, from light for none to dark for all five.

41 questions take more steps without the examples, against about 10 that take fewer. A run on Luna also takes 9.5 seconds [2.6, 16.5] longer on average, whereas the tokens of a run change by at most 2 % on both models. Without the examples, Luna also wrote more than twice as many queries that the database rejected, 82 in its 205 runs against 37 with the examples, whereas Sol wrote 6 against 3. About half of the Luna runs with such a query still ended correct, since the sub-agent repaired the query.

Comparing the runs with and without the examples shows what only the examples carry. A rule that the notes of a collection also state, such as keeping only the active leagues, was applied about as often without the examples as with them, whereas a search term that only an example used disappeared. Where the outcome changed, the examples worked in both directions. On two league questions (M19, M33) they kept Sol from doubting a league's list of clubs, which a club's own league field contradicts for a few clubs, and without them Sol took that field and answered 6 fewer runs correctly. On a question about the time under the current head coach (M42), Luna took over the one-season filter of an example's query and answered 3 fewer runs correctly with the examples than without them.

5.3 Trait Search

5.3.1 Relevance Without Pooling

The trait search returns a ranking, and rankings are usually measured on a test collection whose relevance judgements come from a pool, the method TREC established for comparing retrieval systems (Sanderson, 2010). This evaluation, however, asks how good the ranking of the trait search is, what share of the relevant players it finds and how high it places them, and not only which of its configurations ranks better. A pooled collection answers only which system ranks better. It orders retrieval systems reliably (Voorhees, 2000; Zobel, 1998), but the relevant documents that every system in the pool missed remain unknown, and Zobel estimated that the pools of TREC 5 had found at best 50 to 70 % of them. A scout uses the trait search to find every

player who fits a profile before deciding whom to look at more closely, so the share of the fitting players it finds matters most, while the players it shows can still be checked in their reports.

A judgement also has to record the sign of the evidence, whether it speaks for or against a trait. A player whose reports speak against a trait is removed from the result of the trait search (Section 3.4.4), while a player whose reports do not mention the trait can stay, so the collection has to tell the two apart. A graded judgement in its usual form cannot, since its grades run from not relevant to highly relevant, so a player whose reports speak against a trait would count the same as one whose reports never mention it. Collections that graded some documents below not relevant changed the measure so that such a grade counts against the system instead of as zero (Gienapp et al., 2020; Clarke et al., 2021). For people, the expert search of TREC 2006 came closest. It graded a person as an expert, unknown or not an expert, but counted only experts as relevant, so a person shown not to be an expert scored the same as one without evidence (Soboroff et al., 2006).

The collection has to meet two needs, (1) to count the fitting players that the trait search misses and (2) to tell evidence against a trait apart from silence about it. It meets both by labelling every report instead of assessing players per question. Every report then has its text labels, so all fitting players are known and the missed ones can be counted. Each text label also carries the direction of what the report says, so evidence against a trait stays apart from silence about it. This design joins the collections that infer relevance from units of information (Pavlu et al., 2012) with those for expert finding that rank people by their documents (Craswell et al., 2005; Soboroff et al., 2006). Ganesan and Zhai (2012), who call expert finding a related task but a different one, had derived the relevance of hotels and cars in a similar way, by a rule over the aspect ratings of their reviews. To check the ratings against people, they had two students read the reviews of the ten entities a system returned for each of 25 questions and grade how well each matched. The two students agreed little on the grades, and the authors read in their notes two criteria, one going by the evidence for each aspect and the other marking down entities with conflicting reviews. So whoever assesses relevance per question also decides how the reviews of an entity add up. A score against such judgements would mix two things, whether a system finds and reads the evidence and whether its way of adding the evidence up suits the assessor. A stated rule adds the evidence up the same way for every question, so a score measures the first and not the second. Like Ganesan and Zhai, this collection therefore derives relevance by a stated rule, and the rule is the one the trait search itself follows (Section 5.3.4). It labels each report once for every trait of a fixed vocabulary, without any question, and derives the relevant players of every question from these text labels. So no person and no model decides relevance, and a new question composed of the same traits needs no new text labels.

The collection differs from that of Ganesan and Zhai in two ways. (1) Their gold was the ratings that the reviewers gave beside their texts, while their search read only the texts. In this collection the report texts are real and are labelled for what they say (Section 5.3.3), and the rating fields are made from these text labels (Section 5.3.2). Both the gold and the rating fields thus rest on what the texts say, and the trait search reads the rating fields as evidence beside the texts, as it would in a club's database. (2) Their rule averages over the aspects of a query, so a hotel rated poorly for its location could still rank high for its clean rooms. The rule of this collection instead reads the traits of a question as requirements, so evidence against any one trait removes the player however strong his other traits are.

The sign therefore acts before any grade is assigned. A player with evidence against a trait is removed rather than given a negative grade. Like a graded judgement, nDCG assumes that grades run from not relevant upward, and since every grade then lies at zero or above, nDCG applies unchanged. The sign thus moves out of nDCG and into the definition of relevance. A measure that weights a player by his place, however, cannot say whether a removed player is in the result at all, and that is what the removal is meant to ensure. Harmful inclusion therefore counts how often a removed player still appears in the result, beside nDCG (Section 5.3.5).

What the collection gains over a pooled one is recall as a quantity, exact as far as the text labels are complete, and a place for evidence against a trait.

5.3.2 Trait Vocabulary and Corpus

The vocabulary holds 33 traits, the finest qualities a scout assesses in their own right, such as acceleration, top speed or heading. It was fixed before any report was read and is used in two directions, since the questions of the collection are composed from it and the reports are labelled against it. The traits are fine-grained because a coarse trait would make the gold claim too much. A question for a sprinter asks for top speed, and a coarse trait for pace would also make a player relevant who is quick over the first metres and slow beyond them, although he does not answer the question. Even a fine trait has finer parts, and a report may speak of only one of them, as a report that only says a player directs his teammates loudly speaks of one part of leadership, which also covers taking responsibility. Its text label then stands for the whole trait, so a question asks for whole traits and never for one part of a trait. A broader quality that a question names, such as pace, counts as one trait made up of several traits of the vocabulary. The trait search never sees the vocabulary. It identifies the traits of a question when it runs, without being limited to the vocabulary, and each of its model calls works with these traits, from dividing the question and rewriting each trait to matching the rating fields and judging the chunks (Section 3.4). The trait search is thus open to any quality a scout names, while the collection is closed. Its questions ask only for traits of the vocabulary, since relevant players can be derived only for a trait that every report was labelled for, so the evaluation says nothing about qualities outside the vocabulary. The vocabulary only records what the reports and the questions hold.

The 486 reports are real scouting reports, written in English by the scouts of one club, and they describe 70 players. The selection drew players rather than single reports, since the reports written about one player over time hold the agreement, contradiction and drift that the evaluation needs, in the form that real scouting produces them. How many reports a player has follows the club's own data, so most players have between 2 and 8 reports and a few up to 30. The reports are anonymised, since each player's reports are stored under another player of the database.

Generating the reports from their text labels was considered and dismissed. It would have made the text labels complete by construction, as the visual question-answering benchmark CLEVR renders each image from a scene it sampled first, so that nothing has to be annotated (Johnson et al., 2017). But a report that a model writes to express given traits is uniform in its wording and names the traits it was given. It reads as a model expects a report to read, so retrieval scores could rise without the trait search being better on the reports that scouts write. With real reports the text labels are not complete by construction, so the labelling has to make them complete and measure how far it got (Section 5.3.3).

The rating fields are made for the evaluation, for the reason given in Section 5.1, and their catalogue holds every kind of field the trait search reads (Section 3.4.3). The kinds vary, so the trait search has to read each in its own direction. (1) Of the 24 graded fields, 22 use the German school scale from 1 to 6, on which 1 is best. (2) The other two grade from 1 to 4, on which 4 is best. (3) A sprint over ten metres is timed in seconds, so a lower value is better and the values have no fixed steps. (4) Two fields list qualities as options, one for strengths and one for weaknesses, so a tick means the opposite in each. (5) Two fields offer a single choice. One names the player's main weapon, the one quality that stands out most, such as heading or acceleration. The other records his role, such as a box-to-box midfielder. A role describes a whole profile rather than one trait, and the system splits a role into the position and the traits it implies (Section 3.2.2). The matching therefore has to leave this field out, although a scout would value what it says.

The ratings are made to meet a few conditions. Apart from a few chosen players, which fields of a report are filled does not depend on what the report says, since a scout fills in the form whether or not he writes about a quality, so the number of a player's ratings on a field ranges

from one to many. A rating never contradicts its own report, so a report that praises a quality carries no weak grade for it and one that criticises the quality no good grade. Across players the grades follow what the reports say, and most of them lie in the middle of the scale, as on a real form. These chosen players depart from this on purpose, one rated well although his reports never mention the trait, one whose main weapon names another quality although his reports lean towards the trait, and one rated last although one of his reports praises it. His low grades sit only in his reports that are silent on the trait, so the contradiction runs between reports and never within one. These conditions are there so that the ratings behave as the ratings in a real club's database do. They remain made data, however, so they may differ from a real club's ratings in ways the conditions do not anticipate. Appendix C describes how they were made.

5.3.3 Labelling

A text label states for one report and one trait whether the text of the report speaks of the trait and, where it does, in which direction and how strongly, so it consists of the trait, a direction and a strength. The relevant players of every question are derived from these text labels (Section 5.3.4), so they are the one judgement the collection rests on. Every report was labelled in one call to a language model that answers for all 33 traits at once. Each trait is a required field of the answer, so the model has to state a silence rather than reach it by leaving a trait out, and the task becomes 33 bounded decisions per report instead of a free choice of what to mention. Where the report speaks of a trait, the text label gives its direction and strength on the judge's scale from -5 to $+5$ (Section 3.4.2), with every trait given its definition for the collection and every value of the scale described as in the judge's prompt. It also quotes the passages it rests on and records whether the report names the trait explicitly or only implicitly. A pilot on ten reports showed that values without a description were used less than the described ones, which is why every value carries one. The text label is therefore the score a correct judge would give what the report says about the trait, and the gold needs no translation between two scales. Figure 5.4 shows how the text labels were made and checked, and how the ratings were made from them.

A model labels rather than a person because of the size of the task and because a capable model has labelled relevance as well as people in web search. The 486 reports and 33 traits make about 16,000 decisions, most of them a silence and the rest a direction and a strength with the passage they rest on. A person would need months for them and would make mistakes as well. Two groups of trained assessors agreed only moderately on the same documents, and a model's labels predicted what searchers themselves preferred better than every group of human labellers it was compared with, once its prompt had been chosen against the searchers' own preferences (Thomas et al., 2024). Thomas et al. call their own case an easy one, since a model trained on web text labelled web text, and they leave private corpora such as these reports untested. The labelling model is Claude Opus 5, from another family than the two GPT-6 models under test. It reads one report at a time without its player, its club or the other reports, and it never sees a question.

The text labels record a model's reading of the reports rather than a scout's, and that has two costs. (1) A collection labelled by a model can rate a system that ranks with the same model too high and penalise systems that estimate relevance in another way (Faggioli et al., 2023). A model also scores models similar to it higher, so the risk extends to a system that runs on a model similar to the one that made the text labels (Goel et al., 2025; Kim et al., 2025a). Taking the labelling model from another family and labelling without a question reduce this cost, but they do not remove it, since accurate models err alike even across providers (Kim et al., 2025a). (2) How well a model's labels agree with human ones depends on its prompt (Thomas et al., 2024), so the text labels need checking.

The text labels were checked so that human judgement goes where it matters most. Checking all 16,000 decisions by hand would cost as much as making them, so two further calls per

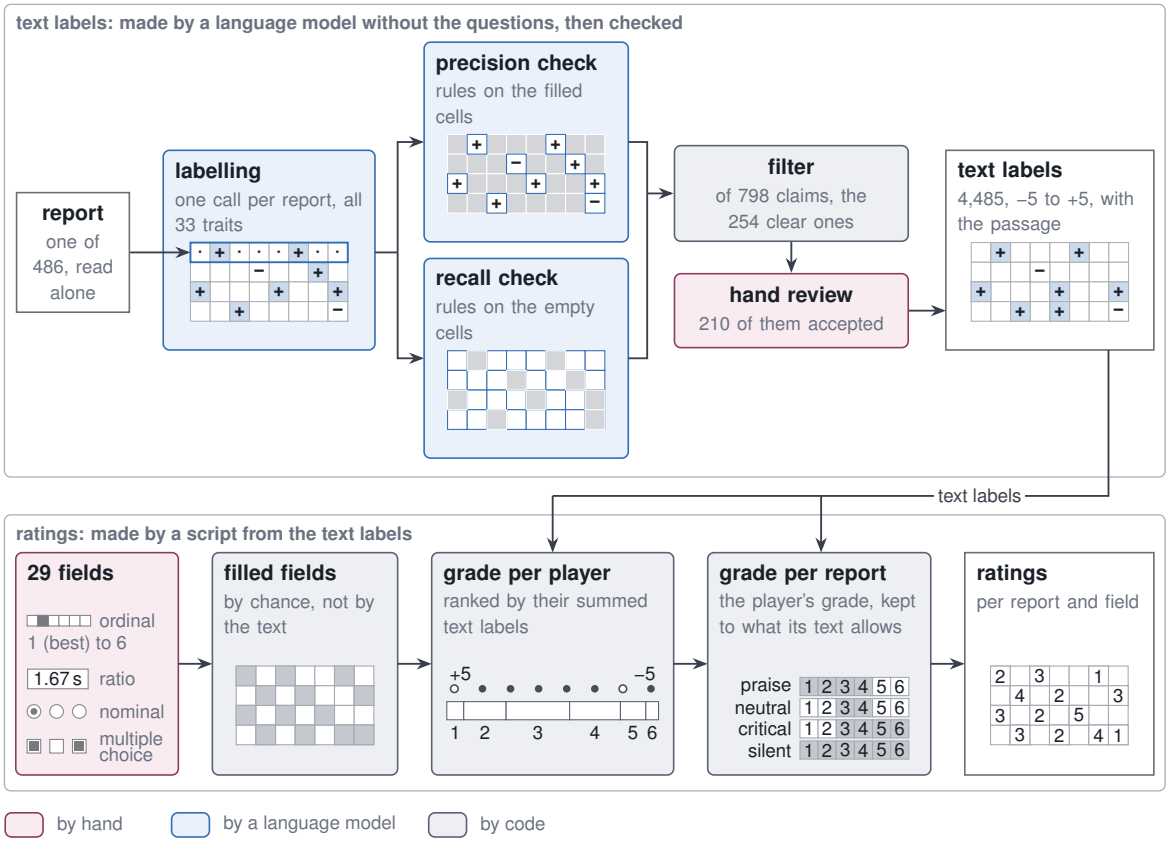


Figure 5.4: How the evidence behind the trait gold was made, with a few reports and traits standing in for 486 and 33. A language model labels each report against all 33 traits in one call and answers for every trait, a silence included. Two further calls check the labels. One rules on the labels a report carries and may keep, change or remove them, the other on the traits it carries none for and may add a label. Of their 798 claims, the 254 that move a strength by two values or more, or add or remove a label with at least medium confidence, were reviewed by hand, and 210 were accepted. A script then fills the 29 rating fields, followed here for a grade from 1 (best) to 6. Which fields of a report are filled is drawn by chance. The players are ranked by the sum of their text labels on the trait and the ranking is cut into grades, and the hollow dots mark players placed against their reports on purpose. Each report's grade is kept to what its own text allows.

report to the labelling model, each with a task of its own, read every decision once more and point out where they disagree. The first reads a report with its text labels and says for each text label whether it belongs and whether its strength is right. The second reads the report with the traits it was not labelled for and says for each whether the report expresses it after all, so that between the two checks every trait of every report is labelled once more. Both run on the labelling model, so they find what a second reading by that model finds, and they only propose. A proposal to add or remove a text label carries the checker's own confidence in it, low, medium or high, and a proposal to change a strength says by how many values. A proposal counts as a clear disagreement when it would move a strength by two values or more, or when it adds or removes a text label with at least medium confidence. A difference of one value does not count, since where one described value ends and the next begins is a matter of interpretation with no right answer. Of the 798 proposals, the 254 clear ones were reviewed by hand. The review accepted 120 removals, 62 additions and 28 changed strengths, in part with a value other than the proposed one, so the 4,543 text labels became 4,485. Figure 5.5 shows for each trait how many reports speak of it, in which direction and how strongly.

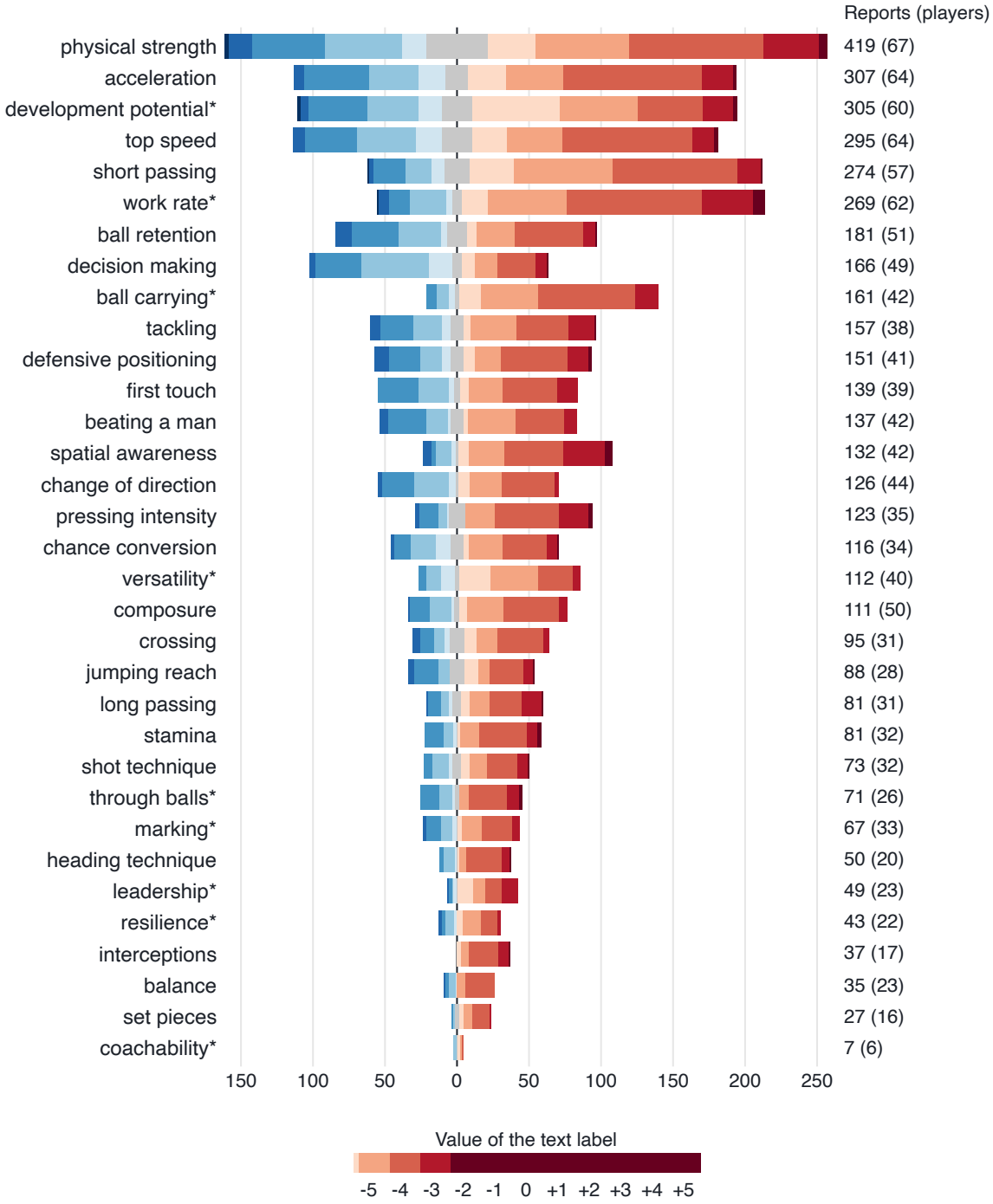


Figure 5.5: The 33 traits of the vocabulary by the text labels of the 486 reports, ordered by how many reports speak of each trait. A label against the trait lies left of zero and a label for it right of zero, and its colour gives its value. The labels of value 0 are split around zero. A trait marked with an asterisk is text-only, since no rating field measures it. The four traits at the bottom count as rare, since fewer reports speak of each of them than a third of the median trait. The column on the right gives the number of reports and, in brackets, of players.

5.3.4 Relevant Players

A player is relevant to a question when his reports and ratings speak for more than half of the traits the question asks for and against none of them. The trait search ranks players by this rule (Section 3.4.4), and the gold applies the same rule. The two differ in what the rule is applied to. In a run, four steps call a language model, (1) the division of the question into traits, (2) the query variants, which decide which chunks retrieval finds, (3) the matching of the rating fields and (4) the judge, which scores the chunks. A language model can decide each of these differently from run to run and is free in how it decides. The gold instead takes the outcome each step should reach, so nothing in it varies, (1) the traits of the vocabulary that the question was composed from, (2) every report that speaks of a trait, as if retrieval had missed none, (3) the fields that the vocabulary lists for a trait and (4) the score a correct judge would give each report for a trait.

The gold thus holds two sets for each question. The relevant players form the first, each with his score under the rule, which says how well he fits the question. The players whom the rule removes for evidence against one of the question's traits form the second. Section 5.3.5 describes how the measures use the two sets, and neither set needs a judgement beyond the text labels.

Only the ranking that the trait search returns is scored, and not its steps one by one, since each step works on the traits the trait search identifies when it runs, which need not match the traits of the vocabulary, so nothing fixed exists to score a single step against. A score therefore measures how close the division of the question, the retrieval, the matching of the fields and the judge together come to the outcomes that the gold sets. It does not measure whether the rule itself gives the list a scout wants, which is argued in Section 3.4.4 and not measured (Section 5.1.3).

In two places the trait search departs from the gold by design, so it can fall short of the gold even where every step reaches its outcome. (1) The trait search adds up the judged chunks of a player, while the gold adds up the text labels of his reports, since chunks are a device of retrieval and the gold should not change when the reports are cut differently. A report that speaks of a trait in three chunks therefore counts three times in the trait search and once in the gold. (2) The judge reads at most five chunks of a player and only the first 50 players of a trait, a budget that bounds the cost of its call, while the gold uses every text label of every player.

5.3.5 Measures and Populations

The trait search is scored on six measures, each taken per question and averaged over the questions (Table 5.4). A question divides the 70 players into the relevant players, the removed players and those who are neither. The measures count the relevant and the removed players over the whole result and in its first ten places, which matter most, so that every failure shows in at least one of them and none can rise alone without another one falling.

Table 5.4: The six measures of the trait search, by the group of players they count and by the kind of share they give.

Part of the result	Players	Share of the group that is in the result	Share of the result that belongs to the group
Whole result	Relevant	Recall	Precision
	Removed	Harmful inclusion	Harmful share
First ten	Relevant	–	nDCG@10, graded and weighted by rank
	Removed	–	Discounted harm@10, weighted by rank

Over the whole result, recall is the share of the relevant players that are in the result, since the trait search serves exploration, in which every player who fits a profile should come into

consideration (Section 3.4). Precision is the share of the players in the result who are relevant, and it is the only measure that falls when the result includes a player whose reports say nothing about the trait. Harmful inclusion and the harmful share are the same two shares for the removed players, as Zhang et al. (2025a) computed the recall of the document a query excluded beside that of the document it wanted. A random result of k players reaches $k/70$ on both recalls, whereas both precisions keep their random value at any length and so compare results of different length directly.

In the first ten only the share of the result that belongs to a group is measured, since whether a player is in the result at all is the question of the whole result, and a question has more relevant players than ten places. nDCG at rank 10 compares the order of the first ten with the ideal order of the gold and weights each place less the lower it stands (Järvelin and Kekäläinen, 2002), so it is a graded precision of the top that a longer result cannot raise. Discounted harm at rank 10 weights the removed players among the first ten by place in the same way, following the harmful score of the TREC Health Misinformation track (Clarke et al., 2021), but binary instead of graded, since how much worse one removed player is than another cannot be judged. With the cut at rank 10, both measures depend on the order of the result and not on its length, which the judge decides. Recall, precision and the measures of harm hold only as far as the text labels are complete.

Gienapp et al. (2020) instead kept the negative grade inside nDCG. They gave documents that assessors had marked as junk a negative gain and rescaled nDCG between the worst and the best possible ranking, so that such a document costs more the higher it stands. The worst ranking, however, depends on how many removed players a question has, so the same list of players who neither fit nor were removed would score 0 on a question without removed players and well above 0 on one with many. One number would also hide whether the relevant players sank or the removed ones rose, and how much a removed player should cost against a relevant one is a choice that nothing in the collection fixes. The measures therefore stay apart.

The 59 trait questions that ask for qualities in the usual way form the main population, over which the measures are averaged and the hypotheses tested. Six questions ask something else and are reported one by one. Two ask more traits than the trait search divides a question into, two ask that a player lacks a weakness, and two have no right answer and are scored on whether the answer is empty, since nDCG and recall are undefined without a relevant player.

A question of the main population has about a third of the 70 players relevant, so even a random ranking scores well above zero, and how far above depends on the question, so Section 5.3.7 sets the score of each question against the random level of that question. The results count both measures of harm only over the players excluded by text, whom the text of their reports removes while their ratings do not, on the 53 questions that have one, since a player whose ratings speak against the trait is removed with or without the judge, and only the judge reads the sign of the text. Recall and harmful inclusion depend on how many players a result holds, since a random result of k players reaches $k/70$ on both, so the two are read together and with the size of the result.

An average over the main population can hide how the trait search handles a particular situation, such as a player whose reports disagree on a trait, a trait for which his reports carry only implicit labels, or a player who fits every trait asked but one against which his reports speak. Most relevant and removed players are ordinary cases, so a trait search could reach a good average while failing in a situation that only a few players are in. Here such situations belong to players rather than to questions, since nearly every question holds a player in most of them. Each relevant and each removed player of a question is therefore marked with the situations he is in, by rules fixed before any run (Appendix D), and for each situation recall over its relevant players and harmful inclusion over its removed players are pooled across the main population. Two situations belong to a trait instead, a trait that few reports mention and a trait that no rating field

measures, on which every player is found through the text alone, so the main questions that ask such a trait are averaged against the rest. No hypothesis is stated about the situations, since they show where the trait search does well and where it fails on cases that real reports contain, which Chapter 6 takes up. The hypotheses compare two configurations or conditions of the trait search instead, with and without the judge, with one variant or two, and in English or German.

5.3.6 Hypotheses

Each hypothesis is tested on the measure of the first ten for the group of players it concerns, nDCG at rank 10 for the relevant players and discounted harm at rank 10 for the removed ones, since these depend on the order of the result and not on its length, which the judge changes most (Table 5.4). The first hypothesis tests the judge. A retriever scores how much a chunk is about a trait and barely tells a statement from its negation (Weller et al., 2024), so the judge reads the direction of each chunk (Section 3.4.2). With the judge switched off, every player among the first 50 that retrieval places for a trait counts as evidence for it, in the order of retrieval, and no text removes a player. These are the players the judge reads, so the two configurations differ only in the reading. Retrieval reaches nearly every player of the collection, so without the judge almost every player whose reports speak against a trait remains in the result. That is a given, and it is the reason the trait search removes such players at all. How much it matters is the open question, so the hypothesis is tested on discounted harm at rank 10. It is scored on the 53 of the 59 trait questions with at least one player whom the text alone removes, since the text of his reports speaks against a trait the question asks for while his ratings do not. Only those players count, since the ratings still remove a player when the judge is off. Harmful inclusion, reported beside it, puts a number on the given, how many of these players remain anywhere in the result without the judge and with it.

The second hypothesis tests the second query variant. The trait search searches each trait with two variants and fuses their lists by rank, since a second wording smooths the uncertainty of a small embedder on scouting prose (Section 3.4.2). The hypothesis compares the two fused variants with the first variant alone, so the division of the question and the rewriting into the register of the reports stay the same in both, and it is tested on nDCG at rank 10 over the 59 questions of the main population. A gain on average can hide losses on single questions, as Benham and Culpepper (2017) found when they fused several phrasings of a question for term-based retrieval. Whether an embedder keeps the same balance cannot be read from their study, so the trait search measures it for itself. Beside the mean, their risk reading therefore counts the questions whose nDCG rises or falls by more than a tenth and weighs a loss once, twice and six times as heavily as a gain, which shows how gains and losses balance in this trait search. Dividing the weighted mean by its standard error gives TRisk (Dinçer et al., 2014), which is read like the statistic of a t-test, so a value below -2, about the two-sided 5 % level, marks a real risk of doing worse than the first variant alone and a negative value above it does not.

The third hypothesis tests the language of the question. The scouts of a club write in their own languages, and the trait search places a question and the reports in one space with a multilingual embedder (Section 3.4.2). Testing that on the data of a club would need reports written in several languages about the same players. Translating the reports would replace the real reports the collection rests on with text no scout wrote, and it would only approximate such data, since a translation is never one to one and scouts who write in another language write their reports differently. The collection therefore changes the language of the questions and not of the reports. Each trait question is asked in German as well, while the reports stay in English as scouts wrote them, so the scores still rest on real data and the German questions test cross-lingual retrieval, a German question against English chunks. The trait search uses a multilingual embedder, and such embedders still lag behind English ones and are trained on data spread very unevenly over the languages (Chen et al., 2024a), so the hypothesis expects German questions to rank the players less well than English ones and is tested on nDCG at

rank 10 over the 59 questions of the main population. It shows what answering a scout in his own language costs.

5.3.7 Results

How well the trait search ranks. In the standard configuration, which runs the judge and both variants on the English questions, the trait search ranks the relevant players far above a random ranking on both models, at an nDCG@10 of 0.78 against 0.22 (Table 5.5). Its result of about 25 players holds about 70 % of the relevant players, twice the share that a random result of the same size holds. Of the players excluded by text, whom the text of their reports removes while their ratings do not, it holds less than half the share of such a random result, and its discounted harm@10 is about a third of a random ranking's. Sol and Luna rank alike, only 0.001 nDCG@10 apart, although Sol costs 18 times as much per question. The judge makes up 96 % of the cost of a question on both models.

Table 5.5: The trait search on the 59 questions of the main population in each configuration on GPT-6 Sol and GPT-6 Luna, against a random ranking of the 70 players.

Configuration	Model	nDCG@10	Recall	Precision	Harmful inclusion	Harmful share	Discounted harm@10
Standard	Sol	**0.779** ± 0.056	0.721 ± 0.058	**0.688** ± 0.051	0.180 ± 0.060	0.046 ± 0.016	0.027 ± 0.015
	Luna	**0.777** ± 0.058	0.693 ± 0.065	**0.725** ± 0.053	0.144 ± 0.053	0.045 ± 0.017	0.026 ± 0.014
w/o judge	Sol	0.737 ± 0.059	**0.884** ± 0.026	0.529 ± 0.051	0.572 ± 0.071	0.086 ± 0.021	0.054 ± 0.025
	Luna	0.744 ± 0.054	**0.884** ± 0.028	0.529 ± 0.050	0.587 ± 0.076	0.086 ± 0.021	0.051 ± 0.022
One variant	Sol	0.776 ± 0.058	0.717 ± 0.060	**0.688** ± 0.051	**0.174** ± 0.060	0.050 ± 0.019	0.032 ± 0.018
	Luna	0.768 ± 0.062	0.686 ± 0.066	0.724 ± 0.054	**0.129** ± 0.051	**0.041** ± 0.016	**0.022** ± 0.014
German	Sol	0.764 ± 0.059	0.700 ± 0.057	0.685 ± 0.054	0.195 ± 0.067	**0.044** ± 0.015	**0.025** ± 0.014
	Luna	0.736 ± 0.065	0.679 ± 0.060	0.669 ± 0.058	0.202 ± 0.066	0.048 ± 0.015	0.026 ± 0.014
Random ranking	Sol	0.220 ± 0.021	0.368 ± 0.038	0.355 ± 0.039	0.368 ± 0.041	0.084 ± 0.020	0.084 ± 0.020
	Luna	0.220 ± 0.021	0.341 ± 0.041	0.355 ± 0.039	0.341 ± 0.044	0.084 ± 0.020	0.084 ± 0.020

Each value is the mean over the questions with the half-width of its 95 % confidence interval. The three measures of harm count only the players excluded by text, over the 53 questions that have one. A random result of k players reaches $k/70$ on recall and on harmful inclusion, which is why each random row takes the result size of the standard configuration on that model, and the share of relevant and of excluded players on precision and on the harmful share at any size. An empty result counts as precision 0. Within each model, bold marks the best value in a column, the lowest for the three measures of harm.

The trait search ranks above a random ranking on 58 of the 59 questions on Sol and on 57 on Luna, by 0.56 nDCG@10 on average (Figure 5.6, left). It places the players excluded by text lower than a random ranking on 47 and 48 of the 53 questions that have one, and on 38 of them no such player reaches the first ten in any run (Figure 5.6, right). All but one of the questions above the random level on the right ask several traits. The questions below the random level on the left, one on Sol and two on Luna, all ask for coachability, for which only 7 of the 486 reports carry a label, all of them implicit, and no rating field exists. On the questions below the random level retrieval ranks the strongest labelled passage too low for the judge to see it, and the judge

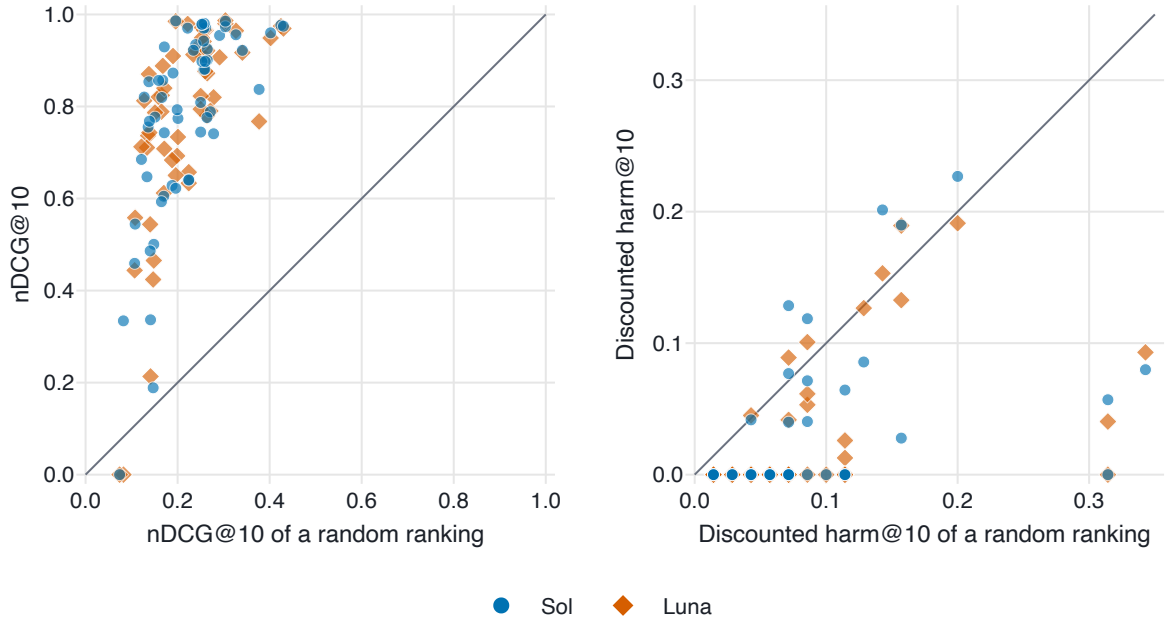


Figure 5.6: The trait search ranks the relevant players higher than a random ranking on 58 of the 59 questions of the main population with GPT-6 Sol and on 57 with GPT-6 Luna (left), and the players excluded by text lower on 47 and 48 of the 53 questions that have one (right). Each point is one question with one model, averaged over five runs, against the value that a random ranking of the 70 players reaches on that question. The left panel plots nDCG@10 and the right panel discounted harm@10. A random ranking lies on the diagonal, so a point above it does better on the left and a point below it does better on the right. The points on the bottom edge of the right panel, 38 with each model, are questions on which no player excluded by text stands in the first ten in any run.

scores most of the others as saying nothing about the trait, so the result is empty or holds one player, and an empty result scores below a random ranking.

The six questions outside the main population rank well above their random levels or return the empty result they call for. The two that ask for the absence of a weakness, in the air and on the ball, reach an nDCG@10 between 0.91 and 0.96 on both models, while their random levels lie below 0.40. The division returns five traits in every run for the two questions that ask six and seven, and these reach between 0.72 and 0.88, against random levels of 0.10. The two questions without a right answer, for players who write clean code and for players who can plate a five-course menu, return an empty result in every run on both models.

The configurations move the measures together, since the size of the result decides much of recall, precision and harm (Table 5.5). Without the judge the result grows from about 25 to 42 players, so recall rises from about 0.7 to 0.88. Precision falls from about 0.7 to 0.53, and the harmful inclusion of the players excluded by text rises from below 0.2 to almost 0.6. Players excluded by text then take 8.6 % of the result, as much as in a random result, against 4.6 % with the judge. The first variant alone and the German questions change the measures far less, except that German questions lower nDCG@10 on Luna.

The hypotheses. Of the three hypotheses, only the one on the judge is supported, since its test rejects on both models (Figure 5.7). Without the judge, 0.6 players excluded by text per question stand in the first ten on Sol and 0.5 on Luna, about twice the 0.3 with the judge, and there are more of them at almost every single place (Figure 5.8). For the hypothesis on the

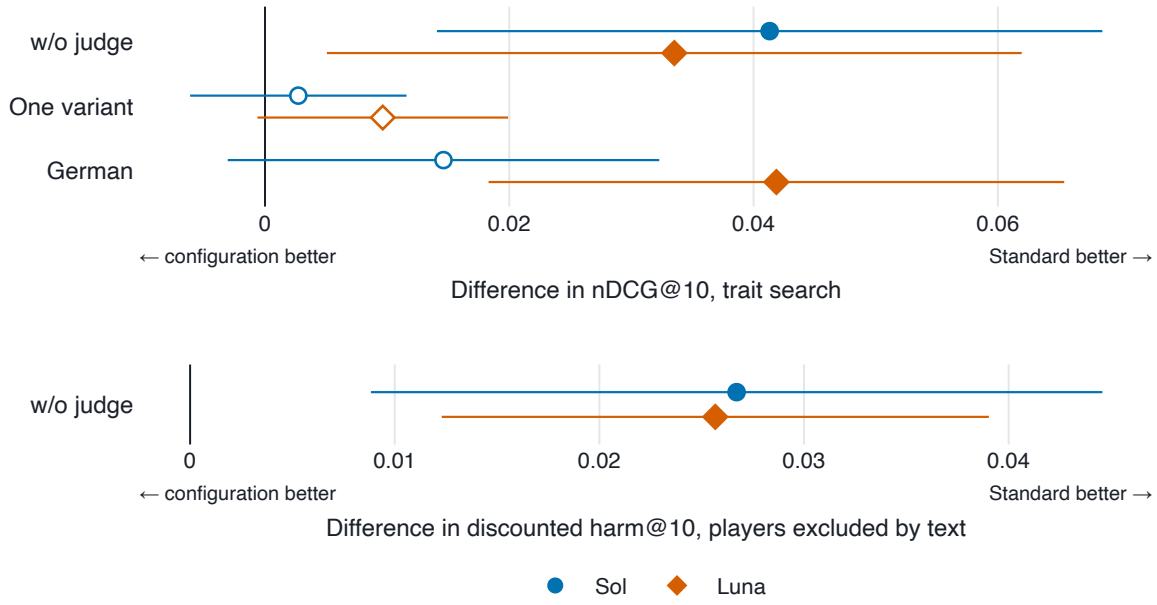


Figure 5.7: Only the hypothesis on the judge is supported, since without the judge players excluded by text stand higher in the result on GPT-6 Sol and GPT-6 Luna. Each row gives how much better *Standard* scores than the configuration the row names, as the mean over the 59 questions of the main population for nDCG@10 and over the 53 with a player excluded by text for discounted harm@10. For a hypothesis the bar spans the lower and the upper one-sided 95 % confidence bound, and for the judge on nDCG@10, which tests no hypothesis, it spans the two-sided 95 % confidence interval. A filled marker marks a difference whose test rejects.

second variant the test rejects on neither model, so it is not supported. For the hypothesis on German questions the test rejects on Luna, where German questions lose 0.042 [0.018, 0.065] nDCG@10, but not on Sol, so it is not supported either. Where a test does not reject, its upper bound limits the effect of the second variant to at most 0.020 nDCG@10 on both models and that of German questions on Sol to at most 0.032. Appendix I gives the p-values and bounds of every test.

The risk reading shows that the small mean gain of the second variant rests on a few large gains against many small losses. Of the chunks that either variant returns among its first 500 for a trait, on average 0.76 are returned by both, but of their first ten they share only about half and of their first fifty about two thirds, so the two variants differ most near the top of the ranking. With the second variant, nDCG@10 rises on fewer than half of the 59 questions and falls on more than half on both models. By the count of the risk reading, nDCG@10 rises by more than a tenth on 6 and 8 questions and falls by as much on 2 and 1, so the large changes are mostly gains and the many small ones mostly losses. With a loss weighted twice as heavily as a gain, TRisk is -1.3 and -0.1, negative but above -2 and so no real risk, and only with a loss weighted six times as heavily do the small losses pull it below -2, to -3.2 and -2.6.

Where the trait search fails. In two cases the trait search does clearly worse than on average. (1) The questions that ask a text-only trait reach an nDCG@10 of 0.63 against 0.87 for the other questions. Both groups start from about the same random level, so the lower nDCG@10 of these questions does not come from a lower random level. Most of the evidence for these text-only traits is implicit, with 70 % of their labels implicit against 49 % of the labels of the traits with a rating field. Within one trait, the result holds a relevant player whose evidence for it is only implicit less often than one with an explicit label, by 15 percentage points [7, 23] on Sol

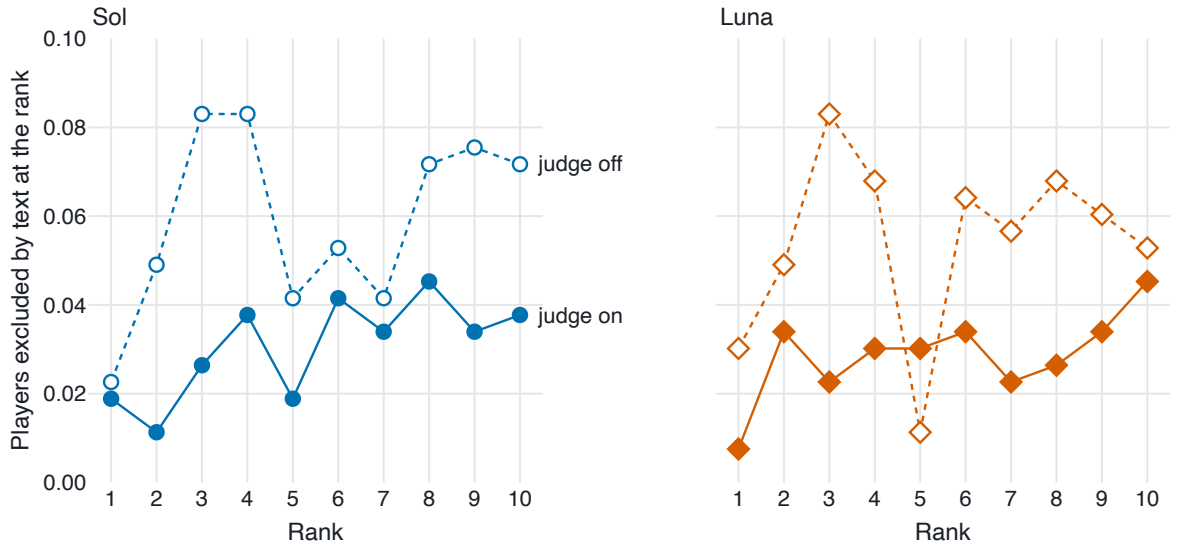


Figure 5.8: Without the judge, more players excluded by text stand at almost every one of the first ten places, at all ten with GPT-6 Sol and at nine of them with GPT-6 Luna. Each point is the mean number of such players at a place, over the five runs and the 53 questions that have one, with the judge (filled, solid line) and without it (hollow, dashed line).

and 17 [9, 25] on Luna. The cause lies in retrieval, which ranks an implicit passage lower than an explicit one of the same trait, and the lower rank costs twice. About half of the gap remains without the judge, since such a player reaches the 50 players read for a trait less often. Most of the other half arises because the judge reads only the five chunks of a player that rank highest, and an implicit passage is less often among them. Once the judge reads a passage, it scores an implicit one as evidence for the trait about as often as an explicit one on Sol and somewhat less often on Luna. (2) As expected, the removed players whose reports support every asked trait but the one they speak against are the hardest to keep out of the result. The judge lowers their harmful inclusion only from about 35 % to about 25 %. For the other removed players, whose reports lack evidence for at least one of the other asked traits or speak against the only trait a question asks, it lowers their harmful inclusion from about 25 % to about 8 %, on both models. Tables D.8 and D.9 in Appendix D give the values for every situation and every group of traits.

No relevant player is lost when the trait search divides the question into trait phrases, and most are lost in retrieval and at the judge. In every run this division returned the trait phrases of the question correctly. Each relevant player missing from the result in a run is assigned to the first step that lost him in that run. About 60 % of these players are lost because retrieval ranks them or their passages too low, at three points. (1) For about 8 %, retrieval returns no chunk of the player. (2) For about 20 %, the player ranks below the 50 players the judge reads. (3) For about 33 %, none of the labelled passages of the player is among the five chunks the judge reads. The judge's own decision causes 24 % of the losses on both models. For 13 and 16 %, it read a labelled passage but did not score it as evidence for the trait, and for about 10 % it found evidence against the trait that the gold does not count. The ratings cause the remaining 14 and 16 % of the losses (Figure E.1 in Appendix E). The rating fields matched to a trait phrase equal those of the gold for 75 % of the phrases on Sol and 72 % on Luna, and wrong fields cause 6 and 8 % of the losses. Of the players in the gold's first ten that the trait search misses in its own first ten, more than half are still in the result at a lower place, and the others are lost at the steps above.

Further looks at the runs. The retriever ranks a chunk higher the more strongly it speaks for a trait. Reports speak for a trait more often than against it, and 25 % of the evidence in the corpus speaks against the trait (Figure 5.5). On the questions that ask a single trait, this share falls to 6 % among the first ten chunks that retrieval returns before the judge reads them, so retrieval favours evidence for a trait beyond its share in the reports. Against chunks that do not concern the trait, however, evidence against it ranks higher with a probability of only 0.53, so retrieval ranks it almost like text about something else. Over all 1,808 chunks of the corpus, a chunk moves up steadily with the signed strength of the text label of its passage, from the middle of the list at -5 to the first quarter at $+5$ (Appendix E). Of the chunks that speak most strongly for a trait, at $+4$ and $+5$, the judge reads about 25 %. Of those that speak most strongly against it, at -4 and -5, it reads under 10 %. So the search queries hide much of the evidence against a trait, although Section 3.4.2 expected the instruction and the rewritten traits to find it as well. As a result, on the questions that ask a single trait the judge excludes only about 20 % of the players excluded by text because it read evidence against the trait, and most of the others leave only because it finds no evidence for them.

When the judge read a passage that the labels mark as speaking for the trait, it scored it as speaking for the trait as well in 89 % of the cases on Sol and 82 % on Luna (Appendix E). When the passage speaks against the trait, it agreed in only 67 and 64 %.

5.4 Router

5.4.1 Route and Division Gold

The gold of every question gives its route and its division into a fact part and a trait part, and the route follows from the conditions the question holds. A question whose conditions are all facts takes the fact route, one whose conditions are all traits the trait route, and one with both the hybrid route (Section 3.2.2). A hybrid question is scored on its route and its division alone and not on the players it returns, since after the filter of its fact part too few of the 70 players in the corpus remain for a ranking by traits to mean anything.

The division of a question is written as text, in the form the router itself produces, since the router divides the question before any query is written. For "Show me left-footed players of at least 1.83 m with the habit of switching the play" the gold fact part is "left-footed players of at least 1.83 m" and the gold trait part "the habit of switching the play". A part the question does not need stays empty, such as the trait part of a fact question. Two correct divisions can word the same condition differently, such as "at least 1.83 m" and "1.83 m or taller", so the router's division is compared with the gold by its meaning and not by its words. Four hybrid questions were written to test the router where the wording of a condition points away from its part, such as a player profile like "a classic number ten", which the router has to split into an attacking midfielder for the fact part and the qualities of a classic number ten for the trait part. Most questions make clear which part each condition belongs to, and these four are edge cases.

5.4.2 Measures

Route accuracy is the share of questions the router sends down the right route, since a wrong route makes a right answer improbable. A hybrid question taken on the fact route loses its trait part, for instance, and one taken on the trait route loses its filter. The router chooses the route from three set values, so code compares it with the gold route.

Whether a part of the router's division is the same as the gold part is decided by a language model, called the division checker below (Zheng et al., 2023), since the router may word a part differently from the gold. A part is the same when a search with it looks for exactly the players a search with the gold part looks for, so other words for a condition keep it the same, such as "younger than 23" for "under 23". A part is not the same when it misses a condition or adds one, or when it holds a condition that belongs to the other part. It is not the same either when a

number the question states changes its value or its comparator or becomes a word, such as "young" for "under 23", or when a quality becomes narrower or wider. A part empty in the gold and in the router's division is the same and a part empty on one side only is not, which code decides without the model. Verdicts on a player profile are the most interpretive, since a profile such as "a classic number ten" counts as the same when the trait part describes qualities that fit it.

The division checker is Claude Opus 5. Like the model that labelled the reports, it belongs to another family than the two models under test, since a model that rates other models scores those similar to it higher (Goel et al., 2025; Kim et al., 2025a). It receives the question, then each gold part beside the router's part, and it answers for each part whether the two are the same, with one sentence naming what differs. Each part is checked apart, since the fact part goes to the MQL sub-agent and the trait part to the trait search, so a part that is not the same names the component a router error reaches. Its prompt states the rules and gives examples written for it, none taken from the 154 questions. Model and prompt are fixed and given in Appendix B. Its verdicts were not checked by hand. The test of the router does not rest on them, since it scores the components behind the router against the gold, so only the share of matching divisions depends on the division checker. The division checker therefore matters most for the hybrid questions, whose division no end-to-end score measures, while for fact and trait questions the comparison of end-to-end with isolated runs shows directly whether the division hurt.

5.4.3 Hypothesis

The router has one hypothesis, on what the router costs the component behind it, and it is tested twice, once for the MQL sub-agent and once for the trait search. It compares their end-to-end runs with their isolated runs (Section 5.1.2) and expects both to score lower end to end. The router decides once and no loop returns to it, so a wrong route or a part that is not the same cannot be repaired by the component after it (Section 3.2.3). The isolated runs are an oracle that always takes the right route, as Jeong et al. (2024) used one to judge Adaptive-RAG. The MQL sub-agent is compared on the success rate over the 41 fact questions and the trait search on nDCG@10 over the 59 trait questions of the main population. Beside each test the share of its isolated score that a component keeps end to end is given, with its interval.

Route accuracy and the division are not tested themselves, but they show where a loss behind the router arises. A router's decisions and the answers behind it need not agree. The classifier of Adaptive-RAG agreed with its automatically assigned strategy, the simplest one that had answered, for 54.5 % of the questions, and a classifier trained on these labels alone agreed more often but answered worse (Jeong et al., 2024). Its strategies retrieve more or less for the same question, so a question sent to a costlier one than it needs can still be answered. Here a wrong route drops a part of the question, so the end-to-end result should stay close to route accuracy, and the two can come apart mainly through the division. A division that is not the same can still lead the component to the right players, and one that is the same can still end in a wrong answer. Whether a failure comes from the router shows in the isolated runs. A failure that the component also shows there points to the component, and one that arises only end to end points to the router.

5.4.4 Results

How well the router routes and divides. The router sends every fact question and every trait question down its route in every run on both models, and it misroutes only hybrid questions, on Sol one question in all five runs and on Luna one in two of its five runs (Table 5.6). Sol routes no better than Luna, although it costs 20 times as much per question. At the median a question takes about 2 seconds in the router, and the router adds at most 3 % to the cost of a question it routes. By the verdict of the division checker, the router divides about 98 % of the questions as the gold does on both models. The four edge cases, whose wording points a condition towards

the wrong part, are routed and divided as the gold does in every run except two runs of one of them on Luna.

Table 5.6: The router on the 154 questions in five runs each on GPT-6 Sol and GPT-6 Luna, by gold route.

		Runs sent to				The same as the gold		
Gold route	Model	Fact	Trait	Hybrid	Right route	Fact part	Trait part	Division
Fact (43 questions, 215 runs)	Sol	215	0	0	1.000 [0.918, 1.000]	0.991	1.000	0.991 [0.902, 0.999]
	Luna	215	0	0	1.000 [0.918, 1.000]	0.991	1.000	0.991 [0.902, 0.999]
Trait (65 questions, 325 runs)	Sol	0	325	0	1.000 [0.944, 1.000]	1.000	0.975	0.975 [0.903, 0.994]
	Luna	0	325	0	1.000 [0.944, 1.000]	1.000	0.985	0.985 [0.918, 0.997]
Hybrid (46 questions, 230 runs)	Sol	5	0	225	0.978 [0.886, 0.996]	0.965	0.970	0.961 [0.861, 0.990]
	Luna	0	2	228	0.991 [0.907, 0.999]	0.974	0.987	0.970 [0.874, 0.993]
All (154 questions, 770 runs)	Sol	220	325	225	0.994 [0.965, 0.999]	0.987	0.981	0.975 [0.936, 0.990]
	Luna	215	327	228	0.997 [0.970, 1.000]	0.990	0.990	0.982 [0.946, 0.994]

The fact questions include the two that ask for facts the database does not hold. Whether a part is the same as the gold is the verdict of the division checker. A part empty in the gold and in the router's division counts as the same, so a misrouted run whose route drops a part counts as not the same. The counts are runs. Each share is the mean over the questions of the share of their five runs, and below it stands its 95 % Wilson interval over the questions (Wilson, 1927).

The hypothesis. End to end, the MQL sub-agent answers 92.2 % of the fact runs correctly on Sol and 80.0 % on Luna, against 89.8 and 82.0 % in the isolated runs. It therefore keeps about its isolated success rate on both models, 103 [98, 109] and 98 [93, 102] %. By the bound of its test, the router costs it at most 1.7 percentage points of success on Sol and 5.4 on Luna. The trait search keeps 99 and 97 % of its isolated nDCG@10. On Sol the router costs it at most 0.019, while on Luna its loss of 0.022 [0.008, 0.036] is significant. Since the hypothesis needs a significant loss on both models, it is supported for neither component (Figure 5.9). Appendix I gives the p-values and bounds of both tests.

Where the router fails. On Sol all five runs of the question for centre backs with "a high return in front of goal" take the fact route, since the router reads the goals as a stored statistic and leaves the trait part empty. On Luna the two misrouted runs belong to the edge case that asks for a left foot and a dangerous ball from a corner or a free kick. They take the trait route and move the left foot into the trait part. Where the division checker found a part not the same, the router had mostly added or dropped a condition, as when two runs on Sol of a hybrid question turned "defenders at Premier League level" into defenders of any league (H06). Since every fact and every trait question took its route in every run, no wrong end-to-end answer of either component followed a wrong route. Where the component also fails in its isolated runs, the router is not the cause. Only a few failures arise behind the router alone, 3 and 7 of the fact runs on Sol and Luna. Of the trait questions, five carry 70 % of the loss of nDCG@10 on Luna, and on each of them the mean of the five end-to-end runs lies below the lowest of the five isolated runs. These failures have two sources. (1) The router adds or changes a condition. It limits the youngest player of each LaLiga squad to the most recent season, so that on Sol the sub-agent took the squads

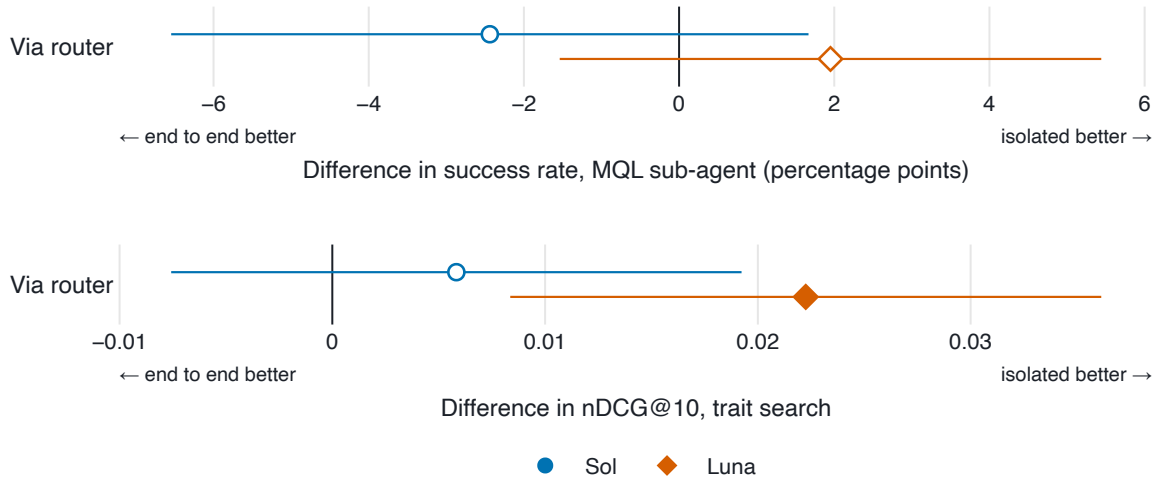


Figure 5.9: Behind the router only the trait search on GPT-6 Luna scores significantly lower. Each row gives how much better a component scores in its isolated runs than behind the router, as the mean over the 41 fact questions for the success rate in percentage points and over the 59 questions of the main population for nDCG@10. A bar spans the lower and the upper one-sided 95 % confidence bound. A filled marker marks a difference whose test rejects, which on the fact questions needs both the t-test and the sign-flip test.

from that season's statistics instead of the current squads (M25). It also drops who flagged a loan target as an injury risk (M28) and asks for a peak instead of a current market value (M16). (2) It rewrites an idiomatic trait part into a longer paraphrase that names further qualities, as when "the character to bounce back from a mistake" becomes resilience, composure and a positive response (T24), and the trait search then ranks the players less well than in its isolated runs. On Luna the router's trait parts have a median of 1.4 times as many words as the gold part, and one of the 295 repeats it word for word. On Sol the median part is as long as the gold part, and 54 of the 295 repeat it word for word. The division checker judged most of these parts the same as the gold, so it misses changes that matter to the component behind the router. The shares of matching divisions are therefore probably too high, most of all for the hybrid questions, which only the division checker scores. On three fact questions the router's part means the same as the gold (M17, M19, M41), and on Luna the two failing runs of M25 also built the squads as the gold does, so these runs fail by the variation of the MQL sub-agent from run to run.

6 Discussion

Chapter 5 reported what the components scored, and this chapter discusses why they scored so and what that means for the design. It takes the components in the order of Chapter 5, the MQL sub-agent, the trait search and the router, then the choice of a model for each of them, and it ends with the limitations of the design and of the evaluation. Where two numbers are given, the first belongs to Sol and the second to Luna.

6.1 MQL Sub-Agent

The techniques that the context of the MQL sub-agent took from the literature changed few questions, because the sub-agent went wrong mostly in how it understood the question and the data, and hardly ever in the queries it wrote. Every fact question apart from 14 on Sol and 21 on Luna was answered correctly in all of its runs, so the configurations could move only these few (Section 5.2.4). Their wrong runs most often took the answer from the wrong collection or field and hardly ever joined two collections wrongly. The two questions that stood out failed for two different reasons, and neither lay in writing the query. On one the runs missed a nuance of the question and answered the neighbouring question that the stored season totals answer, goals per game (M31). On the other Luna lacked knowledge that the question takes for granted, when the summer transfer window ends, and filled it with an assumption that cut the window too early, whereas Sol answered it correctly in most runs (M40). Worked examples and schemas change how the sub-agent finds the data, whereas the failures that remained lay in the nuances of a question and in knowledge that neither of them holds. In about a fifth of the wrong runs the sub-agent named the reading it took or asked the user back, which its prompt allows but the evaluation counted as wrong (Table D.11). In use, these failures would reach the user as a question rather than as a wrong answer, and the evaluation, which wrote its questions to leave nothing open, would have measured the sub-agent more fairly had it told the sub-agent to go through with its reading.

Of what the worked examples carry, the notes of a collection carry most as well, which is why the examples changed the path of a run more than its success. Much of what makes the club's database hard to query lies in its redundancy, since it stores many facts more than once, in several collections and in more than one form, and the copies are not always updated together (Section 3.3.3). TEND's authors likewise place the difficulty of MQL in the documents rather than in the syntax (Lu et al., 2026). The notes state which copy to trust and how a value is written, so this knowledge reached the sub-agent with every schema read, with or without the examples. Without the examples, the rules that the notes also state were still applied, and a run took about half a step more, mostly to explore the data (Section 5.2.4). The sub-agent thus found by probing much of what an example would have shown. In the systems that the literature measured, examples raised accuracy far more (Gao et al., 2024). These systems wrote a query in one call, however, so the model had to take from an example what an agent can read in the notes or probe in the data. For the sub-agent the examples are therefore a shortcut that saves steps rather than a condition of a correct answer.

What the examples still carried differed between the two models. The studies of MQL did not separate whether examples make up for syntax a model lacks or for structure it has to copy (Sivasubramaniam et al., 2024; Qin et al., 2025), and in these runs they served each model with a different one. Without them Luna wrote more than twice as many queries that the database

rejected, whereas Sol's rejected queries stayed few (Section 5.2.4), so for the smaller model the examples carried the form of a valid query. About half of the Luna runs with such a query still ended correct, since the sub-agent repaired it, and the rejections therefore cost steps and time rather than success. For Sol the examples mattered only where they carried what the notes left out, since on two league questions they kept it from trusting a club's own league field over the league's list of clubs (M19, M33), a case of redundancy whose rule belongs in the notes of these collections. Both readings come from comparisons chosen after the runs and not tested, so they suggest the difference rather than show it.

The retrieved examples met two recommendations that point in different directions, and neither of the two predicted effects appeared. R3 of Table 2.4 recommends keeping out content that relates to the question but does not contain its answer, since in question answering such passages lowered accuracy most (Cuconasu et al., 2024). The text-to-query literature, by contrast, chose examples by their similarity to the question, and such examples worked better than fixed ones (Sivasubramaniam et al., 2024; Gao et al., 2024). Retrieved examples are therefore both content that R3 would keep out and the examples that the text-to-query literature recommends. They lowered the success rate significantly on neither model against the examples tied to a collection, and on the four questions for which none of them read a collection of the gold query, they cost no correct run against no examples (Section 5.2.4). The success rate probably did not move because both effects occurred. An example shows how a query is written in MQL and roughly how a problem of its kind is solved, whichever collection it reads, whereas a detail close enough to be copied can mislead, as when Luna applied an example's filter for the current season to a coach's whole time at the club (M42).

The two recommendations of Table 2.4 that conflict for an agent over a database, R5 and R8, showed neither of their predicted effects at the size of this schema. Preloading every schema saved about one step per run at about twice the tokens, and no difference in success was detected (Section 5.2.4). R8 guards against a needed column that a schema linker drops (Maamari et al., 2024). Reading on demand can miss a collection that a question needs in the same way. A schema linker decides once before the query is written, however, whereas the sub-agent can read another schema whenever a result shows that it needs one. R5 guards against the loss of accuracy with length (Levy et al., 2024; Hsieh et al., 2024a), which did not appear either. Those studies measured models of 2023 and 2024, whose windows were smaller and whose capability was lower, so neither their lengths in tokens nor their shares of the window carry over to the fullest preloaded prompt, which filled about 5 % of the window. At this size the choice between the two therefore rests on cost and time. The provider served the extra tokens of preloading almost entirely from its cache, and a preloaded run was even faster on Sol. Reading on demand would matter once the context fills, since in a chat session every run also carries the memory of the runs before it (Section 3.3.3). A session about a few collections then needs only their schemas rather than the whole schema, and the evaluation, which asked each question on its own, did not test such a session.

The runs did not bear out the premise of the step limit that a long run has gone in a wrong direction. The longest runs were mostly searching for a stored name and were more often correct than wrong, and only one question ever reached the limit (Section 5.2.4). The runs therefore do not set the limit, and where it lies is a choice between waiting and being asked. A run that reaches the limit ends with an answer that states what it tried and what the user would have to specify (Section 3.3.2). A lower limit thus hands a question back to the user sooner, as BIRD-INTERACT let an agent ask its user instead of guessing and found its success rising with the turns it could ask (Huo et al., 2026). A higher limit, by contrast, lets the user wait longer for an answer that needs no turn of his own.

German fact questions were answered at least as often as English ones on both models, against the expectation that the loss MultiTEND measured in other languages carries over (Qin et al., 2025). The expectation rested on the gap between the words of a question and the names the

data stores, which a German question would widen. MultiTEND translated the field names along with the questions and kept the stored values English, whereas here both stay English, so the gap would have been wider still. The English questions, however, already name most values in other words than the data stores, since 27 of the 41 were written that way on purpose to make the set harder (Section 5.2.1), so the sub-agent had to infer the stored form in either language. A German question therefore added little to a gap that was already there, and where its wording lay closer to the stored name by chance, it even did better. On the question for the final selection on the scouting board, which the data stores as the status `Shortlisted`, every German run took *Endauswahl* for that status, whereas some English runs read the same phrase, "final selection", as the board's last stage (Appendix H).

How the component that writes the database queries should be built is answered here by argument, since the comparison with a fixed workflow of the same steps was not run (Section 5.1.3). On both benchmarks on which a fixed workflow outscored an agent, the workflow read the stored values before it wrote the final query (Deng et al., 2025; Lu et al., 2026). The sub-agent takes this step as well, since its instructions tell it to check the stored values before it answers, and its loop adds only the freedom to repeat or extend a step when a result calls for it. That freedom can cost in two ways, a run that does not end its exploration and a run that spends steps checking its own answer. In the runs both stayed small, since a run took about 2 steps more than the shortest path of its gold query and only one question reached the step limit (Section 5.2.4). The agent thus keeps the step that the workflows showed to be worth having and adds the freedom of the loop, whereas whether that combination outscores a workflow with the same steps was not tested.

The fact questions start from the product's own player search, whose filters were built with clubs, so they ask for what scouts look for. Some of them are searches that the product can already run, and most extend such searches to combinations or facts that its filters do not offer, so the set is realistic and harder than what the product does today. The success rate on the searches that the product can run shows whether the sub-agent reaches what scouts can already do, and the rate on the others how far it goes beyond that. The questions run on a database that holds the platform's football data from production and club data made for the evaluation with deliberate variety. The database is as messy as databases in production often are, since its football data hold many facts in copies that do not always agree, so the sub-agent has to find which copy to trust and how a value is written. Most questions also name a value in other words than the database stores it. Each gold query follows the path the sub-agent is built to take, so the steps a run takes beyond that path measure what its freedom costs. Each question also carries tags for what makes it hard, and each wrong run the cause of its first error, with causes added for a wrong stored value, a failed process and a reading that the question allows (Section 5.2.2). Together they showed that the sub-agent answers what the product's search answers and that where it fails, it has mostly misread the question or the data rather than written a faulty query.

6.2 Trait Search

A player reaches the right place in the result of the trait search only if every step passes his evidence on. The division of the question has to name the trait and write its search queries, and the retriever has to rank the chunks that speak about the trait at the top, so that they are among the five chunks per player of the first 50 players that the judge reads. The judge then has to score each of these chunks in the right direction, and the matching has to pick the rating fields that measure the trait. In the runs the division lost no relevant player, the judge about a quarter of the lost ones and the rating evidence the rest, whereas retrieval lost about 60 % of them (Section 5.3.7). Most of the evidence is thus lost before the judge reads it.

The first failure of retrieval concerns the direction of the evidence. The design of Section 3.4.2 expected the rewritten traits, which read like praise, not to hide the passages that deny a trait, since retrievers barely tell a statement from its negation (Weller et al., 2024). Retrieval

instead followed the direction of the search query, since it ranked a chunk higher the more strongly it spoke for the trait and ranked the evidence against it almost like text about something else (Section 5.3.7). Whether current retrievers fail on polarity as they fail on negation was unmeasured, and this embedder ranks by polarity, in the direction of the search query. Since the evidence against a trait rarely reached the judge, most players whose reports speak against an asked trait stayed out of the result because they had no evidence for it, not because the judge removed them. This keeps out a player whose reports are silent on the other traits of a question, but not one whose reports speak for every other trait, and these players were the hardest to keep out. A search query phrased against the trait would probably rank this evidence higher, and the evaluation did not test it.

The second failure concerns implicit evidence. A dense embedder scores how similar two texts are, whereas an implicit passage is evidence for a trait only through an inference from what a player did to what he is, and Su et al. (2025) showed that similarity fails where relevance needs such a step. Retrieval accordingly ranked an implicit passage below an explicit one of the same trait, and a relevant player whose evidence for a trait was only implicit was found clearly less often (Section 5.3.7). The fixed budget of the judge turns this lower rank into a lost player, since a passage outside a player's five best chunks or a player below the first 50 is never read. A cut-off that follows the similarities of the retriever, as methods of ranked-list truncation set one (Section 4.3.3), could read more where many chunks speak about a trait and less where few do. Where a rating field measures the trait, the ratings catch part of what retrieval loses, since players with evidence in both views were found most often and those with text alone least often (Table D.8). The ratings were made from the same labels as the gold, however, so this shows how the two views combine rather than how much a club's own ratings would add.

Given the chunks that retrieval showed it, the judge lowered the harm at the top of the ranking, and its hypothesis is the only one on the trait search that holds on both models (Section 5.3.7). Recall fell with the judge because it excludes players, but by about a fifth, whereas the harmful inclusion of the players excluded by text fell by two thirds to three quarters. Most of that loss of recall comes from the five chunks the judge reads of a player rather than from how it read them (Appendix E). The judge also read evidence against a trait less reliably than evidence for it and scored closer to zero than the labels, so even a passage against the trait that it was shown did not always remove the player.

These findings rest on the signed labels of the test collection. Labelling every report for every trait made the relevant players that the trait search missed countable, and the sign of each label made the harm of a ranking measurable and showed that retrieval follows the direction of the search query, which relevance without a sign could not have shown. The labels have a limit, however. The labelling model received each trait with a definition written for the collection, such as "takes instruction on and visibly applies it" for coachability, whereas the trait search works from the words of the question (Section 5.3.3). On the coachability questions, retrieval ranked the strongest labelled passage too low and the judge scored most of the others as saying nothing about the trait. The judge is a model of the same generation as the labelling model and should know about as much about football, so a gap it shares with the retriever lies at least partly in the gold rather than only in the embedder. The definitions that the labelling model received might cause it, or implicit labels that another reader would not give (Appendix E). How a collection derived from labels can close this gap is open.

The two other hypotheses on the trait search did not hold. The second query variant raised nDCG@10 on a few questions and lowered it a little on many, probably because it lies further from the words of the question, as Benham and Culpepper (2017) found for fused phrasings. With a loss weighted six times as heavily as a gain it risked more than it gained, so whether it stays depends on how heavily a single loss weighs (Section 5.3.7). German questions lowered nDCG@10 significantly on Luna but not on Sol. Both models search with the same embedder, so the loss arises in the steps that the model takes from the German words.

6.3 Router

The hypothesis that the router lowers the components behind it held for neither of them, and its only significant loss, the nDCG@10 that the trait search lost on Luna, came from its division rather than from its routes. It sent every fact and every trait question down its route, and its few misroutes fell on hybrid questions whose wording put a condition near the line between fact and trait, as when Sol read "a high return in front of goal" as a stored statistic. That line is the one the router has to draw from the question alone, without the data (Section 3.2.2). Behind the router the MQL sub-agent kept its success rate, although in a few runs the router added or changed a condition of the fact part (Section 5.4.4).

Most of that loss came from five questions on which the router had rewritten an idiomatic trait part in other, longer words. Luna rewrote almost every trait part and made it longer, whereas Sol kept the length of the gold part and in about a fifth of its runs its exact words. The router keeps the numbers, names and comparisons of a question as the user wrote them, but no rule tells the router to keep the words of a trait. The router should therefore keep the trait part in the user's words as well, which would probably avoid most of this loss.

The division checker judged most of these rewritten parts the same as the gold, because its rule asks whether a part looks for the same players, which is a question of meaning (Section 5.4.2). The trait search, however, works from the words of the part, so a part with the same meaning in other words can still change its result. A language model that judges the output of another model, as the division checker does (Zheng et al., 2023), thus tests a division by a weaker standard than the score of the component that receives it. That such a part still changes the result could be seen only because each component ran both behind the router and behind an oracle router that passes it the gold part of the collection, which traced each loss to the component or to the router (Section 5.1.2). The hybrid questions, which only the division checker scores, lack that comparison, so their shares of matching divisions are probably too high. The routing workflow thus answered fact and trait questions behind the router about as well as behind the oracle, whereas its combination of the two legs on hybrid questions was tested only up to the division (Section 6.5).

6.4 Model Choice per Component

Nothing in the design requires one model for every stage, since the configuration sets for each stage which model serves it (Section 4.2). By the standard of Kapoor et al. (2025), who weigh accuracy against cost, the choice depends on what a club needs from each component. The trait search ranked almost alike on both models at an eighteenth of the cost, so the smaller model suffices for it, except where scouts ask in other languages than English, since on the German questions, which stand in for them, only Luna lost significantly (Section 5.3.7). On the MQL sub-agent the larger model answered 7.8 [1.1, 14.6] percentage points more fact questions correctly, although the smaller model also answered almost all of the questions that the product's own search can already answer (Table D.10). Each extra correct answer cost about 1.2 dollars on Sol at the reference price, since Sol cost about 9.6 dollars more per 100 fact questions (Section 5.2.4). Sol is therefore the choice where correct answers matter more than cost, whereas Luna suffices where the cost decides or the questions stay close to what the product's search already offers.

For the router the choice is less clear, since its routes and its division point in different directions. Sol routed no better than Luna, so by its routes alone the smaller model suffices (Section 5.4.4). Luna's trait parts, however, drifted further from the words of the gold part, and its loss behind the router belongs to the Luna router and the Luna trait search together, since every end-to-end run used one model for both (Section 6.3). Which model the router should run on therefore stays open until two routers run in front of the same component.

6.5 Limitations

Five limits of the design bound what the system can answer and which evidence it reads. (1) The dispatch combines the two parts in one direction only, from the players that the facts admit to the trait search that ranks them, so a question whose fact part depends on a trait result cannot be answered in one run (Section 3.2.3). ER-RAG, by contrast, combines its sources inside one query whose steps can read them in either order (Xia et al., 2025). (2) The router decides once, and no later step can correct its route or its division, which lowered the nDCG@10 of the trait search on Luna. An agent that holds the trait search as one more tool could revise such a choice once a result shows it to be wrong, at the price of at least one more step for every question (Section 3.2.2). (3) The judge reads the same budget for every trait, 5 chunks of each of the first 50 players. On 70 players this budget reaches most players of every trait, whereas on a club's full database a common trait has more fitting players than the judge reads, so the fixed budget would cut off more of them. A budget that varies with the trait, set by a cut-off on the similarities of the retriever, could account for this (Section 6.2). (4) The embedder is limited to 0.6 billion parameters, since it runs on the processor of the club's server (Section 4.1). An embedder hosted outside that server, as the language models already are, could be a larger one that retrieves better. (5) A tick at an option that marks a weakness is not read as evidence against a trait (Section 3.4.3). Matching also the options that count against a trait, and reading a tick at them as evidence against the player, could close this gap.

Three comparisons that would test these choices were not run. (1) No single agent that holds every tool was set against the routing workflow (Section 5.1.3), so the choice of a routing workflow rests on the design argument of Section 3.2.2 and on what the router cost the components behind it (Section 6.3). (2) No fixed workflow with the same steps was set against the MQL sub-agent, so its freedom to choose its next step is argued rather than measured (Section 6.1). (3) The judge was tested only together with its five chunks per player, so the evaluation does not separate what the judge decides from what it is shown, which a configuration with more chunks per player would have done.

The data of the evaluation limit what it shows in four ways. (1) It ran on one database with 70 players and the 486 reports of one club, whereas the archive of a club holds tens of thousands of reports and up to about 200,000 in extreme cases (Section 4.1). This matters most on the trait route, which searches the whole archive and would meet far more fitting players per trait than here, whereas on the hybrid route the fact part often narrows the players to a group of about the size of this collection. (2) A fact condition left only a handful of the 70 players, so the hybrid questions were scored only on their route and division (Section 5.1.1). How well the ratings are read when a player's standing rests on the few players a fact part returns, and how often an error of the fact part carries into the trait part, therefore remains unmeasured. (3) All reports are in English, so an index in which reports of several languages are ranked against each other was not tested (Section 3.4.2). (4) The club's own data, such as its watchlists, tags and scouting statuses, was made for the evaluation, so it may differ from the data that clubs create. The databases of several real clubs would test the sub-agent on that variety, which one made copy cannot.

The gold of the trait search rests on labels that a language model made and on the rule of the trait search itself, and four properties of this gold limit what its scores measure. (1) The labels and both of their checks come from Claude Opus 5, and only the 254 clear disagreements were reviewed by hand (Section 5.3.3). Its family differs from that of the models under test, which lowers but does not remove the chance that they err alike (Goel et al., 2025; Kim et al., 2025a), and the same holds for the division checker. (2) The gold applies the rule by which the trait search combines its evidence, so a score measures how closely the steps of the search reach the outcomes of that rule and not whether the rule gives the list a scout wants (Section 5.3.4). (3) The rating fields were made from the same text labels (Section 5.3.2), so a rating rarely holds what the reports of its player do not say, whereas a club's own ratings can record what no

report mentions. The made fields reproduce such cases, as players rated well on a trait that their reports never mention, but by construction rather than from how ratings and text relate in a club's data. (4) The labelling model received each trait with a definition written for the collection, whereas the trait search works from the words of the question, so part of the loss of the trait search lies in how a trait is defined (Section 6.2).

Most hypotheses were not supported, and with 41 fact and 59 trait questions the tests could detect only effects of some size. A hypothesis that was not supported therefore shows how large its effect could at most be, not that the two configurations are equal, since a difference that is not significant is an absence of evidence and not evidence of equality (Altman and Bland, 1995). The bounds of Chapter 5 thus state what the results rule out, such as a loss of more than 5.4 percentage points for the MQL sub-agent behind the router on Luna. Questions on the same trait are not fully independent, so the intervals of the trait search are somewhat too narrow (Section 5.1.3).

7 Conclusion and Future Work

This thesis built a chat system that answers a scout's questions from a club's structured data and from the text of its reports, and it evaluated the system on a test collection built for that purpose. The system is a routing workflow, in which a router divides a question into facts and traits, an MQL sub-agent queries the database for the facts and a trait search ranks the players by what their reports and ratings say about the traits. Behind the router both components scored about as well as behind an oracle router that always divides a question as the gold does, the judge of the trait search kept players whose reports speak against a trait away from the top of the ranking, and most of what the trait search lost, it lost in the retrieval of text passages.

7.1 Research Questions and Contributions

The first of the four contributions, the test collection of 154 questions, answers RQ1, and its core lies in how it scores the trait search. For scoring the trait search, a language model labelled each of the 486 real reports of one club for every one of the 33 traits, as silent or with a direction and a strength, without seeing any question. In this way the collection combines entity retrieval, in which players are ranked by the reports about them, with the direction of the evidence, for or against a trait. The relevant players of a question follow from these labels by a stated rule, so no pool is needed, and every relevant player that the trait search misses can be counted and traced to the step that lost him. The direction also keeps evidence against a trait apart from silence about it, and a player whose reports speak against an asked trait therefore counts as harm. Harmful inclusion, the harmful share and discounted harm@10 measure this harm beside recall, precision and nDCG@10, and they are adapted from ExcluIR and the TREC Health Misinformation track (Zhang et al., 2025a; Clarke et al., 2021). With these measures and with the situations marked for each player, such as evidence that is only implicit, the results show where the trait search loses relevant players and where removed players still reach its result (Section 6.2).

For the MQL sub-agent and the router, the collection is built to trace each failure to where it arose. The fact questions start from the product's own player search and extend it, and they run on a messy copy of a production database whose stored values they name in other words. Each gold query follows the path that the sub-agent is built to take, and the steps a run takes beyond that path therefore measure what the sub-agent's freedom to choose its steps costs. Each wrong run also carries an outcome and the cause of its first error, and the two show how its answer ended and where the run went wrong (Section 6.1). For the router, every question carries its gold route and division, and each component runs both behind the router and behind an oracle router, so that each loss can be traced to the component or to the router (Section 6.3). In addition, every question is asked in German with the same gold, so the MQL sub-agent and the trait search are tested in a language other than that of their data. The gold limits the collection through labels made by a model, the trait search's own rule, ratings made from the same labels and the definitions behind the labels (Sections 6.2 and 6.5). The data limit it through one club with 70 players, reports only in English, club data made for the evaluation and hybrid questions scored only on their route and division (Section 6.5).

The answer to RQ2 starts from a case that the earlier systems combining structured and unstructured data did not need to handle, since in a club's data the rating fields record part of what the text of the reports says about the same quality. Those systems decided for each

question or each part of it which source holds the answer (Liu et al., 2024b; Xia et al., 2025; Bai et al., 2026), whereas the system built in this thesis divides a question by the information it needs, a fact that the data states or a trait that has to be judged from the evidence. The system then combines the two kinds of data on two levels, (1) the router and the dispatch join the MQL sub-agent with the trait search, and (2) inside the trait search the text and the rating fields are two views of the same trait. The second contribution is the rule by which the trait search combines the reports and the rating fields of a football club as two views of one trait (Section 3.4.4). In this rule each view can speak for or against the trait, a second view only adds to the first, and evidence against the trait in one view is cancelled only by evidence for it in the other.

Of the components of this routing workflow, only the MQL sub-agent is an agent, since each query it writes usually depends on the result of the one before (Section 3.3.1). It takes over steps that added accuracy inside the fixed workflows that beat an agent, such as reading the stored values before it answers, and it keeps the freedom to choose its next step. This design of the component that writes the database queries is the third contribution, and it is a proposal for others to test, since no fixed workflow with the same steps was compared with it (Section 6.1). The trait search, by contrast, is a fixed procedure that ranks players by the reports about them and is therefore a case of entity retrieval. Its judge reads for every chunk it receives whether the chunk speaks for or against the trait. Earlier work recorded this direction for documents or ranked documents by it (Ounis et al., 2006, 2008), or it averaged the direction into the relevance grade of an entity (Ganesan and Zhai, 2012). The trait search instead removes a player on evidence against an asked trait, and this use of the direction in entity retrieval is the fourth contribution.

The routing workflow cost its components little, since both scored about as well behind the router as behind the oracle router (Section 6.3). Only one loss was significant, that of the trait search on Luna, and it came from the router rewording the trait part. Of the components themselves, the MQL sub-agent answered 89.8 and 82.0 % of the fact questions correctly on Sol and Luna, and its wrong answers came from how it read a question and the data rather than from the queries it wrote (Section 6.1). Its freedom cost about 2 steps per run beyond the path of the gold query, and German questions were answered at least as often as English ones. The trait search ranked the relevant players at an nDCG@10 of 0.78 against 0.22 for a random ranking, and its judge halved the harm among the first ten places (Section 6.2). About 60 % of the relevant players it missed were lost in retrieval before the judge read their chunks, and German questions lowered its nDCG@10 significantly only on Luna. The combination of the two legs on hybrid questions, however, was tested only up to the division (Section 6.5).

For RQ3, none of the ways of composing the agent's context changed its success measurably at the size of this database, so cost and time decide between them. Table 7.1 sets each recommendation of Table 2.4 against how the design applies it and what the runs show. Only R5 against R8 was tested as a hypothesis, and neither showed its predicted effect. The runs also bear on two recommendations for the agent, since examples similar to the question did not harm while a closely fitting detail was copied (R3), and about half of the Luna runs with a rejected query still ended correct (R6).

7.2 Recommendations

Two recommendations concern the trait search and the worked examples. (1) The trait search should keep the judge, since it lowered the harm at the top on both models and recall fell by far less than the harmful inclusion of the players excluded by text (Section 6.2). (2) With Luna the worked examples should stay, since without them it wrote more than twice as many queries that the database rejected. With Sol they did not matter, since their only gain came from a rule that the notes of the collections should state (Section 6.1). The three ways of giving them trade against each other. Examples tied to a collection show which copy of a fact to trust and save steps, but a closely fitting detail can be copied. Without examples nothing can be copied

Table 7.1: The recommendations on context engineering of Table 2.4, how the design applies them and what the runs show.

	Design	What the runs show
R1	the question stands before the results in the synthesis prompt	not tested, since the synthesis call is not scored and position mattered mainly on the models of 2023 (Section 2.1)
R2	samples, rows and a display budget bound what an observation shows	the longer preloaded prompts cost no accuracy at this size (Section 6.1)
R3	examples chosen by similarity to the question, as one configuration	similar examples did not harm, a closely fitting detail was copied (Section 6.1)
R4	retrieval selects the chunks that the judge reads	the judge read chunks against the trait and implicit ones less often than chunks for it and explicit ones (Section 6.2)
R5	schemas read on demand, handles instead of ids	no difference in success against R8, so cost and time decide (Section 6.1)
R6	every step stays in the context, failed ones included	about half of the Luna runs with a rejected query still ended correct (Section 6.1)
R7	the sub-agent returns only its answer and the handles of its results	not tested, since a comparison would need a second architecture
R8	every schema preloaded, as one configuration	no gain at about 5 % of the context window (Section 6.1)
R9	the session memory keeps the steps as they were taken	not tested, since every question ran on its own

wrongly and nothing has to be maintained, but what only an example carried is lost. Retrieved examples cost fewer tokens and are known before any schema is read, but half of them come from collections a run never reads.

Four further recommendations concern the context and the models. (3) Whether the schemas are read on demand or preloaded made no difference at the size tested, and reading on demand would probably matter only once a chat session fills the context (Section 6.1). (4) The trait search can run on Luna except where scouts ask in other languages than English, and the MQL sub-agent on Sol where correct answers matter more than cost, whereas the router should keep the trait part in the user's words on either model (Sections 6.3 and 6.4). (5) The step limit is a choice between letting the user wait and asking him back, since a long run was not more often wrong (Section 6.1). (6) Whether the second query variant should stay depends on how heavily a single loss weighs (Section 6.2).

7.3 Future Work

Future work falls into three groups. (1) Retrieval decides most of what the trait search can find, so it comes first. A budget that varies with the trait, set by a cut-off on the similarities of the retriever as methods of ranked-list truncation set one, could read more players and chunks where many chunks speak about a trait, which matters most for clubs with many reports. A search query phrased against the trait could bring the evidence against a trait to the judge, and an embedder hosted outside the club's server could be a larger one that retrieves better (Sections 6.2 and 6.5). (2) Since every end-to-end run used one model for the router and for the component behind it, a Sol router and a Luna router in front of the same component could show which model the router should run on. More players would let the hybrid questions be scored end to end, and a combination in both directions, as ER-RAG builds one, would answer questions whose fact part depends on a trait (Sections 6.3 to 6.5). (3) For the test collection, how to close the gap between the definitions behind the labels and the words of the questions is

open, and the databases of several real clubs and reports in several languages would test the system on the variety that one made copy lacks (Sections 6.2 and 6.5).

Bibliography

Ken Aizawa. Writing effective tools for agents — with agents. Anthropic Engineering blog, 2025. URL https://www.anthropic.com/engineering/writing-tools-for-agents. Accessed 23 August 2026.

Douglas G. Altman and J. Martin Bland. Statistics notes: Absence of evidence is not evidence of absence. *BMJ*, 311(7003):485, 1995. doi:10.1136/bmj.311.7003.485.

Artificial Analysis. GPT-6 Sol and Luna push the cost efficiency frontier. Artificial Analysis article, 2026. URL https://artificialanalysis.ai/articles/gpt-6-sol-and-luna-push-the-cost-efficiency-frontier. Accessed 7 October 2026.

Haoyue Bai, Haoyu Wang, Shengyu Chen, Zhengzhang Chen, Lu-An Tang, Wei Cheng, Yanjie Fu, and Haifeng Chen. Learning to route: A rule-driven agent framework for hybrid-source retrieval-augmented generation. In *Proceedings of the ACM Web Conference 2026*. Association for Computing Machinery, 2026. URL https://arxiv.org/abs/2510.02388v2.

Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. Formal models for expert finding in enterprise corpora. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50. Association for Computing Machinery, 2006. doi:10.1145/1148170.1148181.

Krisztian Balog, Yi Fang, Maarten de Rijke, Pavel Serdyukov, and Luo Si. Expertise retrieval. *Foundations and Trends in Information Retrieval*, 6(2–3):127–256, 2012. doi:10.1561/1500000024.

Rodger Benham and J. Shane Culpepper. Risk-reward trade-offs in rank fusion. In *Proceedings of the 22nd Australasian Document Computing Symposium*, pages 1–8. Association for Computing Machinery, 2017. doi:10.1145/3166072.3166084.

Roger L. Berger. Multiparameter hypothesis testing and acceptance sampling. *Technometrics*, 24(4):295–300, 1982. doi:10.1080/00401706.1982.10487790.

Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16(2):101–133, 2001. doi:10.1214/ss/1009213286.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Chien Dinh Bui, Ha Hoang Nguyen, Tao Quoc Ngo, Huyen Khanh Vu-Thi, Chien Hong Nguyen, Duc Van Nguyen, and Son Tung Ngo. A systematic survey of LLM-based text-to-SQL: Methodologies, security vulnerabilities, and future challenges. *PeerJ Computer Science*, 12: e3773, 2026. doi:10.7717/peerj-cs.3773.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. Dr.Spider: A diagnostic evaluation

benchmark towards text-to-SQL robustness. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Wc5bmZZU9cy.

Abhay Chaturvedi, Shreya Bhattacharya, Rashmika Gopalkrishnan, and Peter van der Putten. DTOC: Dynamic tool output compression for adaptive context management in AI agents. In *29th International Conference on Discovery Science (DS 2026)*, 2026. doi:10.48550/arXiv.2609.26121. To appear in Springer LNCS.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-Embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335. Association for Computational Linguistics, 2024a. doi:10.18653/v1/2024.findings-acl.137.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi:10.48550/arXiv.2107.03374.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177. Association for Computational Linguistics, 2024b. doi:10.18653/v1/2024.emnlp-main.845.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024c. URL https://openreview.net/forum?id=KuPixIqPiq.

Charles L. A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–666. Association for Computing Machinery, 2008. doi:10.1145/1390334.1390446.

Charles L. A. Clarke, Alexandra Vtyurina, and Mark D. Smucker. Offline evaluation without gain. In *Proceedings of the 2020 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 185–192. Association for Computing Machinery, 2020. doi:10.1145/3409256.3409816.

Charles L. A. Clarke, Maria Maistro, and Mark D. Smucker. Overview of the TREC 2021 health misinformation track. In *Proceedings of the Thirtieth Text REtrieval Conference (TREC 2021)*, NIST Special Publication 500-335. National Institute of Standards and Technology, 2021. URL https://trec.nist.gov/pubs/trec30/papers/Overview-HM.pdf.

Jacob Cohen. Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological Bulletin*, 70(4):213–220, 1968. doi:10.1037/h0026256.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759. Association for Computing Machinery, 2009. doi:10.1145/1571941.1572114.

Nick Craswell, Arjen P. de Vries, and Ian Soboroff. Overview of the TREC-2005 enterprise track. In *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, NIST Special Publication 500-266. National Institute of Standards and Technology, 2005. URL https://trec.nist.gov/pubs/trec14/papers/ENTERPRISE.OVERVIEW.pdf.

Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. The power of noise: Redefining retrieval for RAG systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–729. Association for Computing Machinery, 2024. doi:10.1145/3626772.3657834.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. ReFoRCE: A text-to-SQL agent with self-refinement, consensus enforcement, and column exploration. *arXiv preprint arXiv:2502.00675v5*, 2025. doi:10.48550/arXiv.2502.00675.

B. Taner Dinçer, Craig Macdonald, and Iadh Ounis. Hypothesis testing for the risk-sensitive evaluation of retrieval systems. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 23–32. Association for Computing Machinery, 2014. doi:10.1145/2600428.2609625.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, et al. MMTEB: Massive multilingual text embedding benchmark. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zl3pfz4VCV.

Epoch AI. Capabilities & benchmarking. Epoch AI benchmark database, 2026. URL https://epoch.ai/benchmarks. Accessed 20 September 2026.

Guglielmo Faggioli, Laura Dietz, Charles L. A. Clarke, Gianluca Demartini, Matthias Hagen, Claudia Hauff, Noriko Kando, Evangelos Kanoulas, Martin Potthast, Benno Stein, and Henning Wachsmuth. Perspectives on large language models for relevance judgment. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 39–50. Association for Computing Machinery, 2023. doi:10.1145/3578337.3605136.

Kavita Ganesan and ChengXiang Zhai. Opinion-based entity ranking. *Information Retrieval*, 15 (2):116–150, 2012. doi:10.1007/s10791-011-9174-8.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024. doi:10.14778/3641204.3641221.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777. Association for Computational Linguistics, 2023a. doi:10.18653/v1/2023.acl-long.99.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023b. doi:10.48550/arXiv.2312.10997.

GeeVec. GeeVec-Embeddings-1.0-Lite. Hugging Face model card, 2026. URL https://huggingface.co/geevec-ai/geevec-embeddings-1.0-lite. Accessed 24 September 2026.

Andrew Gelman and Phillip N. Price. All maps of parameter estimates are misleading. *Statistics in Medicine*, 18(23):3221–3234, 1999. doi:10.1002/(SICI)1097-0258(19991215)18:23<3221::AID-SIM312>3.0.CO;2-M.

Lukas Gienapp, Maik Fröbe, Matthias Hagen, and Martin Potthast. The impact of negative relevance judgments on NDCG. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pages 2037–2040. Association for Computing Machinery, 2020. doi:10.1145/3340531.3412123.

Shashwat Goel, Joschka Strüber, Ilze Amanda Auzina, Karuna K. Chandra, Ponnurangam Kumaraguru, Douwe Kiela, Ameya Prabhu, Matthias Bethge, and Jonas Geiping. Great models think alike and this undermines AI oversight. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 19621–19678. PMLR, 2025. URL https://proceedings.mlr.press/v267/goel25b.html.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi:10.1038/s41586-025-09422-z.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, 2025. URL https://research.trychroma.com/context-rot. Accessed 17 September 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024a. URL https://openreview.net/forum?id=kIoBbc76Sy.

Cheng-Yu Hsieh, Yung-Sung Chuang, Chun-Liang Li, Zifeng Wang, Long T. Le, Abhishek Kumar, James Glass, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Found in the middle: Calibrating positional attention bias improves long context utilization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14982–14995. Association for Computational Linguistics, 2024b. doi:10.18653/v1/2024.findings-acl.890.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024b. doi:10.48550/arXiv.2402.02716.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, et al. BIRD-INTERACT: Re-imagining text-to-SQL evaluation via lens of dynamic interactions. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=nHrYBGujps.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002. doi:10.1145/582415.582418.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.naacl-long.389.

Yichao Ji. Context engineering for AI agents: Lessons from building Manus. Manus blog, 2025. URL https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus. Accessed 23 August 2026.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary

visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, 2017. doi:10.1109/CVPR.2017.215.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.emnlp-main.550.

Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–48. Association for Computing Machinery, 2020. doi:10.1145/3397271.3401075.

Elliot Kim, Avi Garg, Kenny Peng, and Nikhil Garg. Correlated errors in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 30038–30066. PMLR, 2025a. URL https://proceedings.mlr.press/v267/kim25e.html.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025b. doi:10.48550/arXiv.2512.08296.

Anton Korikov, George Saad, Ethan Baron, Mustafa Khan, Manav Shah, and Scott Sanner. Multi-aspect reviewed-item retrieval via LLM query decomposition and aspect fusion. In *Proceedings of the Workshop on Information Retrieval's Role in RAG Systems (IR-RAG) at SIGIR 2024*, volume 3784 of *CEUR Workshop Proceedings*, pages 23–33, 2024. URL https://ceur-ws.org/Vol-3784/paper3.pdf.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019. doi:10.1162/tacl_a_00276.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, pages 28691–28735, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/46c10f6c8ea5aa6f267bcdabcb123f97-Abstract-Conference.html.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: The impact of input length on the reasoning performance of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.acl-long.818.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In

Advances in Neural Information Processing Systems, volume 33, pages 9459–9474. Curran Associates, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, 2023. doi:10.52202/075280-1835.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.findings-emnlp.438.

Geling Liu, Yunzhi Tan, Ruichao Zhong, Yuanzhen Xie, Lingchen Zhao, Qian Wang, Bo Hu, and Zang Li. Solid-SQL: Enhanced schema-linking based in-context learning for robust text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9793–9803. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.coling-main.654/.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi:10.1162/tacl_a_00638.

Shicheng Liu, Jialiang Xu, Wesley Tjangnaka, Sina Semnani, Chen Yu, and Monica Lam. SUQL: Conversational search over structured and unstructured data with large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4535–4555. Association for Computational Linguistics, 2024b. doi:10.18653/v1/2024.findings-naacl.283.

Jinwei Lu, Jiawei Lu, Chen Zhang, Zhiqian Qin, Haodi Zhang, Yuanfeng Song, and Raymond Chi-Wing Wong. Bridging the gap: Enabling natural language queries for NoSQL databases through text-to-NoSQL translation. *arXiv preprint arXiv:2502.11201v3*, 2026. doi:10.48550/arXiv.2502.11201.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021. doi:10.1162/tacl_a_00369.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? Text-to-SQL in the age of well-reasoned language models. In *NeurIPS 2024 Workshop on Table Representation Learning (TRL)*, 2024. URL https://openreview.net/forum?id=fglyh5pa7d. Non-archival workshop paper.

Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi:10.1109/TPAMI.2018.2889473.

Nestor Maslej, Loredana Fattorini, Raymond Perrault, et al. The AI Index 2025 annual report. Technical report, AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, 2025. URL https://hai.stanford.edu/ai-index/2025-ai-index-report. Accessed 19 October 2026.

Microsoft. harrier-oss-v1-0.6b. Hugging Face model card, 2026a. URL https://huggingface.co/microsoft/harrier-oss-v1-0.6b. Accessed 23 September 2026.

Microsoft. harrier-oss-v1: Open-source multilingual text embeddings. Microsoft Foundry Labs project page, 2026b. URL https://labs.ai.azure.com/innovations/harrier-oss-v1/. Accessed 24 September 2026.

Microsoft. Prompt caching with Azure OpenAI in Microsoft Foundry Models. Microsoft Learn documentation, 2026c. URL https://learn.microsoft.com/en-us/azure/ai-foundry/openai/how-to/prompt-caching. Accessed 18 October 2026.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 44554–44570. PMLR, 2025. URL https://proceedings.mlr.press/v267/modarressi25a.html.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *International Conference on Learning Representations*, pages 45544–45613, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/70cf215430492f7d34830a24e744b3f1-Abstract-Conference.html.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.findings-emnlp.63.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. doi:10.48550/arXiv.2303.08774.

OpenAI. Introducing Structured Outputs in the API. OpenAI blog, 2024. URL https://openai.com/index/introducing-structured-outputs-in-the-api/. Accessed 15 October 2026.

OpenAI. Models. OpenAI API documentation, 2026a. URL https://platform.openai.com/docs/models. Accessed 17 September 2026.

OpenAI. API pricing. OpenAI API price list, 2026b. URL https://openai.com/api/pricing/. Accessed 18 October 2026.

Iadh Ounis, Maarten de Rijke, Craig Macdonald, Gilad Mishne, and Ian Soboroff. Overview of the TREC-2006 blog track. In *Proceedings of the Fifteenth Text REtrieval Conference (TREC 2006)*, NIST Special Publication 500-272. National Institute of Standards and Technology, 2006. URL https://trec.nist.gov/pubs/trec15/papers/BLOG06.OVERVIEW.pdf.

Iadh Ounis, Craig Macdonald, and Ian Soboroff. Overview of the TREC-2008 blog track. In *Proceedings of the Seventeenth Text REtrieval Conference (TREC 2008)*, NIST Special Publication 500-277. National Institute of Standards and Technology, 2008. URL https://trec.nist.gov/pubs/trec17/papers/BLOG.OVERVIEW08.pdf.

Arif Görkem Özer, Recep Firat Cekinel, Ismail Hakki Toroslu, and Pinar Karagoz. DocSpider: A dataset of cross-domain natural language querying for MongoDB. *Natural Language Processing*, 31(6):1367–1398, 2025. doi:10.1017/nlp.2024.63.

Luca Pappalardo, Paolo Cintia, Paolo Ferragina, Emanuele Massucco, Dino Pedreschi, and Fosca Giannotti. PlayeRank: Data-driven performance evaluation and player ranking in soccer via a machine learning approach. *ACM Transactions on Intelligent Systems and Technology*, 10(5):1–27, 2019. doi:10.1145/3343172.

Virgil Pavlu, Shahzad Rajput, Peter B. Golbus, and Javed A. Aslam. IR system evaluation using nugget-based test collections. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, pages 393–402. Association for Computing Machinery, 2012. doi:10.1145/2124295.2124343.

Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. SemEval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 27–35. Association for Computational Linguistics, 2014. doi:10.3115/v1/S14-2004.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36, pages 36339–36348. Curran Associates, 2023. doi:10.52202/075280-1577.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Özcan, and Sercan Ö. Arık. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *International Conference on Learning Representations*, pages 60385–60415, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/974ff7b5bf08dbf9400b5d599a39c77f-Abstract-Conference.html.

Zhiqian Qin, Yuanfeng Song, Jinwei Lu, Yuanwei Song, Shuaimin Li, and Chen Jason Zhang. MultiTEND: A multilingual benchmark for natural language to NoSQL query translation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24632–24657. Association for Computational Linguistics, 2025. doi:10.18653/v1/2025.findings-acl.1265.

Zackary Rackauckas. RAG-Fusion: A new take on retrieval-augmented generation. *International Journal on Natural Language Computing*, 13(1):37–47, 2024. doi:10.5121/ijnlc.2024.13103.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Prithvi Rajasekaran, Ethan Dixon, Carly Ryan, and Jeremy Hadfield. Effective context engineering for AI agents. Anthropic Engineering blog, 2025. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents. Accessed 19 October 2026.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-SQL capabilities of large language models. *arXiv preprint arXiv:2204.00498*, 2022. doi:10.48550/arXiv.2204.00498.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992. Association for Computational Linguistics, 2019. doi:10.18653/v1/D19-1410.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi:10.1561/1500000019.

Mark Sanderson. Test collection based evaluation of information retrieval systems. *Foundations and Trends in Information Retrieval*, 4(4):247–375, 2010. doi:10.1561/1500000009.

Erik Schluntz and Barry Zhang. Building effective agents. Anthropic Engineering blog, 2024. URL https://www.anthropic.com/engineering/building-effective-agents. Accessed 23 August 2026.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, et al. The prompt report: A systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608v6*, 2025. doi:10.48550/arXiv.2406.06608.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-SQL tasks. *ACM Computing Surveys*, 58(2):1–37, 2025. doi:10.1145/3737873.

Sithursan Sivasubramaniam, Cedric Osei-Akoto, Yi Zhang, Kurt Stockinger, and Jonathan Fürst. SM3-Text-to-Query: Synthetic multi-model medical text-to-query benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pages 88627–88663. Curran Associates, 2024. doi:10.52202/079017-2812.

Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, pages 623–632. Association for Computing Machinery, 2007. doi:10.1145/1321440.1321528.

Ian Soboroff, Arjen P. de Vries, and Nick Craswell. Overview of the TREC 2006 enterprise track. In *Proceedings of the Fifteenth Text REtrieval Conference (TREC 2006)*, NIST Special Publication 500-272. National Institute of Standards and Technology, 2006. URL https://trec.nist.gov/pubs/trec15/papers/ENT06.OVERVIEW.pdf.

Sports Reference LLC. Scouting reports explained. FBref documentation, 2026. URL https://fbref.com/en/about/scouting-reports-explained. Accessed 5 October 2026 as the archived snapshot of 8 January 2026, http://web.archive.org/web/20260108232949/https://fbref.com/en/about/scouting-reports-explained.

S. S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946. doi:10.1126/science.103.2684.677.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan Ö. Arık, Danqi Chen, and Tao Yu. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *International Conference on Learning Representations*, pages 48941–48991, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7a0f8055c838df8e62329a76c7c6403d-Abstract-Conference.html.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? Investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937. Association for Computational Linguistics, 2023. doi:10.18653/v1/2023.emnlp-main.923.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon agent via context folding. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 116894–116910. PMLR, 2026. URL https://proceedings.mlr.press/v306/sun26x.html.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755v3*, 2024. doi:10.48550/arXiv.2405.16755.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html.

Paul Thomas, Seth Spielman, Nick Craswell, and Bhaskar Mitra. Large language models can accurately predict searcher preferences. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1930–1940. Association for Computing Machinery, 2024. doi:10.1145/3626772.3657707.

Julián Urbano, Harlley Lima, and Alan Hanjalic. Statistical significance testing in information retrieval: An empirical analysis of type I, type II and type III errors. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 505–514. Association for Computing Machinery, 2019. doi:10.1145/3331184.3331259.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Ellen M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing & Management*, 36(5):697–716, 2000. doi:10.1016/S0306-4573(00)00010-8.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557. Association for Computational Linguistics, 2025a. URL https://aclanthology.org/2025.coling-main.36/.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual E5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024. doi:10.48550/arXiv.2402.05672.

Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Efficient agents: Building effective agents while reducing cost. *arXiv preprint arXiv:2508.02694*, 2025b. doi:10.48550/arXiv.2508.02694.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, 2022. doi:10.52202/068431-1800.

Orion Weller, Dawn Lawrie, and Benjamin Van Durme. NevIR: Negation in neural information retrieval. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2274–2287. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.eacl-long.139.

Jonny Whitmore. Slice it up: Introducing Opta player radars. The Analyst (Opta Analyst, Stats Perform), web article, 2023. URL

https://theanalyst.com/articles/introducing-opta-radars-compare-players/. Accessed 5 October 2026.

Bernard Widrow, István Kollár, and Ming-Chang Liu. Statistical theory of quantization. *IEEE Transactions on Instrumentation and Measurement*, 45(2):353–361, 1996. doi:10.1109/19.492748.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023. doi:10.48550/arXiv.2307.09702.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. doi:10.1080/01621459.1927.10502953.

Yikuan Xia, Jiazun Chen, Yirui Zhan, Suifeng Zhao, Weipeng Jiang, Chaorui Zhang, Wei Han, Bo Bai, and Jun Gao. ER-RAG: Enhance RAG with ER-based unified modeling of heterogeneous data sources. *arXiv preprint arXiv:2504.06271*, 2025. doi:10.48550/arXiv.2504.06271.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics, 2018. doi:10.18653/v1/D18-1425.

ChengXiang Zhai, William W. Cohen, and John Lafferty. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 10–17. Association for Computing Machinery, 2003. doi:10.1145/860435.860440.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, pages 86069–86100, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Wenhao Zhang, Mengqi Zhang, Shiguang Wu, Jiahuan Pei, Zhaochun Ren, Maarten de Rijke, Zhumin Chen, and Pengjie Ren. ExcluIR: Exclusionary neural information retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 13295–13303. Association for the Advancement of Artificial Intelligence, 2025a. doi:10.1609/aaai.v39i12.33451.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b. doi:10.48550/arXiv.2506.05176.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, 2023. doi:10.52202/075280-2020.

Justin Zobel. How reliable are the results of large-scale information retrieval experiments? In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 307–314. Association for Computing Machinery, 1998. doi:10.1145/290941.291014.

A Model Capability Reference

Chapter 2 reports results that were measured on different models. Table A.1 gives, for the models behind the results that the chapter sets against each other, the independent measurements of Epoch AI (2026) so that a difference between two results can be read against the difference between the models behind them. The tier of a model is assigned by generation: before GPT-4, the GPT-4 class of 2023 and 2024, the reasoning models of 2024 and 2025 trained by reinforcement learning (Section 2.1), and the models of 2026. Models without a measurement in the source are placed by era and size and marked.

Table A.1: Capability of the models behind the results that Chapter 2 sets against each other, from the benchmark data of Epoch AI (2026) retrieved on 20 September 2026 (CC BY 4.0). ECI is the Epoch Capabilities Index, a composite the source fits across its benchmarks, used here only to order the models. GPQA Diamond and the mock AIME 2024/25 are the source's own runs; MMLU and Aider Polyglot, a code-editing benchmark, are as reported by the model's developer or the benchmark's maintainer. For reasoning models the medium or high effort setting is given. A dash means no measurement in the source.

Model (version)	Date	Tier	ECI	GPQA D.	AIME	MMLU	Aider
GPT-3.5 Turbo (Jan 2024)	2024-01	1	115.6	27.2	2.2	67.3	–
Gemini 1.0 Pro	2023-12	1	117.0	34.0	1.1	70.0	–
Llama 3 8B	2024-04	1	–	26.1	1.9	68.8	–
Code models before 2024[e]	2021–23	1	–	–	–	–	–
GPT-4 (Mar 2023)	2023-03	2	125.9	35.7	0.6	86.4	–
GPT-4 Turbo (Apr 2024)	2024-04	2	127.3	46.6	6.7	81.3	–
GPT-4o mini	2024-07	2	126.6	37.7	6.9	81.8	–
Mistral Large 2 (Nov 2024)[f]	2024-11	2	128.5	51.3	7.8	–	–
GPT-4o (Aug 2024)	2024-08	2	128.8	49.2	6.4	84.3	23.1
Llama 3 70B	2024-04	2	–	40.6	4.3	79.3	–
Llama 3.1 405B	2024-07	2	–	50.9	9.7	84.5	–
Qwen2.5-72B	2024-09	2	129.0	49.1	8.1	85.3	–
Gemini 1.5 Pro (Sep 2024)	2024-09	2	131.7	57.2	23.1	86.9	–
DeepSeek-V3	2024-12	2	132.4	56.5	15.8	87.2	48.4
Claude 3.5 Sonnet (Oct 2024)	2024-10	2	133.6	55.3	8.5	87.3	51.6
o1-preview	2024-09	3	134.8	50.3	31.1	–	–
o3-mini (high)	2025-01	3	140.4	77.0	76.9	–	60.4
o4-mini (high)	2025-04	3	145.7	79.6	81.7	–	72.0
o3 (medium)	2025-04	3	146.9	80.8	84.4	–	76.9
DeepSeek-V4-Flash (Apr 2026)	2026-04	4	146.1	–	–	–	–
DeepSeek-V4-Flash (Jul 2026)	2026-07	4	154.5	91.0	94.4	–	–
GPT-6 Luna	2026	4	–	–	–	–	–
GPT-6 Sol	2026	4	–	–	–	–	–
GPT-6 Astra	2026-09	4	166.5	–	–	–	–

Tiers: 1 before GPT-4, 2 GPT-4 class, 3 reasoning models of 2024 and 2025, 4 models of 2026. [e] Codex, CodeLlama 70B, DeepSeek Coder 33B, Mistral 7B and Vicuna-33B, placed by era and size, no measurement in the source. GPT-6 Luna and GPT-6 Sol have no published ECI yet (Epoch AI, 4 October 2026). [f] CHASE-SQL names Mistral Large without a version, so the row gives the latest version at the time of its conference paper.

B Prompts

The prompts of the system are not reprinted here, since they are long and assembled when a stage runs. The prompt of the MQL sub-agent, for instance, combines its rules with the notes and schemas of the collections it reads and with the worked examples of its configuration, so a reprint would need one version per configuration. The code repository holds every part. Every text written for the model of a stage, the descriptions of its output schema included, lies in the file `prompts.py` of that stage, in the folders of Figure 4.1. The file `llm_schema.py` of the stage holds the output schema and attaches these descriptions to it. The rules, collection notes and worked examples of the MQL sub-agent lie in `backend/app/mql/prompts.py`. The prompt of the division checker lies in `eval/scripts/5_run/judge_prompts.py` and is given in full below, since it is a fixed text, the same in every run, and belongs to the evaluation rather than to the system: its verdicts alone decide the share of matching divisions (Section 5.4.2).

The rule for a list. Section 5.2.4 refers to rule 4 of the MQL sub-agent's prompt, whose natural ranking field for players is the market value in every worked example that sorts them.

```
4. Match the answer form to the question form. "How many ..." → a count is the
   answer. "Which / who / list / show ..." of ANY entity (players, transfers, clubs,
   competitions, ...) → probe, then attach its handle. Order the set with `sort` on
   the collection's natural ranking field so the entities come back ranked.
   For a specific count ("top 10", "five ...") ALSO set `limit` to that number so the
   set is exactly N; for an open-ended "list all", omit `limit` (full set, the UI
   pages it).
```

The prompt of the division checker. The division checker of Section 5.4.2 runs on Claude Opus 5 with the following system prompt.

```
You check how a query router divided a football-scouting question.

The router divides a question into two parts. The fact part holds the conditions a database
   filter can answer: position, age, height, foot, club, league, nationality, market value,
   contract, injuries, watchlists, tags, scouting statuses and the like. The trait part holds
   the qualities a scout describes in reports: playing style, skills, physical and mental traits.
   A part can be empty.

You get one message per question. It holds the question, the gold division, which is correct by
   definition, and the router's division:

QUESTION: <the question>
GOLD fact part: <text, or (empty)>
ROUTER fact part: <text, or (empty)>
GOLD trait part: <text, or (empty)>
ROUTER trait part: <text, or (empty)>

For each part, decide whether the router's part asks for the same as the gold part. The words
   may differ. A part is the same when a search with the router's part looks for exactly the
   players a search with the gold part looks for. The rules decide the cases where that is not
   plain.

Rules:
1. Other words or another name for the same condition are the same, also in another language.
2. A player profile, a type of player such as a role, names a position and the qualities that go
   with it. The gold fact part holds the position and the gold trait part the profile. The
   router's trait part is the same if it keeps the profile or describes qualities that fit it;
   they need not be the qualities a scout would list first. The profile in the trait part is not
   a condition of the fact part.
```

```
3. A missing condition makes a part not the same.
4. An added condition makes a part not the same.
5. A number the question states that has another value or another comparator makes a part not
   the same.
6. A number the question states that became a word makes a part not the same.
7. A quality that became narrower or wider makes a part not the same.
8. A part that contains a condition belonging to the other part is not the same, however that
   condition is worded.

Judge only what is written. Do not reward length or detail, and do not guess what the router
   meant. The question and the parts may be in English or German.

Examples, invented for this prompt. Each names the rules it shows, then gives a message and the
   exact answer it gets.

Example 1, rule 1. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
ROUTER fact part: strikers who are 1.88 m or taller and younger than 22
GOLD trait part: good at shielding the ball
ROUTER trait part: protects the ball well
Answer: {"fact_part_same": true, "trait_part_same": true, "reason": null}

Example 2, rule 1. Message:
QUESTION: Which left backs at Feyenoord are under 22?
GOLD fact part: left backs at Feyenoord under 22
ROUTER fact part: left backs at Feyenoord Rotterdam younger than 22
GOLD trait part: (empty)
ROUTER trait part: (empty)
Answer: {"fact_part_same": true, "trait_part_same": true, "reason": null}

Example 3, rule 2. Message:
QUESTION: Find me a box-to-box midfielder.
GOLD fact part: central midfielders
ROUTER fact part: central midfielders
GOLD trait part: a box-to-box midfielder
ROUTER trait part: covers a lot of ground, gets into the box and wins the ball back
Answer: {"fact_part_same": true, "trait_part_same": true, "reason": null}

Example 4, rules 3 and 8. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
ROUTER fact part: strikers of at least 1.88 m
GOLD trait part: good at shielding the ball
ROUTER trait part: good at shielding the ball and under 22
Answer: {"fact_part_same": false, "trait_part_same": false, "reason": "'under 22' moved from the
   fact part to the trait part."}

Example 5, rule 4. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
ROUTER fact part: strikers of at least 1.88 m under 22
GOLD trait part: good at shielding the ball
ROUTER trait part: good at shielding the ball and a good crosser
Answer: {"fact_part_same": true, "trait_part_same": false, "reason": "Trait part: adds 'a good
   crosser'."}

Example 6, rule 5. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
ROUTER fact part: strikers over 1.88 m under 22
GOLD trait part: good at shielding the ball
ROUTER trait part: good at shielding the ball
Answer: {"fact_part_same": false, "trait_part_same": true, "reason": "Fact part: 'over 1.88 m'
   leaves out players of exactly 1.88 m."}

Example 7, rule 6. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
```

```
ROUTER fact part: young strikers of at least 1.88 m
GOLD trait part: good at shielding the ball
ROUTER trait part: good at shielding the ball
Answer: {"fact_part_same": false, "trait_part_same": true, "reason": "Fact part: 'young'
   replaces the stated bound 'under 22'."}

Example 8, rule 7. Message:
QUESTION: Which wingers are good crossers?
GOLD fact part: wingers
ROUTER fact part: wingers
GOLD trait part: good crossers
ROUTER trait part: good crossers from the left
Answer: {"fact_part_same": true, "trait_part_same": false, "reason": "Trait part: 'from the
   left' narrows the quality."}

Example 9, rule 8. Message:
QUESTION: Find me strikers of at least 1.88 m under 22 who are good at shielding the ball.
GOLD fact part: strikers of at least 1.88 m under 22
ROUTER fact part: strikers of at least 1.88 m under 22
GOLD trait part: good at shielding the ball
ROUTER trait part: young and good at shielding the ball
Answer: {"fact_part_same": true, "trait_part_same": false, "reason": "Trait part: 'young'
   repeats the age condition of the fact part."}

Answer with fact_part_same, trait_part_same and reason. The reason is one sentence naming what
   differs, and null when both parts are the same.
```

Code writes each question into one user message of five lines, with the text of the question or of the part in place of the angle brackets and `(empty)` for an empty part.

```
QUESTION: <question>
GOLD fact part: <fact part of the gold division>
ROUTER fact part: <fact part of the router's division>
GOLD trait part: <trait part of the gold division>
ROUTER trait part: <trait part of the router's division>
```

The answer is forced through the tool `record_verdict`, whose description and the descriptions of its three fields follow.

```
record_verdict: Record, for the fact part and the trait part, whether the router's part asks for
   the same as the gold part.
fact_part_same: True when the router's fact part asks for the same as the gold fact part.
trait_part_same: True when the router's trait part asks for the same as the gold trait part.
reason: One sentence naming what differs, or null when both parts are the same.
```

C Construction of the Rating Fields

The rating fields of the test collection are made by a script in four steps, so that they meet the conditions of Section 5.3.2.

Roles. Before any field is filled, the script chooses the players who depart from the conditions on purpose. For each kind of departure it takes up to five players who qualify on each of five qualities, the pair of kind and quality with the fewest such players first and, within a pair, the players with the most reports. A player qualifies for a high rating without mention when none of his reports mentions the quality, for a low rating against praise when one of his reports praises the quality with a strength of at least $+3$ and some of his reports are silent on it, and for another main weapon when his reports lean towards the quality.

Coverage. For every report and every field the script draws once whether the field is filled in that report. The chance depends on how common the field is and on the type of the report (Table C.1). For a graded field it is multiplied by the weight of the quality for the player's position divided by 0.65, so a quality that matters more for a position is rated more often for its players. A player chosen to depart then gets the field filled in at least three of his reports, or in all of them where he has fewer.

Table C.1: Chance that a rating field is filled in a report, by how common the field is and by the type of the report, before the position of the player is taken into account.

Field	Examples	Match report	Player report
common	acceleration, top speed, endurance, short passing, decision making, composure	0.80	0.95
middle	agility, heading, one-v-one attacking, shot technique, pressing, vision	0.45	0.75
rare	long passing, chance conversion, crossing, set pieces, interceptions, 10-metre time	0.20	0.50
strengths	tick list	0.60	0.85
weaknesses	tick list	0.35	0.60
main weapon	single choice	0.30	0.75

Standing. For each graded field, the players rated on it in at least one report form its pool. Each gets a target, the sum of his text labels for the quality cut off at -5 and $+5$, plus a random amount between -0.4 and $+0.4$. A player chosen for a high rating without mention gets the target 6, and one chosen for a low rating against praise the target -4. The pool is sorted by target, and a player's place becomes his grade on the scale from 1 to 6, on which 1 is best. The best 8 % get a 1, the next up to 28 % a 2, up to 58 % a 3, up to 82 % a 4, up to 94 % a 5 and the rest a 6.

Value. Each report in which the field is filled starts from its player's grade and moves one step down, not at all or one step up, with the chances $1/4$, $1/2$ and $1/4$. The grade is then held to what the report's own text allows, 1 to 4 where it praises the quality, 3 or 4 where it mentions the quality without leaning either way, 3 to 6 where it criticises the quality and any grade where it is silent. A report of a player chosen for a low rating that praises the quality with at least $+3$ starts

from a 2 instead of his grade. The two fields on which 4 is best turn the grades 1 to 6 into 4, 4, 3, 3, 2 and 1, and the 10-metre time into 1.53 to 1.88 seconds in steps of 0.07. A report ticks a strength for each quality it praises with at least $+1$ and a weakness for each quality it criticises with at least -1, and its main weapon is the quality it praises most, if one reaches $+1$. For a player chosen for another main weapon, a report whose main weapon would be that quality names another quality instead. The role and the position follow the player's position group, and the recommendation follows the mean of the report's text labels.

D Statistics of the Test Collection

The 70 players of the corpus are 24 defenders, 23 midfielders and 23 attackers, and none of them is a goalkeeper. The tables below describe what the scores of the trait search rest on (Section 5.3). Table D.1 gives for each trait of the vocabulary how many reports speak of it and in which direction, Table D.2 the spread of the strengths of the text labels, Table D.3 the sizes of the relevant and removed sets of the trait questions, Table D.4 the situations a player is marked with, Table D.5 the trait tags and Table D.6 the question tags (Section 5.3.5). Table D.7 gives the tags of the fact questions, by which the success rate of the MQL sub-agent is split (Section 5.2.2).

Tables D.8 and D.9 give the scores of the trait search per situation and per trait tag (Section 5.3.7). Tables D.10 and D.11 give the success rate of the MQL sub-agent per tag and the outcomes and first causes of its wrong runs (Section 5.2.4).

Table D.1: How often the reports speak of each trait of the vocabulary, and which rating fields measure it.

Trait	Reports	For	Neither	Against	Players	Rating fields
acceleration	307	186	35	86	64	Acceleration, 10-metre time (s), Strengths (option), Main weapon (option)
top speed	295	171	39	85	64	Top speed
change of direction	126	70	7	49	44	Agility
balance	35	26	1	8	23	Body control
physical strength	419	236	60	123	67	Physical robustness, Strengths (option), Main weapon (option)
stamina	81	58	3	20	32	Endurance
jumping reach	88	49	10	29	28	Leap
heading technique	50	37	2	11	20	Heading technique, Strengths (option), Main weapon (option)
first touch	139	82	8	49	39	First touch
ball retention	181	90	18	73	51	Retention under pressure
short passing	274	203	27	44	57	Short passing, Strengths (option)
long passing	81	57	9	15	31	Long passing, Strengths (option)
through balls	71	44	5	22	26	–
beating a man	137	79	11	47	42	One-v-one attacking
ball carrying	161	138	7	16	42	–
shot technique	73	47	9	17	32	Shot technique, Strengths (option), Main weapon (option)
chance conversion	116	66	19	31	34	Chance conversion
crossing	95	59	14	22	31	Crossing, Strengths (option)
set pieces	27	22	3	2	16	Set pieces, Strengths (option)
tackling	157	92	15	50	38	Tackling
defensive positioning	151	89	15	47	41	Defensive positioning
marking	67	43	4	20	33	–
interceptions	37	37	0	0	17	Interceptions
pressing intensity	123	88	13	22	35	Pressing
spatial awareness	132	107	5	20	42	Vision, Strengths (option), Main weapon (option)
decision making	166	60	23	83	49	Decision making
composure	111	75	6	30	50	Composure, Strengths (option)
resilience	43	30	2	11	22	–

Trait	Reports	For	Neither	Against	Players	Rating fields
work rate	269	210	11	48	62	–
leadership	49	42	3	4	23	–
coachability	7	4	1	2	6	–
development potential	305	184	37	84	60	–
versatility	112	84	13	15	40	–

For counts the reports whose text label for the trait is $+1$ or more, Against those of -2 or less and Neither the others, and Players counts the players with at least one such report.

Table D.2: The text labels by value on the scale from -5 to $+5$.

Value	-5	-4	-3	-2	-1	0	$+1$	$+2$	$+3$	$+4$	$+5$
Text labels	7	115	537	526	179	256	352	784	1,297	385	47

Table D.3: The sizes of the relevant and the removed sets over the 65 trait questions, of 70 players.

Set	Minimum	Lower quartile	Median	Upper quartile	Maximum
relevant	0	15	26	31	49
removed	0	13	19	26	59

Table D.4: The situations a relevant or removed player is marked with, and how many relevant and removed players are in each over the 59 questions of the main population.

Situation	Rule	Relevant	Removed
Text only	relevant, and his reports support a trait asked that a rating field measures, but no rating of his does	114	–
Ratings only	relevant, and his ratings support a trait asked on which none of his reports speaks	192	–
Text and ratings	relevant, and his reports and his ratings both support the same trait asked	672	–
Implicit only	relevant, and his reports support an asked trait only through implicit labels	540	–
Most traits	relevant in a question of three or more traits without support on every one	88	–
Reports disagree	one of his reports speaks for a trait asked and another against it	402	406
Rating against praise	a report praises a trait asked with a strength of at least $+3$ while his ratings speak against it	5	149
Named a weakness	his reports lean against a trait asked and a scout ticked it as a weakness	11	205
Another main weapon	his reports lean towards a trait asked and his main weapon is always another quality	128	59
Against one trait	removed in a question of two or more traits, with support on every trait but the one his reports speak against	–	160

A player is counted once for every question he is in.

Table D.5: The trait tags of the 59 questions of the main population.

Trait tag	Rule	Traits	Questions
Rare trait	fewer reports speak of it than a third of the median trait	balance, set pieces, interceptions, coachability	11
Text-only trait	no rating field measures it, so only the text can show it	through balls, ball carrying, marking, resilience, work rate, leadership, coachability, development potential, versatility	22

A question carries the tags of the traits it asks.

Table D.6: The question tags of the 65 trait questions.

Question tag	Rule	Questions
Too many traits	asks more than five traits, the most the trait search divides a question into	2
Asks for an absence	asks that a player lacks a weakness rather than has a quality	2
No right answer	asks only qualities outside the vocabulary, so no player is relevant	2

A question with one of them is reported on its own line instead of in the main population.

Table D.7: The tags of the 41 fact questions of the main population.

Tag	From	Rule	Questions
Aggregation	wording	the answer needs a quantity combined from several documents or entries, such as a count, a sum, an average, a peak or a ratio	12
Ranking	wording	the answer is the first players by a quantity, over all players or within each group	6
Set logic	wording	the answer takes one set of players out of another or unites two sets from different sources	6
Stored value	wording	the question names a value only the documents hold, such as a club, a competition, a nationality or one of the club's lists	27
Search	product	the player search of the product can answer the question with its own filters and sort	11
Array	gold query	the gold query reads an array of embedded documents	24
1 hop	gold query	the gold query has 1 hop	9
2 hops	gold query	the gold query has 2 hops	16
3 hops	gold query	the gold query has 3 hops	12
4 hops	gold query	the gold query has 4 hops	4

A question can carry several tags, and the tags from the wording are assigned by hand.

Table D.8: Recall of the relevant players and harmful inclusion of the removed players in each situation of Table D.4, pooled over the 59 questions of the main population and the five runs, with and without the judge on GPT-6 Sol and GPT-6 Luna.

Situation	Relevant	Removed	Model	With the judge		Without the judge	
				Recall	Harmful inclusion	Recall	Harmful inclusion
All players	1,466	1,245	Sol	0.762	0.105	0.886	0.260
			Luna	0.736	0.085	0.885	0.258
Text only	114	–	Sol	0.732	–	0.842	–
			Luna	0.714	–	0.840	–
Ratings only	192	–	Sol	0.823	–	0.892	–
			Luna	0.825	–	0.896	–
Text and ratings	672	–	Sol	0.892	–	0.944	–
			Luna	0.879	–	0.942	–
Implicit only	540	–	Sol	0.618	–	0.806	–
			Luna	0.583	–	0.807	–
Most traits	88	–	Sol	0.648	–	0.702	–
			Luna	0.611	–	0.714	–
Reports disagree	402	406	Sol	0.792	0.163	0.917	0.259
			Luna	0.774	0.143	0.912	0.249
Rating against praise	5	149	Sol	0.600	0.118	0.200	0.122
			Luna	0.720	0.107	0.200	0.121
Named a weakness	11	205	Sol	0.745	0.098	0.691	0.155
			Luna	0.618	0.083	0.745	0.155
Another main weapon	128	59	Sol	0.706	0.220	0.863	0.217
			Luna	0.672	0.197	0.869	0.224
Against one trait	–	160	Sol	–	0.280	–	0.352
			Luna	–	0.231	–	0.336

Relevant and Removed count the players of a situation over the questions, a player once for every question he is in. Harmful inclusion counts all removed players, including those whom the ratings remove. A dash marks a situation that holds no relevant or no removed player.

Table D.9: The trait search on the questions of the main population that ask a rare or a text-only trait, each set against the other questions, in the standard configuration on GPT-6 Sol and GPT-6 Luna.

Questions	Number	Model	nDCG@10	Random level	Recall	Harmful inclusion	Discounted harm@10	Players
Rare trait	11	Sol	0.607	0.161	0.615	0.052	0.065	18.1
		Luna	0.571		0.584	0.047	0.051	16.6
Other questions	48	Sol	0.818	0.234	0.746	0.100	0.043	27.5
		Luna	0.825		0.718	0.080	0.044	25.5
Text-only trait	22	Sol	0.627	0.200	0.552	0.128	0.075	21.4
		Luna	0.631		0.495	0.106	0.072	18.3
Other questions	37	Sol	0.869	0.232	0.822	0.070	0.031	28.4
		Luna	0.865		0.811	0.056	0.030	27.1

A question carries the tag of each trait it asks (Table D.5). The random level is the nDCG@10 of a random ranking and the same on both models. Harmful inclusion and discounted harm@10 count all removed players, including those whom the ratings remove. Each question enters with its mean over five runs, and the groups are described without a test.

Table D.10: The success rate of the MQL sub-agent per tag of Table D.7, in the standard configuration on GPT-6 Sol and GPT-6 Luna.

		Success rate (%)	
Tag	Questions	Sol	Luna
Aggregation	12	71.7	63.3
Ranking	6	96.7	90.0
Set logic	6	93.3	76.7
Stored value	27	93.3	83.7
Search	11	98.2	90.9
Array	24	85.0	76.7
1 hop	9	75.6	77.8
2 hops	16	97.5	88.8
3 hops	12	90.0	83.3
4 hops	4	90.0	60.0
All questions	41	89.8	82.0

A question can carry several tags, and each question enters with its share of correct runs over five runs. The groups are described without a test.

Table D.11: The wrong runs of the MQL sub-agent by outcome and by the cause of their first wrong step, over the five configurations and the end-to-end runs, 1,230 runs on each of GPT-6 Sol and GPT-6 Luna.

		Sol		Luna	
Outcome or cause	Rule	Runs	Questions	Runs	Questions
No answer	reached the step limit and attached no player	2	–	7	–
Empty answer	finalised without a player	1	–	36	–
Too few	only right players, but not all of them	42	–	54	–
Too many	all right players and others besides	35	–	41	–
Partly right	right and wrong players, and some right ones missed	40	–	59	–
None right	only wrong players	1	–	25	–
Wrong order	the right players in the wrong order	4	–	0	–
Schema linking	answered from the wrong collection or field, or never opened a second place that holds the same fact	67	7	94	10
Wrong value	read the right field with a wrong stored value	0	0	5	3
Filter or condition	applied a wrong condition, or none, on the right field	27	1	57	4
Join or hop	linked two collections by the wrong key	0	0	2	1
Aggregation or set logic	computed a quantity or combined two sets wrongly	0	0	14	3
Process: sort rule	replaced a computed ranking by the sort on a stored field that the prompt's rule for a list prescribes	4	2	0	0
Process: lost result	lost a right result in the last steps in another way	0	0	1	1
Process: step limit	reached the step limit without an earlier wrong step	2	1	0	0
Stated reading	took a reading the wording of the question allows and stated it in the answer or asked back	25	6	49	5
All wrong runs		125	14	222	21

Claude Opus 5.5 assigned each cause after reading the trace of the wrong run. Each wrong run has one outcome and one cause, so both blocks sum to all wrong runs. A cause chosen without a stated reading counts by what the run chose, a field under schema linking and a value under wrong value. A question counts once for each cause it occurs with, so the questions of the causes sum to more than all failing questions.

E Further Analyses of the Trait Search

The tables and the figure below give the numbers of the analyses, chosen after the runs, on which Sections 5.3.7 and 6.2 draw. All of them read the standard configuration on the English questions of the main population, with five runs on both models.

The first analysis concerns how the retriever orders the evidence (Table E.1). Averaged over the 30 traits with at least 5 chunks for and 5 against them, a chunk that speaks for a trait ranks above one that speaks against it with a probability of 0.66 [0.63, 0.69] in the ranking that a run fuses from its two query variants. Over all 33 traits, a chunk with evidence on the trait ranks above a chunk from a report that does not speak of the trait with a probability of 0.64 [0.62, 0.66]. Of the evidence among the first 10, 50, 250 and 500 chunks of the fused ranking, 6, 9, 13 and 16 % speaks against the trait, against 25 % in the corpus. The first 500 chunks hold 57 % of the evidence for a trait and 31 % of the evidence against it.

Table E.1: The evidence on the 37 questions of the main population that ask a single trait of the vocabulary, by the signed strength of its text label from -5, strongly against the trait, to $+5$, strongly for it.

Strength	-5	-4	-3	-2	-1	0	$+1$	$+2$	$+3$	$+4$	$+5$
Chunk–trait pairs	16	158	652	599	220	353	428	899	1,581	615	93
Mean rank	0.52	0.50	0.44	0.44	0.42	0.40	0.40	0.35	0.28	0.27	0.23
In the first 250 chunks (%)	8.8	6.6	15.2	17.3	19.1	20.8	21.9	26.9	38.7	39.5	49.1
Read by the judge (%)	9.4	7.6	13.2	16.8	18.8	17.8	20.4	23.3	29.6	24.6	26.7

A chunk–trait pair is a chunk that holds a passage labelled for the trait. The mean rank orders all 1,808 chunks by their mean similarity to the query variants of the question, from 0 at the top to 1 at the bottom. The two shares count each pair once in each of the 370 runs, five on each model for each question, in the ranking the run fused from its query variants.

The second analysis follows what the judge was shown (Table E.2). In 2,211 cases over both models, a passage for the trait that the judge was not shown stood in a report of which it read another chunk. There it lost to the chunk of that report that came closest to the query variants by a median of 0.050 in cosine similarity, against a median standard deviation of 0.055 of a search query's similarities over all chunks. The losing passage was implicit in 66 % of these cases, against 40 % of the evidence the judge was shown. Its chunk was not longer than the winning one, with a median of 239 against 247 characters, and held passages for fewer traits, 4.0 against 4.4 on average. With 10 chunks per player the judge's call would take about 22.5 instead of 11.3 thousand prompt tokens per trait, and with whole reports about 33.5 thousand.

The third analysis sets the judge against the text labels where both read the same passage (Table E.3). A reading is one score the judge gave a chunk in one run, and the comparison takes the readings of chunks that hold a passage labelled for the trait asked. The judge scores closer to zero than the labels, so a label of $+5$ receives $+3.2$ from it on average and a label of -3 receives -2.1. A blind comparison then took 50 cases, of which 40 were disagreements and 10 agreements as a control. A model of the labelling model's family read each case and saw the two scores in random order, without knowing which came from the judge (Table E.4). Weighted by how often each kind of case occurs, the label was the better reading in 51 % of the disagreements in which the judge scored lower, the judge in 37 % and both in 13 %.

Table E.2: What the judge on GPT-6 Sol and GPT-6 Luna was shown for the relevant players whose reports hold a passage for the trait, on the questions that ask one trait, what it decided and where a passage it was not shown lay.

	Share (%)	
	Sol	Luna
Relevant players with a passage for the trait, 5,055 on each model		
Read by the judge	86	86
Shown the passage, of those read	76	75
Admitted, of those read and shown the passage	90	86
Admitted, of those read and not shown it	39	28
Never shown the passage, of those not admitted	81	77
Read players not shown the passage, 1,061 and 1,067		
Passage beyond the player's 5 best chunks	42	42
Passage beyond the first 500 chunks of both query variants	55	56
Passage among the 5 best chunks, missed by the approximate search	3	2
Read players shown the passage, in a simulation of another budget		
10 best chunks per player	89	89
Whole reports of the chunks shown	90	90

A player counts once for each run in which a question asks the trait. The judge reads a player among its first 50 and is shown the passage when a chunk that holds it is among the chunks it reads of him. The players not admitted include those it never read. The approximate search of the index misses a few chunks that an exact comparison places among a player's 5 best. The simulation changes only what the judge reads.

Table E.3: The judge on GPT-6 Sol and GPT-6 Luna against the text labels on chunks that hold a passage labelled for the trait asked.

				Same side as the label (%)	
Readings	Number	Weighted kappa	Judge minus label	For	Against
All	38,269	0.77	−0.40	85	66
Sol	19,224	0.80	−0.31	89	67
Luna	19,045	0.73	−0.49	82	64
Explicit label	22,688	0.79	−0.40	90	70
Implicit label	15,581	0.72	−0.40	78	59

The kappa is quadratically weighted (Cohen, 1968). For is the share of the readings labelled $+1$ or more that the judge scores $+1$ or more, and Against the same share for -2 or less. An implicit label rests on a passage that implies the trait.

Table E.4: The blind comparison of 50 cases by stratum of the judge's score against the label.

Stratum	Cases	Label better	Judge better	Both fit
Judge lower, explicit label	15	9	2	4
Judge lower, implicit label	15	7	7	1
Judge higher	5	2	2	1
Judge $+1$ or more on a chunk without a label	5	5	0	0
Agreement, as a control	10	0	0	10

Each case showed the definition of the trait, the chunk and the whole report. The blinding was partial, since most disagreements were chosen with the judge lower and a missing label showed as a score of 0.

The fourth analysis assigns each relevant player who is missing from the result of a run to the first step that lost him (Figure E.1). Each pair of a relevant player and a run counts once, 7,330 on each model, and 1,748 of them miss the result on Sol and 1,938 on Luna. The steps follow the order in which the trait search runs them, the division of the question, retrieval, the judge, the matching of the rating fields and the ratings, and a pair goes to the first step at which the

evidence for the player was lost. A pair counts toward the matching when the fields of the gold would have admitted the player, and toward the ratings when even those fields would not.



Figure E.1: Retrieval lost about 60 % of the relevant players missing from the result, the judge about a quarter and the ratings the rest, on GPT-6 Sol and GPT-6 Luna. Each bar splits the pairs of a relevant player and a run that miss the result, 1,748 on Sol and 1,938 on Luna, by the first step that lost the player, on the 59 questions of the main population in the standard configuration.

F Questions of the Test Collection

The 154 questions of the test collection by kind, in their English wording. Each question also has a German phrasing with the same gold (Section 5.1). No question carries the IDs M14, T23, T37 and T38.

ID	Question
Fact questions (43)	
M01	Which players wearing squad number 10 are worth more than €40m?
M02	Which players out on loan have a market value of at least €15m?
M03	Which left-footed centre backs under 21 are worth more than €10m?
M04	Centre backs worth at least €10m whose contract expires in the summer of 2027
M05	Brazilians with a second Italian passport and a market value above €5m
M06	Which are the five most valuable Borussia Dortmund players out of contract by summer 2028?
M07	Who are the ten players worth at least €40m whose market value has risen by the highest percentage since the start of 2025?
M08	Who is on our shortlist for the 24/25 season?
M09	Which of the players we have ever recommended signing, in whatever form, are under 23?
M10	Which players are in the final selection on our scouting board?
M11	Which players without a club still carry a market value of at least €5m?
M12	Which players represented by the agency Gestifute are worth more than €25m?
M13	Which players worth at least €10m are either out on loan or without a club?
M15	Which players worth at least €40m missed more than 100 days through injury in the 25/26 season?
M16	Which players worth at least €40m first reached their peak market value before 2024 and are now worth under 70 % of that peak?
M17	Which of Juventus' current players had a single lay-off of more than 30 days in the 2024 season?
M18	Which players were added to one of our watchlists in the last 30 days?
M19	Which players in the German Bundesliga have no agent?
M20	Which players sit on a watchlist without carrying a scouting status?
M21	Which players worth at least €50m are the subject of at least three transfer rumours?
M22	Which centre backs in the English Premier League are at least 1.95 m tall?
M23	Who did we ever put into the narrow selection on our scouting board (anyone who was ever there, not just those still there)?
M24	Which players are on at least two of our watchlists?
M25	Who is the youngest player worth at least €1m in each LaLiga squad?
M26	Which players sit on a watchlist and are flagged as an injury risk?
M27	Which players in the narrow selection are not on our shortlist for the 25/26 season?
M28	Which of our loan targets have we not flagged as an injury risk?
M29	Which players are linked with Borussia Dortmund at a transfer probability of at least 30 %?
M30	Which players were rated 1 or 2 for Acceleration in a report?
M31	Which players worth at least €30m scored in more than half of the games they played in the 24/25 season?
M32	Who are the five players worth at least €5m who were youngest when they made their English Premier League debut?
M33	Which players in the current Serie A squads made their Serie A debut before turning 18?
M34	Who were the top five scorers in the French Ligue 1 in the 24/25 season?

ID	Question
M35	Which players worth at least €30m have seen their minutes halved in the 25/26 season compared with 24/25?
M36	Which players were in a Juventus matchday squad in the 2024 season and had a lay-off of more than 30 days that season?
M37	Which players average 3 or better across all their Chance conversion ratings?
M38	Which players in the current Serie A squads are worth more than three times their own club's average?
M39	Which players in the current Serie A squads are either on one of our watchlists or carry one of our tags?
M40	Which players were worth over €30m when they moved in the summer transfer window of the 25/26 season and played under a quarter of the available minutes for their new club in that season?
M41	Which players worth at least €20m scored 15 goals or more in Serie A across the last two seasons?
M42	The ten players with the most minutes under Bayern Munich's current head coach
M43	Which players hold a pilot's licence?
M44	Which players have released a music album?
Trait questions (65)	
T01	Which players have an explosive first step?
T02	I need players with the strength to hold defenders off.
T03	Who are the players with a real spring?
T04	Show me players with a cleanly directed header.
T05	Which players have a clean first touch?
T06	I need players with the dribble to beat a man.
T07	Who are the players with the ability to drive the ball up the pitch?
T08	Show me players with a clean strike.
T09	Which players have good timing in tackles?
T10	I need players with sound defensive positioning.
T11	Who are the players with the intensity to press from the front?
T12	Show me players with the ceiling to play at a higher level.
T13	Which players are solid defensively?
T14	I need players with clinical finishing.
T15	Who are the players with the engine for ninety minutes?
T16	Show me players with the ability to keep the ball under pressure.
T17	Which players have a crisp short pass?
T18	I need players with the range to switch the play.
T19	Who are the players with the awareness to scan before receiving?
T20	Show me players with good decision-making on the ball.
T21	Which players have composure in the big moments?
T22	I need players with the work rate to run for the team.
T24	Show me players with the character to bounce back from a mistake.
T25	Who are the players with the reading to cut out passes?
T26	I need players with real top-end speed.
T27	Which players have the range to be used in more than one role?
T28	Show me players with the leadership to take responsibility.
T29	Which players have aerial ability?
T30	I need players with a sharp change of direction.
T31	Who are the players with the balance to ride a challenge?
T32	Show me players with a line-breaking pass.
T33	Which players have the finishing to bury a chance?
T34	I need players with dangerous crossing.
T35	Who are the players with the quality to take a set piece?

ID	Question
T36	Show me players with tight marking discipline.
T39	Who are the players with a real willingness to learn?
T40	Show me players with the versatility to fill more than one role.
T41	Which players have reliable passing?
T42	I need players with security in possession.
T43	Who are the players with real speed over the ground?
T44	Show me players with the work rate to run for the team and a grip on the runner.
T45	Which players have a good ball into the box and sharp one-v-one dribbling?
T46	I need players with the awareness to know the next pass before receiving and a cool head.
T47	Who are the players with room to step up and quick uptake of instruction?
T48	Show me players with the agility to turn a defender and the balance to stay upright through contact.
T49	Which players are not weak in the air?
T50	Which players have no obvious weakness on the ball?
T51	Which players have the habit of scanning early, endurance over the full ninety and reliable passing?
T52	I need players with aggressive pressing, a decisive tackle and a sharp eye for the interception.
T53	Who are the players with a short memory for mistakes, a voice that organises the others and an ear for coaching?
T54	Show me players with the will to demand the ball when it is hard. There should be more still to come from them.
T55	Which players have a shot from distance and a grip on the runner?
T56	I need players with secure possession under pressure, clean short passing, a head that is always up and the running to last the full match.
T57	Who are the players with an extra yard over the first few strides, a first touch that goes forward, the appetite to take people on, a shot the keeper does not enjoy, a high return in front of goal and relentless front-foot pressing?
T58	Show me players with the timing to come away with the ball in the tackle, the positional sense to be in the right place, tight man-marking, good heading technique, a leap above the marker, the strength to win the physical exchange and a long passing range.
T59	Which players have an explosive first step and real speed over the ground?
T60	I need players with a real spring and the strength to hold defenders off.
T61	Show me players with reliability in tight areas, the habit of switching the play and the vision to play a line-breaking pass.
T62	Who are the players with good timing in tackles, tight marking discipline and the reading to be in the right place before the danger arrives?
T63	Which players have ruthless finishing, genuine straight-line speed, sharp one-v-one dribbling and the technique to strike a ball cleanly from distance?
T64	I need players with speed of thought on the ball, the scanning to know what is around before the ball arrives, the assurance not to give it away and the willingness to run for team-mates.
T65	Show me players with sound defensive positioning, good heading technique, a grip on the runner, an instinct for cutting out passes and the body to hold opponents off the ball.
T66	Who are the players with clean short passing, a diagonal that finds the far side, a calm head when it matters, aggressive pressing and the engine for ninety minutes?
T67	Show me players who write clean, well-documented code.
T68	Who are the players that can plate a five-course menu?
Hybrid questions (46)	
H01	Find me a classic number ten.
H02	Which centre backs of at least 1.90 m are dominant in the air?
H03	I need someone with a left foot and a dangerous ball from a corner or a free kick.
H04	I need a target man.
H05	I need a centre back who is quick over the first ten metres.
H06	Which defenders would have the upper hand in a physical duel at Premier League level?

ID	Question
H07	Show me left-footed players of at least 1.83 m with the habit of switching the play.
H08	Which players have at least five ratings on Acceleration and the burst to be gone in three strides?
H09	Which players on at least three of our watchlists have the awareness to scan before receiving?
H10	Who are the English Premier League players with genuine straight-line speed?
H11	Which players under observation on our scouting board have the upper hand in a physical duel?
H12	Which players out on loan and worth at least €15m have balance in the challenge?
H13	Which left-footed centre backs under 21 and worth more than €10m have a change of direction at speed?
H14	Among Brazilians with a second Italian passport and a market value above €5m, who has the ceiling to play at a higher level?
H15	Among players worth at least €40m who missed more than 100 days through injury in the 25/26 season, who has a sharp change of direction?
H16	Which centre backs have a high return in front of goal?
H17	I need defenders with a shot from distance.
H18	I need centre backs with the pass that splits the last line.
H19	Among players at Borussia Dortmund who are out of contract before summer 2028, who has a clean pass under pressure?
H20	Take the players we have recommended signing who are under 23 — which of them have the leadership to take responsibility?
H21	I need players in the final selection on our scouting board with tight marking discipline.
H22	Which players represented by the agency Gestifute and worth more than €25m have a clean tackle?
H23	Take the players worth at least €10m who are either out on loan or without a club — which of them have the range to be used in more than one role?
H24	Of the players at Juventus who had a single lay-off of more than 30 days in the 2024 season, which have a real willingness to learn?
H25	Among players in the German Bundesliga with no agent, who has a sharp eye for the interception?
H26	Which players on a watchlist without a scouting status have a long passing range?
H27	Which players worth at least €80m have the confidence to go past a defender and ruthless finishing?
H28	Which forwards have relentless pressing, are under 24 and have the finishing to bury a chance?
H29	Show me forwards under 25 with dangerous crossing and quality on set pieces.
H30	Which centre backs have strength in the air from crosses, are at least 1.87 m tall and still find a clean pass under pressure?
H31	I need midfielders worth at least €40m with an instinct for cutting out passes and a diagonal that finds the far side.
H32	Which midfielders can play a ball that finds the runner in behind, are still under 25 and have the assurance not to give the ball away?
H33	I need defenders of at least 1.85 m with a cleanly directed header and a real spring.
H34	Which midfielders under 24 have real press resistance and an appetite for the dirty work?
H35	Which midfielders have the range to switch the play, are worth at least €80m and show speed of thought on the ball?
H36	Who is on our shortlist for the 25/26 season with the touch to receive on the half-turn and a calm head when it matters?
H37	Which of the players we recommend signing have an eye for the right option under pressure and the willingness to run for team-mates?
H38	Left-footed players under 25 with a clean first touch and composure in the big moments
H39	Which players have the freedom to carry the ball into midfield, play in LaLiga and have the intensity to press from the front?
H40	Take the centre backs worth at least €10m whose contract expires in the summer of 2027 — which of them have quick uptake of instruction and the positioning to hold the line and see danger early?
H41	Among players worth more than €40m wearing squad number 10, who has the balance to ride a challenge and a carry at high tempo?
H42	Show me players on our shortlist for the 24/25 season with a cool head and an ear for coaching.

ID	Question
H43	Among players without a club who still carry a market value of at least €5m, who has a head that stays up after a setback and the legs to last?
H44	Which players have positional discipline, were added to one of our watchlists in the last 30 days and show the potential for a bigger club?
H45	Of the players worth at least €50m who are the subject of at least three transfer rumours, which have the timing to win the tackle and tight man-marking?
H46	Which forwards worth at least €60m have pace, the dribble to beat a man and a clean strike?

G Accepted Second Answers

On 8 of the 41 fact questions of the main population, the gold accepts at least one second answer beside the answer its gold query returns, and a run counts as correct when it matches any answer of the gold. The fact questions were written to leave nothing open (Section 5.2.1), because the evaluation scores a single turn of a conversation. In use, a question that allows two readings is settled in a further turn, when the scout answers the question the system asks back (Section 3.2.2) or asks again in other words. In the evaluation nobody answers, since the clarification route is switched off and a run that asks back counts as wrong. On a question with two readings, a run could therefore count as wrong because the evaluation leaves out that turn, and not because the MQL sub-agent made a mistake.

The second answers were added after the runs. Tracing each wrong run to its cause showed that some went back not to the MQL sub-agent but to the question or to the data. (1) The wording of three questions allows a second reading that was missed when the questions were written. (2) On six questions a correct query over the stored data returns another answer than the gold query, because the frozen copy of the database holds the fact in a later state than the date the question takes as today, holds only part of it, or holds it wrongly. One question, M40, falls under both.

A second answer was accepted only where a run that gives it made no mistake of its own. A second reading had to be one that the wording allows and a careful scout could hold. A second answer from the data had to come from a correct query over the stored data whose steps keep to the sub-agent's instructions and to the notes of the collections, so that it differs from the answer of the gold query only because of the data. The evaluation tests whether the sub-agent queries the stored data correctly, not whether the data are current or complete, so such a run counts as correct. However, a run that reached a defensible answer by going against its instructions or the notes stays wrong.

Two of the second readings arise from the club data made for the evaluation, in which several names are similar. A scouting status "Shortlisted" stands beside watchlists named "Shortlist 24/25" and "Shortlist 25/26", and a tag "Loan target" beside a watchlist "Loan options". The question for the players in "the narrow selection" who are not on the shortlist for 25/26 (M27) therefore also accepts the watchlist "Priority". That watchlist narrows the players down as much as the status does, and the English phrase is the name of neither. The question for the loan targets (M28) likewise accepts the watchlist "Loan options", which serves the same purpose as the tag. A second reading was rejected where the wording does not allow it. Fourteen runs on Luna and one on Sol answered the question for the players in the final selection on the scouting board (M10) with the stage "Passed", the last stage of the board, instead of "Shortlisted". Since "Passed" marks the players the club has declined, they form no selection, and these runs stay wrong.

Table G.1 lists every second answer. Table G.2 sets the success rates that Chapter 5 reports, which count every answer of the gold, beside those that count only the answer of the gold query.

Table G.1: The second answers that the gold of a fact question accepts beside the answer of its gold query, by question.

ID	Question	Rule	Second answer
M19	Players in the German Bundesliga without an agent	data, later state	the clubs of the 2025/26 season from that season's fixtures instead of the stored list of 2026/27, and players filed under the agency "ohne Berater" count either way
M25	The youngest player worth at least €1m in each LaLiga squad	data, later state	the clubs of the 2025/26 season from that season's fixtures
M27	Players in the narrow selection who are not on the shortlist for 25/26	reading	the watchlist "Priority" instead of the status "Shortlisted"
M28	Loan targets not flagged as an injury risk	reading	the watchlist "Loan options" instead of the tag "Loan target"
M33	Players in the current Serie A squads with a Serie A debut before turning 18	data, later state	the clubs of the 2025/26 season from that season's fixtures, the debuts up to the question's date, or both
M38	Players in the current Serie A squads worth more than three times their club's average	data, later state	the clubs of the 2025/26 season from that season's fixtures
M40	Players worth over €30m at a summer move in 25/26 who then played under a quarter of the available minutes	(1) reading (2) data, partial (3) data, wrong	(1) loans counted as moves, (2) the new club's games from its players' own match records, since the fixtures hold only part of the season, with or without loans, (3) the season totals with loans counted, which store one player's league minutes under his old club
M42	The ten players with the most minutes under Bayern Munich's current head coach	(1) data, partial (2) data, later state	(1) all 110 of the coach's matches from the players' own match records, since the fixtures hold 72 of them, and (2) the minutes up to the question's date

The copy of the database holds the data of August 2026, while every question with a later state takes 1 January 2026 as today, so its second answer is the state that the data's history lets a query reconstruct for that day. A row with several rules accepts each second answer on its own and their combinations where the table names them.

Table G.2: The success rate of the MQL sub-agent on the 41 fact questions in each configuration on GPT-6 Sol and GPT-6 Luna, counting only the answer of the gold query as correct and counting every answer of the gold, as the results do.

	Success rate (%)			
	Sol		Luna	
Configuration	Gold query only	Every answer of the gold	Gold query only	Every answer of the gold
Standard	80.5	89.8	76.6	82.0
w/o examples	79.5	88.3	76.1	83.9
Retrieved examples	78.5	88.8	73.2	79.0
Preloaded schemas	77.6	88.8	78.5	82.9
German	82.0	91.2	82.0	83.9
Via router	82.0	92.2	76.1	80.0

H Wording Differences Between the English and the German Questions

The German phrasings of the questions are idiomatic scout language, translated by meaning into the expressions German scouts use rather than word for word (Section 5.1). An audit by a language model compared each German phrasing of the fact and trait questions with its English one and with what the gold fixes, without access to any score. It flagged phrasings that name a quality more broadly or more narrowly than the gold, and logic words that a model may apply otherwise than the gold intends. A scout whose first language is German would have to confirm its findings. Table H.1 lists the questions on which the two phrasings differ in what they ask or led the runs to different conditions.

On the fact questions, the conditions that the runs applied show where the wording mattered. The German wording changed the condition the MQL sub-agent applied on two questions, and on three the English wording was the weaker side. The logic words that the audit flagged were applied as the gold intends in every run of both languages. “3 oder besser” (M37) became an average of at least 3 on a field on which 4 is best, although 1 is best in German school grades. “Entweder … oder” (M39) became an inclusive or, although it is exclusive in German. The German questions that drop the country of a league (M19, M22, M32 and M34) found the right competition, although leagues of the same name exist in other countries.

On the trait questions, the audit found seven in the main population on which the German scout term and the trait of the vocabulary differ in scope. On five of them the German term is broader than the trait or names a neighbouring trait, and on two the English phrasing misses the trait that the German one names.

Table H.1: The questions on which the English and the German phrasing differ in what they ask or led the runs to different conditions.

Question	English	German	Difference
Fact questions on which the German wording changed the condition			
M06	out of contract by summer 2028	einen Vertrag, der bis Sommer 2028 ausläuft	The English runs kept the contracts that expire before 1 July 2028. German runs moved the deadline to the end of August or read a window in 2028, and 4 of 5 lost the answer on Luna.
M36	a lay-off of more than 30 days	eine Verletzungspause von mehr als 30 Tagen	*Verletzungspause* names an injury, and 2 of 5 German runs on Sol left out a lay-off for illness, which every English run counted.
Fact questions on which the English wording was the weaker side			
M10	the final selection	der Endauswahl	Every German run took the scouting status `Shortlisted`. English runs took the board's last stage, `Passed`, in 1 of 5 runs on Sol and 4 of 5 on Luna.
M27	the narrow selection	der engeren Auswahl	The English is a calque of *engere Auswahl*. Every German run took `Shortlisted`, whereas on Luna the English runs took the watchlist `Priority`, which counts as a second reading (Appendix G), or asked back.
M38	their own club's average	der Schnitt ihres eigenen Vereins	The English probably led most runs to an average stored for the club, whereas the gold computes the mean of the squad, as German runs did more often.
Trait questions on which the German term is broader or names a neighbouring trait			
T31	the balance to ride a challenge	Standfestigkeit im Zweikampf	The German names robustness in a challenge, which lies closer to physical strength than to balance.
T44	a grip on the runner	dem Zugriff auf den Gegenspieler	*Zugriff* is the German word for getting at an opponent in pressing and duels and drops the runner, so it is broader than marking.
T52	a sharp eye for the interception	gutem Antizipationsvermögen	Anticipation drops the interception and also covers reading danger early, which is defensive positioning.
T55	a grip on the runner	dem Zugriff auf den Gegenspieler	As in T44.
T65	good heading technique, a grip on the runner	gutem Kopfballspiel, dem Zugriff auf den Gegenspieler	*Kopfballspiel* covers play in the air as a whole and adds jumping reach to heading technique, and *Zugriff* is as in T44.
Trait questions on which the English phrasing misses the trait			
T32	a line-breaking pass	Schnittstellenpässen	The English covers a pass through any line. The trait, through balls, is the pass through the last line, which *Schnittstellenpass* names.
T61	the vision to play a line-breaking pass	dem Auge für den tödlichen Pass	As in T32, the English covers any line, whereas the German names the final pass, which lies closer to the trait.

The fact questions are compared on the conditions their runs applied in the standard configuration, five runs per language on GPT-6 Sol and GPT-6 Luna. Appendix F gives each question in full.

I Test Results of the Hypotheses

Table I.1 gives the mean difference, its bounds and the p-values of every test of Chapter 5 on both models.

Table I.1: The verdict on each hypothesis of Table 5.2 on GPT-6 Sol and GPT-6 Luna, with the router hypothesis once for each component.

				Difference		p-value		
Model	Measure	n	RQ	Mean	Bounds	t-test	Sign-flip	Verdict
The worked examples raise the success rate								
Sol	success rate	41	RQ2	1.5	[−2.7, 5.6]	0.28	0.38	Not supported
Luna				−2.0	[−5.2, 1.3]	0.84	0.88	
Retrieved examples and examples tied to a collection differ								
Sol	success rate	41	RQ2	1.0	[−2.7, 4.7]	0.60	0.84	Not supported
Luna				2.9	[−4.5, 10.3]	0.43	0.52	
Preloaded schemas and schemas read on demand differ								
Sol	success rate	41	RQ3	1.0	[−3.9, 5.9]	0.69	0.84	Not supported
Luna				−1.0	[−6.1, 4.1]	0.70	0.85	
German fact questions are answered less accurately								
Sol	success rate	41	RQ2	−1.5	[−5.3, 2.3]	0.74	0.80	Not supported
Luna				−2.0	[−7.4, 3.5]	0.72	0.76	
The judge keeps players excluded by text out of the top								
Sol	discounted harm@10	53	RQ1	0.027	[0.009, 0.045]	0.008		Supported
Luna				0.026	[0.012, 0.039]	0.001		
Two query variants rank better than the first alone								
Sol	nDCG@10	59	RQ2	0.003	[−0.006, 0.012]	0.30		Not supported
Luna				0.010	[−0.001, 0.020]	0.061		
German trait questions rank the players less well								
Sol	nDCG@10	59	RQ2	0.015	[−0.003, 0.032]	0.086		Not supported, rejects on Luna only
Luna				0.042	[0.018, 0.065]	0.002		
Behind the router the MQL sub-agent scores lower								
Sol	success rate	41	RQ2	−2.4	[−6.5, 1.7]	0.84	0.87	Not supported
Luna				2.0	[−1.5, 5.4]	0.18	0.25	
Behind the router the trait search scores lower								
Sol	nDCG@10	59	RQ2	0.006	[−0.008, 0.019]	0.24		Not supported, rejects on Luna only
Luna				0.022	[0.008, 0.036]	0.005		

Each difference gives how much better the standard configuration, or for the router the isolated runs, scores than the configuration the hypothesis names, so for a hypothesis with a direction a positive value agrees with its prediction. For a hypothesis with a direction the brackets give the lower and the upper one-sided 95 % bound, so its test rejects exactly when the lower bound lies above zero. For one without a direction they give the two-sided 95 % interval. Differences of the success rate are in percentage points. The sign-flip test is run on the fact questions only, and on these a test rejects only when both p-values lie below 0.05. A hypothesis is supported only when its test rejects on both models.

J Glossary

The terms below are used with one meaning throughout the thesis. Where two literatures use one word for different things, the thesis binds the word once, here, and keeps that binding in every chapter.

Question What the user writes and the router receives. Never “user query”.

Query A statement in a database language, written as SQL query, MQL query or database query wherever the retrieval sense is nearby.

Search query The text sent to a retrieval index, the information-retrieval literature’s “query”.

Topic In a TREC-style test collection, an information need with its query and relevance judgements. The items of this thesis’s collection are called questions.

Schema The declared structure of the whole database: every table with its columns and keys, or every collection with its fields. One collection’s part of it is that collection’s schema. A schema fits the context window when its whole text fits in one prompt.

Model The language model, always. Mongoose’s “models” are collection schemas and are called so.

Data model The paradigm of a database, relational or document.

Table, column, row The relational words; their document counterparts are collection, field and document, and a join corresponds to a lookup stage or to nesting. A system is described in the words of its own side, and a finding keeps the words of the side it was measured on.

Schema linking, schema selection Choosing, from the whole schema, the tables and columns (collections and fields) a question needs, before the query is written; what is not chosen is left out of the prompt. Pruning is the same act seen from the schema’s side, at the level of tables (collections) or of columns (fields).

Pruning by a model Schema selection made by a language-model call rather than by a retriever or a rule. Its cost is recall: a field the query needed that the selector dropped.

Collapsing Merging tables that share one layout into one representative, so that the whole schema gets shorter without losing a column type (the table grouping in ReFoRCE’s compression).

Sampling Showing raw rows or documents in the prompt.

Exploration A query the system issues to learn what a field holds before it writes the final query. The thesis’s probe is directed exploration of one field.

Grounding Checking the names and values a query will use against the schema and the stored values before the query runs.

Workflow A system whose steps are fixed in code, the model filling each step (Schluntz and Zhang, 2024). A decision whether a fixed step repeats or which of several predefined paths is taken keeps a system a workflow.

Agent A system in which the model directs its own process, choosing each action from the result of the last (Schluntz and Zhang, 2024; Yao et al., 2023).

Tool A callable function the model chooses to invoke; the tool executes and returns.

Sub-agent An agent that a larger system calls for one part of a task and that returns its result to that system. In this thesis the MQL sub-agent, which the dispatch calls for the fact part of a question.

Execution accuracy The share of questions whose generated query returns the same result as the gold query when both are executed.

Exact match The share of questions whose generated query equals the gold query as text.

Test collection A set of documents, a set of topics and their relevance judgements, on which retrieval systems are measured.

Trait A quality of a player that a scout asks about and that has to be assessed from what scouts wrote and rated, such as pace or heading.

Trait of the vocabulary One of the 33 finest traits of the test collection, against which every report is labelled, such as acceleration or top speed.

Text label The reading of one report for one trait of the vocabulary: whether the report's text speaks of the trait and, where it does, its direction and its strength on the scale from -5 to $+5$.

Relevance judgement An assessor's decision whether, or to what degree, a document answers a topic. Graded relevance has more than two levels.

Run One answer of a component to one question in one configuration, for the MQL sub-agent with the steps that produce it. Every question is answered in five runs per configuration and model.

Pool The merged top ranks of the rankings that the compared systems returned for a topic. What lies outside the pool counts as not relevant.

Nugget A unit of relevant information a topic decomposes into.

Sparse and dense retrieval A sparse retriever scores term overlap between query and document, as BM25 does. A dense retriever compares embeddings of the two.

Bi-encoder, cross-encoder, late interaction A bi-encoder embeds query and document separately and compares the vectors. A cross-encoder reads the two together and scores the pair. Late interaction keeps one vector per token and matches token to token.

Fusion Merging several rankings into one, as reciprocal rank fusion does.

Reranking Reordering a short candidate list with a costlier model than the one that retrieved it.

Entity retrieval, expert finding Ranking things other than documents, with the documents as the evidence. Expert finding ranks people for a topic. Its document model ranks documents and propagates their scores onto the people associated with them; its candidate model ranks one profile per person.

Multilingual and cross-lingual retrieval A corpus with documents in several languages is multilingual retrieval. A query in one language over documents in another is cross-lingual retrieval.

Use of AI Tools

Generative AI tools were used as aids in this thesis, as listed below. They were not used as a source of information. Every statement on the literature rests on the cited sources, and every result on the runs and data of the evaluation. The research questions, the architecture of the system, the choice and combination of its techniques, the structure of its code, the design of the evaluation and the conclusions were decisions that were not delegated to these tools. Every text produced with them was checked and revised before it entered the thesis. The language models that are part of the method, which labelled the reports, checked the divisions of the questions and assigned the causes of errors, are described where they are used (Sections 5.3.3 and 5.4.2).

Tool	Use	Parts of the thesis
Claude Code (Anthropic) with models of the Claude Opus family	discussing and ordering ideas, structuring chapters and paragraphs	whole thesis
	formulating and revising text, checking language and consistency	whole thesis
	searching papers by citation count and relevance to the topic for an overview of the literature, summarising them and comparing the claims the thesis makes about them with what their authors report	Chapter 2, bibliography
	programming assistance for the system and the evaluation scripts, producing tables and figures	Chapters 4 and 5, appendices

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin oder des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 19.10.2026
Ort, Datum



Kristan Böttjer