Automated Extraction and Cross-Document Analysis of Aerospace Project Requirements Using LLMs

Imededdine Ben Slimene
German Aerospace Center (DLR)
Münchner Str. 20
82234 Weßling, Germany
imededdine.benslimene@dlr.de

Maximo A. Roa
German Aerospace Center (DLR)
Münchner Str. 20
82234 Weßling, Germany
Maximo.Roa@dlr.de

Daniel Leidner
German Aerospace Center (DLR)
Münchner Str. 20
82234 Weßling, Germany
Daniel.Leidner@dlr.de

Martin Stelzer
German Aerospace Center (DLR)
Münchner Str. 20
82234 Weßling, Germany
Martin.Stelzer@dlr.de

***Abstract*— Requirements engineering remains a fundamental yet time-consuming activity for aerospace and other complex systems, where large project documents define mission- and safety-critical specifications. Traditional approaches are challenged by the scale, heterogeneity, and ambiguity of such documents, making it difficult to ensure traceability, consistency, and completeness. This paper introduces a human-in-the-loop requirements analysis pipeline that automates requirement extraction, classification, and verification for aerospace project documentation. The tool employs a hybrid architecture: a document-agnostic, Large Language Model (LLM)-assisted parser induces regex-based rules to extract requirement blocks into a normalized database, while subsequent analysis pipelines evaluate the requirements from complementary perspectives. The structural pipeline identifies redundancy, traceability gaps, and document coverage issues; the semantic/logic pipeline leverages LLMs to assess ambiguity, conflicts, and requirement clarity. A human-in-the-loop review process validates and refines outputs for reliability in aerospace contexts. While the system is designed for on-premises execution to satisfy export-control and privacy constraints, our experiments were conducted with cloud-hosted LLMs pending local deployment. An initial evaluation on a synthetic European Cooperation for Space Standardization (ECSS)-style corpus representative of spacecraft project documents demonstrates the feasibility of the approach, with strong performance in structured extraction and classification, and clear insights into current limitations of semantic reasoning. Across 225 requirements, leading models achieve Balanced Accuracy (BAcc) of 0.70–0.80 and we have 3% parse errors. The architecture provides a foundation for integration with formal modeling workflows and future verification tasks.**

TABLE OF CONTENTS

979-8-3315-7360-7/26/$31.00 ©2026 IEEE

1. INTRODUCTION

Aerospace projects capture mission- and safety-critical behavior in large System Requirement Documents (SRDs) and subsystem Technical Requirement Documents (TRDs). Scale, many authors, and frequent revisions make clarity, non-redundancy, and end-to-end traceability hard to sustain. Ambiguities or duplicated intent can persist unnoticed and surface late, where fixes are costly [1].

Industrial practice relies on requirements management (RM) platforms such as IBM DOORS/DOORS Next, Jama Connect, and Siemens Polarion ALM to author, version, baseline, and link requirements artifacts. In parallel, model-based systems engineering (MBSE) environments such as Eclipse Capella (Arcadia) capture the system architecture and behavior, typically integrating with RM tools via ReqIF/OSLC for traceability. These platforms are effective for governance and change control, but they still depend on manual authoring and review; they offer limited assistance for detecting ambiguity or reasoning across multiple documents and subsystems [2–6]. Research in Natural Language Processing (NLP) and Machine Learning (ML) shows promising point solutions: Bidirectional encoder representations from transformers (BERT) models can classify requirement types, and LLM-assisted methods can extract entities and relations to build requirement knowledge graphs [7, 8]. Yet aerospace prose is long and heterogeneous, jargon is domain-specific, and cross-document dependencies are often implicit. Automated ambiguity checkers tend to achieve high recall but low precision on real specifications, conflict identification typically needs structured formalisms or search, and SRD↔TRD trace links are still created and maintained largely by hand [9–11]. In addition, because requirements may include proprietary and export-controlled content, many projects mandate *on-premises* (or Virtual Private Cloud (VPC)-isolated, no-retention) model inference for privacy and compliance; our design therefore targets an on-premises execution path, with cloud-hosted LLMs used only for prototyping and ablation studies.

To address these gaps, this paper contributes a human-in-the-loop pipeline that complements existing RM/MBSE workflows rather than replacing them. First, an LLM-assisted regex/template generator induces document-agnostic parsing rules and converts heterogeneous SRDs/TRDs into a normalized requirements database [12]. Second, a structural analysis path detects redundancy, coverage gaps, and SRD↔TRD

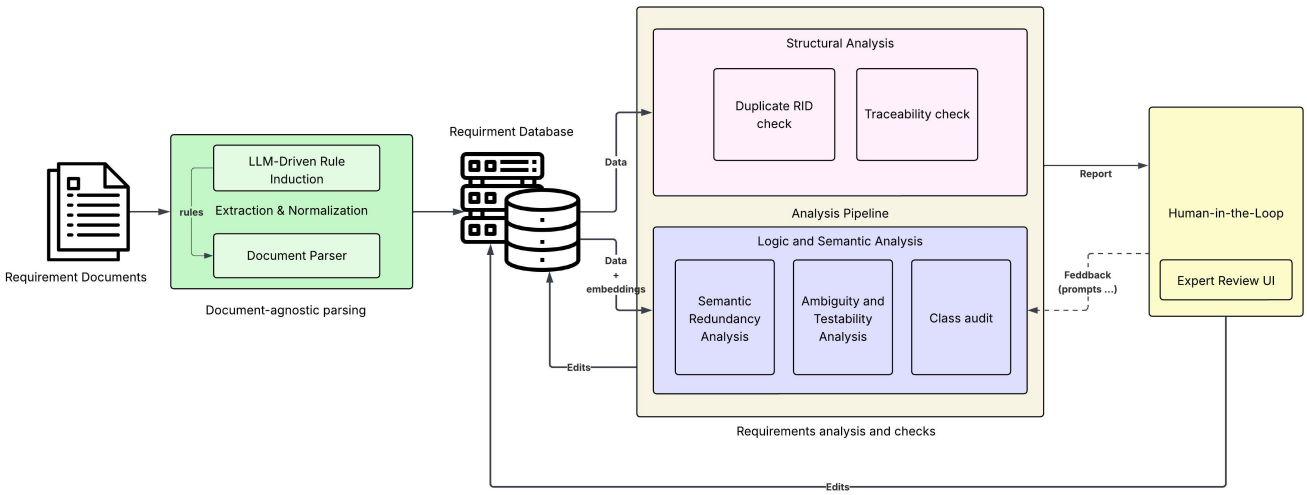


Figure 1: Software architecture for requirement management using LLMs

traceability issues. Third, a semantic and logic path uses LLMs to flag ambiguity and testability concerns [13]. Expert supervision closes the loop: reviewers validate outputs, correct edge cases, and feedback improvements to rules and prompts so results remain reliable and auditable in aerospace contexts.

The paper is organized as follows. Section 2 details the software architecture, including LLM-driven rule induction, the structural pipeline (Requirement ID (ReqID) duplicates, SRD↔TRD traceability), the semantic/logic pipeline (semantic similarity, ambiguity, testability), and human-in-the-loop components. Section 3 details the dataset and evaluation protocol (SRD and subsystem TRDs, expert/LLM labels, a gold set with inter-rater κ, and extraction/task-level metrics including precision/recall/F_1 and confusion matrices). Section 4 presents experiments and results (pipeline runs, model comparisons) and key findings, and Section 5 presents the conclusion.

2. Software Architecture

Overview

Figure 1 summarizes the end-to-end pipeline. Requirement documents enter a parser where a small sample from each document is used to induce regex-style rules with an LLM; these rules are then applied to parse all documents into normalized requirement blocks stored in a project database. On this foundation, two analysis lanes run in parallel: a *structural* lane that checks redundancy, coverage, and cross-document traceability, and a *semantic/logic* lane that analyse semantic duplication and uses LLMs to assess ambiguity and testability. A human-in-the-loop review validates findings, edit the database and feeds back improvements to rules and prompts.

Why not a full LLM pipeline?— We initially experimented with an end-to-end LLM design in which the model parsed entire documents, managed cross-document traceability, and performed semantic audits. This proved impractical: (i) requirement documents are long and heterogeneous, so prompts quickly exceeded feasible context windows, (ii) processing times and costs scaled poorly with large inputs, and (iii) hallucination rates increased as prompts grew and spanned multiple documents. We also tried a lightweight Retrieval-Augmented Generation (RAG) over indexed requirement blocks; in practice, the retrieval windows still missed necessary cross-document context, and stitching many passages reintroduced long prompts without improving reliability. Project-specific fine-tuning on the document set was deemed costly (data curation, training, and validation) with no guaranteed improvement in grounding or output quality, and would need repeating for each new project.

We therefore adopt a hybrid architecture. LLMs are used only where their strengths are clear: *(a) rule induction* from small representative samples, where a profile is synthesized once and then frozen into deterministic regex-style rules; and *(b) focused semantic audits* on short, localized snippets with only the necessary context. All high-volume structural tasks run with explicit, versioned algorithms. This division keeps parsing reproducible and deterministic at scale while applying LLM judgment only where language understanding is essential.

Implementation note: the pipeline is implemented in Python as modular jobs; parsed data are stored in a local SQLite repository with a lightweight embedding index for semantic search; batch runs execute via a Command-line interface (CLI) with report exports; LLM components are invoked through a cloud Application Programming Interface(API); and a small Graphical User Interface (GUI) provides interactive visualization and review.

LLM-driven rule induction

We handle all requirement files as *PDF* documents; the pipeline does not assume access to native sources (Word/LaTeX) and always starts from PDFs. Heterogeneous requirement-document styles make fixed parsers brittle—page headers, numbering schemes, and field labels vary by subsystem—so hand-written rules do not transfer well. To stay document-agnostic, we induce parsing rules per document family with a short, repeatable procedure.

Sampling and peek JSON— For each PDF, a lightweight prober extracts a compact "peek": table-of-contents cues, a few representative pages, and simple signals (ID-like to-

kens, label-like lines, and normative verbs such as *shall/must/should*) as provided in the Appendix 5 (Listing 1). This peek is serialized as JSON and serves as the only input to the rule synthesizer.

Rule synthesis with an LLM—A structured prompt requests a strict JSON "regex profile" that specifies: the requirement ID family, header and inline ID regexes, separators, a canonical field set with label patterns, a statement detector (*shall/must/should*), and optional helpers (e.g., decorative labels, verification tokens) as provided in Appendix 5 (Listings 2). The LLM returns this profile as JSON; no free text is accepted. We then compile the regex profile and validate it (rejecting unsafe or incomplete patterns) before use.

Parsing and normalization—Using the compiled profile, the full document set is parsed into normalized requirement blocks. Each block stores at least: ReqID, the requirement statement, section/Table of Content (ToC) path, optional parent link, and any extracted metadata (labels, verification hints). The blocks are written to a project database and become the single source of truth for downstream analysis.

Because rules are induced from local evidence, the same pipeline adapts across subsystems without hard-coded layouts. Analysts review early parses, fix edge cases, and feedback corrections to the prompt or label maps. This keeps the parser precise while preserving portability. The following subsections build on this foundation: the structural pipeline uses the normalized blocks to check redundancy, and cross-document traceability; the semantic/logic pipeline applies LLM checks for ambiguity and testability.

Structural analysis pipeline

The structural pipeline runs deterministic checks over the requirements repository produced by the parser. It does not interpret meaning; instead, it verifies identifier uniqueness, referential integrity, and cross-document coverage.

Duplicate identifiers—The pipeline reads requirement identifiers (ReqIDs) directly from the database and groups identical values across all documents. Any identifier that appears more than once—within the same document or across different documents—is reported as a duplicate for review.
A "duplicate ReqID" means multiple parsed rows with the same ReqID header; ReqID strings found in Trace/Mother REQ or inline are modeled as references, not counted as duplicates.

Traceability builder—For each subsystem requirement, the pipeline looks for a cited parent—typically the "Mother REQ" or equivalent reference to a system requirement. When the cited parent exists in the repository, a link is created from the TRD-Req to the SRD-Req and if a non existant parent is cited, the TRD-Req is flagged as a *dangling reference*. If no parent is cited at all, the TRD-Req is flagged as *missing trace* (by policy, every TRD-Req should trace to an SRD-Req) [1].

Coverage and orphans—Using the built links, the pipeline computes coverage in both directions. SRD-Req with no incoming links are flagged as *uncovered*. TRD-Req without a valid SRD link are flagged as *orphans*. The tool aggregates these results into per-document and per-section summaries to guide reviews.

Semantic/logic analysis pipeline

This pipeline inspects each requirement's text to surface meaning-level issues that typically require engineering judgment. It can be run on a whole document or scoped to a selected requirement. The checks are independent and complementary: (i) an LLM-based ambiguity and clarity audit, (ii) testability and verification, (iii) semantic duplicate detection using embeddings, and (iv) a class audit that tests whether requirements are filed under the right sections.

Ambiguity audit— We use a structured prompt (see Appendix 5, Listing 3) that applies a compact taxonomy and returns machine-readable labels with a one-sentence rationale. The taxonomy covers: *vague quantifiers* (e.g., "as soon as possible," "sufficient"), *missing bounds* (no numeric limits or conditions), *underspecified conditions* (when/if/while not defined), *modality clarity* (use of non-binding modal verbs where the intended enforceability is unclear), *passive phrasing*, and *unresolved placeholders* : values marked To Be Determined (TBD)/ To Be Confirmed (TBC), indicating immaturity of the requirement).

For modality, we follow common requirements style: *shall* denotes a mandatory requirement; *should* denotes a recommendation. We therefore do not flag *should* when a statement is clearly intended as a recommendation. We only flag *modality clarity* when a sentence otherwise reads like a binding requirement but uses non-binding language that makes enforceability unclear.

The model is constrained to judge only the given requirement text and to return: (a) zero or more ambiguity tags, (b) a short rationale, (c) the exact phrase span that triggered the tag, and (d) a confidence score. Results (see Appendix 5, Listing 4) are designed for triage, not automation; reviewers accept, edit, or dismiss each flag.

Testability audit—The model judges whether a requirement is *testable as written* and gives a brief rationale. When —**A**nalysis, **T**est, **R**eview, **I**nspection —(A/T/R/I) tags are present, it checks consistency with the statement and flags implausible assignments (e.g., marked "A" although acceptance hinges on a controlled test, or marked "T" for pure document conformance). A requirement is considered testable when it yields an objective pass/fail—through explicit numeric bounds, a referenced test procedure, or a binary artifact that can be verified by review or inspection. It is not testable when verification would require adding measurable limits or conditions or when a placeholder (TBD/TBC) gates acceptance. Unlike the ambiguity audit, this check targets *verifiability* rather than phrasing; for non-testable cases, the tool proposes a minimal rewrite to make acceptance objectively checkable, and reviewers confirm or amend the verdict.

Semantic duplicate detection (embeddings)— To reveal duplicated intent expressed with different wording, each requirement statement is embedded into a vector space using the **BAAI/bge-m3** model. We compute cosine similarity between pairs and surface candidates whose similarity exceeds a threshold τ_{dup}:

$$s(i,j) = \frac{\langle \mathbf{e}_i, \mathbf{e}_j \rangle}{\|\mathbf{e}_i\| \, \|\mathbf{e}_j\|}, \quad \text{flag if } s(i,j) \geq \tau_{\text{dup}}.$$

Candidate pairs are clustered into duplicate groups for review. We choose **bge-m3** for its strong retrieval/Semantic Textual Similarity (STS) performance and practical fit to technical specifications: long-context inputs (up to $\sim$8k tokens) and

a unified design supporting dense, sparse, and multi-vector retrieval; here we use its *dense* embeddings for cosine-based detection while retaining a path to lexical or multi-vector modes if needed [14, 15].

Section audit (TOC–kNN)— We derive a lightweight topic map from each document's table of contents by embedding section titles (and brief blurbs) and comparing each requirement's embedding to section embeddings via k-nearest neighbors (k-NN). If the assigned section is not among the top-k matches or its similarity falls below τ_{section}, we flag *suspect placement*.This component is excluded from evaluation because it is not yet mature; a correct end-to-end workflow depends on the standards-aware, cross-document improvements planned in Section. 5.

Workflow and outputs— Each check produces a small, auditable record (labels, spans, scores, and one-line rationales). A reviewer queue groups findings by type and severity: *Ambiguity*, *Not testable*, *Duplicate candidates*, and *Suspect placement*. Quantitative performance for ambiguity and testability is reported in the next sections using precision/recall and confusion matrices; duplicate and class-audit findings are summarized as candidate sets with reviewer outcomes.

Human-in-the-loop

After the automated checks, the tool provides a compact review workspace where reviewers can inspect findings, edit requirement text or metadata, and adjudicate cases (accept, revise, or dismiss). Typical actions include fixing a cited parent, merging or splitting duplicate groups, adjusting ambiguity/testability labels, and relocating a requirement to the correct section. Every decision is stored as a structured annotation with rationale, creating an auditable trail and a reusable "gold" dataset. Accepted edits update the repository, derived artifacts are then recomputed from the updated database—links are rebuilt, coverage and orphan reports refreshed.

3. Evaluation and Metrics

Dataset creation

Corpus.— We use a synthetic mission rather than a real project. Space-project requirements are typically proprietary and often export-controlled, which prevents running third-party LLMs on the text or releasing it publicly without an on-premise setup. A synthetic corpus lets us share data and code for full reproducibility while keeping the structure and phrasing of ECSS-style documents (sections, verification tokens A/T/R/I, units and limits). This choice is appropriate for a first proof-of-concept; we discuss generalization limits in the results.

We compiled a set of **225** requirements from six engineering documents: the System Requirements Document (SRD) and five subsystem TRDs (Communications, Thermal, Power, Onboard Software, Electrical), as presented in Table 1.

Annotation and adjudication.—Each requirement was independently labeled by a human expert and by an LLM (*OpenAI GPT-5*) for *Ambiguity* (Yes/No) and *Testability* (Yes/No) under a *frozen prompt that operationalizes the decision rules* (see Appendix 5, Listing 3). Both annotators followed the same rules; the prompt was fixed after a brief calibration to avoid post-hoc drift. Disagreements were resolved via adjudication to produce the final *gold* labels. The workflow was: (i) double annotation, (ii) disagreement review, (iii) refinement of edge-case examples, and (iv) consensus labels. This follows standard multi-rater practice [16].

Table 1: **Source documents and requirement counts.**

PDF (source)	Acronym	#Reqs
System Requirements Document	GEN	40
Communication Subsystem TRD	COM	31
Thermal Control Subsystem TRD	THM	40
Power Subsystem TRD	PWR	41
Onboard Software Subsystem TRD	SFT	41
Electrical Subsystem TRD	ELC	32
	Total	**225**

Inter-rater agreement.— **Why two raters ?** Ambiguity is context-dependent; a single reader (human or LLM) can introduce systematic bias. Dual annotation lets us quantify consistency, expose prevalence effects (few positives), and detect directional bias (one rater calling more positives). We then adjudicate only what needs attention and refine the adjudication rules with concrete edge cases, this motivates the contingency table and agreement statistics below.

Contingency table. For two raters (R1 and R2) on a binary label (Yes or No), we use the 2×2 table:

	R2:Yes	R2:No
R1:Yes	a	b
R1:No	c	d

a = both Yes, b = R1:Yes/R2:No, c = R1:No/R2:Yes, d = both No, $N = a+b+c+d$.
Notation: N denotes the total count; "No" denotes the negative label.

Observed vs. chance agreement.—We separate "how often we agree" p_o from "agreement expected by chance" p_e given the marginals.

$$p_o = \frac{a+d}{N}, \qquad p_e = \frac{(a+b)(a+c)}{N^2} + \frac{(c+d)(b+d)}{N^2}. \quad (1)$$

Cohen's κ (chance-corrected).—κ rescales observed agreement by subtracting chance and normalizing the headroom; κ=0 means chance-level; 0.6–0.8 is typically *substantial*.

$$\kappa = \frac{p_o - p_e}{1 - p_e}. \quad (2)$$

Positive/negative agreement (prevalence-aware).— Overall agreement can be high while positives are inconsistent when the positive class is rare. PPA/NPA make that explicit.

$$\text{PPA} = \frac{2a}{2a+b+c}, \qquad \text{NPA} = \frac{2d}{2d+b+c}. \quad (3)$$

PABAK (prevalence- and bias-adjusted κ).—A simple adjustment that corresponds to chance-correcting under balanced prevalence; useful in the "prevalence paradox."

$$\text{PABAK} = 2p_o - 1. \quad (4)$$

McNemar's test (directional bias).—Tests whether discordant pairs are symmetric ($b \approx c$). If not, one rater systematically

says "Yes" more often. We report the continuity-corrected χ^2; for small $b+c$ we give the exact binomial p-value.

$$\chi^2_{\text{McNemar}} = \frac{(|b-c|-1)^2}{b+c}. \quad (5)$$

Decision policy during gold construction.—

- **Substantial agreement:** if $\kappa \geq 0.75$ and PPA≥ 0.80, trust agreements; adjudicate only $(b+c)$ and keep the rubric.
Note: The rubric is the adjudication rules in our case.
- **Moderate agreement:** if $0.60 \leq \kappa < 0.75$, adjudicate disagreements; spot-check $\sim$10% of agreements; if PPA is low, tighten the "positive" criterion.
- **Weak agreement:** if $\kappa < 0.60$, pause gold build; run a 30–60 min calibration on 10–20 hard cases; refine rubric; relabel slice; recompute κ.
- **Prevalence paradox:** if κ is low but PABAK high (few positives), emphasize PPA; if PPA≥ 0.8, proceed with adjudication.
- **Systematic bias:** if McNemar $p < 0.05$ ($b \neq c$), update rubric to fix asymmetry; re-run a small batch to confirm $b \approx c$.

*Notes on construct difficulty.—*Ambiguity is not purely a taxonomy issue; it often depends on project-closed acceptance criteria (limits, units, referenced standards). Accordingly, we report κ/PPA alongside the adjudication log and publish the *exact prompt and taxonomy* used for labeling (Appendix 5, Listings 3–4).

Metrics

Overall model metrics are reported as *micro* averages (we pool all decisions across documents); per-subsystem bars are *macro* averages (we compute metrics per subsystem and report their unweighted mean). Fleiss' κ is reported descriptively as an agreement statistic.

*Note: structural checks (deterministic).—*Duplicate ReqIDs and cross-document traceability defects are computed by explicit algorithms over the parsed repository. They are *quality-control checks*, not learned predictions, so we do not score them as model metrics. We report raw counts in table 4.

Why these metrics ?— Accuracy can be misleading under class imbalance, so we emphasize Precision/Recall/F_1 for error trade-offs, *Specificity* and *Balanced Accuracy* (BAcc) to account for the negative class, and *MCC* as a prevalence-robust single-number summary. For multi-rater consistency (humans/models), we report Fleiss' κ (chance-corrected) alongside the mean per-item agreement $\bar{P}$ to make prevalence effects explicit. A plain-language glossary appears in Table 2.

Duplicate semantic *detection.—*We classify pairwise duplicates by cosine similarity with a fixed threshold (definition given earlier Sec. 2). Evaluation uses Precision, Recall, and F_1 for the positive (*duplicate*) class.

*Ambiguity (binary).—*Positive class = *Ambiguous = Yes*. For each model vs. gold we compute True Positives (TP), False Positives (FP), False Negatives (FN), True Negatives (TN) and derive all reported metrics from these counts.

*Precision, recall, F_1.—*Precision penalizes false alarms; recall penalizes misses; F_1 summarizes the trade-off.

$$P = \frac{\text{TP}}{\text{TP+FP}}, \qquad R = \frac{\text{TP}}{\text{TP+FN}}, \qquad F_1 = \frac{2PR}{P+R}. \quad (6)$$

*Specificity, Accuracy, Balanced Accuracy.—*Specificity complements recall on the negative class; accuracy is reported but prevalence-sensitive; Balanced Accuracy is more robust under imbalance.

$$\text{Spec} = \frac{\text{TN}}{\text{TN+FP}} \quad , \text{Acc} = \frac{\text{TP+TN}}{\text{TP+FP+FN+TN}} \quad , \text{BAcc} = \frac{1}{2}\big(R + \text{Spec}\big) \quad (7)$$

*Matthews correlation (MCC).—*A correlation over all four cells; stable under imbalance and recommended as a single-number summary.

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}. \quad (8)$$

Table 2: **Metric glossary (plain-language reference).**

Metric	What it captures / why we use it	Range / neutral
Precision (P)	Among predicted positives, share that are correct; penalizes false alarms.	$[0, 1]$ (higher is better)
Recall / TPR (R)	Among true positives, share we found; penalizes misses.	$[0, 1]$ (higher is better)
F_1 score	Harmonic mean of P and R; single number for the precision–recall trade-off.	$[0, 1]$ (higher is better)
Specificity / TNR	True-negative rate; complements recall; important with many negatives.	$[0, 1]$ (higher is better)
Balanced Accuracy (BAcc)	Mean of TPR and TNR; more robust than Accuracy under class imbalance.	$[0, 1]$; chance baseline ≈ 0.5
Accuracy	Overall fraction correct; can look high under imbalance; reported but interpreted cautiously.	$[0, 1]$; prevalence-sensitive
Matthews corr. (MCC)	Correlation over all TP/FP/FN/TN; robust single-number summary under imbalance.	$[-1, 1]$; 0 = chance
Mean agreement $\bar{P}$	Average per-item rater agreement; prevalence-sensitive; descriptive only (not chance-corrected).	$[0, 1]$ (higher = more raw agreement)
Fleiss' κ	Chance-corrected multi-rater agreement; summarizes rater consistency (per subsystem).	$[-1, 1]$; 0.4–0.6 moderate, 0.6–0.8 substantial

Testability (binary).—Positive class = *Non Testable* = *Y*. Metrics mirror Ambiguity (same formulas and interpretation), with both overall (micro) and per-subsystem (macro) reporting.

Inter-rater agreement (multi-rater; descriptive).—When aggregating several raters (gold + models or model subsets), we describe consistency and prevalence using mean per-item agreement $\bar{P}$ and Fleiss' κ (chance-corrected). Let items $i = 1, \ldots, N$, categories $j = 1, \ldots, K$, with n_{ij} raters assigning item i to category j, and $n_i = \sum_j n_{ij}$ (items may have different n_i if some ratings are missing).

Per-item and mean agreement.—

$$P_i = \frac{1}{n_i(n_i-1)} \sum_{j=1}^{K} n_{ij}(n_{ij}-1), \qquad \bar{P} = \frac{1}{N} \sum_{i=1}^{N} P_i. \quad (9)$$

Category prevalences and chance agreement.—

$$p_j = \frac{\sum_{i=1}^{N} n_{ij}}{\sum_{i=1}^{N} n_i}, \qquad P_e = \sum_{j=1}^{K} p_j^2. \quad (10)$$

Fleiss' κ (chance-corrected multi-rater agreement).—

$$\kappa_{\text{Fleiss}} = \frac{\bar{P} - P_e}{1 - P_e}. \quad (11)$$

These describe labeling consistency and prevalence; they are not used as scoring metrics.

4. Experiments and Results

We run the full pipeline on the 225–requirement corpus (Sec. 3): PDF parsing → requirement extraction (ReqIDs, text, metadata) → structural checks (duplicate ReqIDs, TRD→SRD traceability) → semantic flags (Ambiguity, Testability). Structural checks are deterministic and reported as counts/coverage; semantic flags are produced by five instruction-tuned LLMs under the same prompt settings and evaluated against the gold labels with the metrics in Table. 2. To reduce tuning bias, thresholds for duplicate *semantic* matching (cosine) are fixed once on a small dev slice and held constant. For space, we center the discussion on *Ambiguity*; the full *Testability* set appear in the Appendix (Fig. 9-10-11-13-14-15).

Structural Quality Check results.—We report performance-style summaries for deterministic repository checks; no model scoring applies. Parser-induced issues are separated from QC outcomes.

Table 3: **Structural QC — simple summary (counts; parser-caused issues separated).**

What we measured	Result
Duplicate ID collisions detected / total	4 / 4 (100%)
TRD→SRD parent links resolved (excluding parser errors[2])	100%
TRD→SRD links unresolved (due to parser errors)	2
Blocks parsed correctly / total	218 / 225 (96.9%)
Requirements with no cited parent ("orphans")	6
Cited parent missing in SRD ("dangling" refs)	3

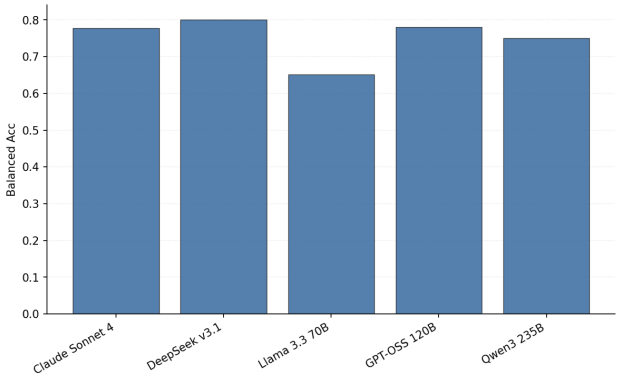


Figure 2: Ambiguity—Balanced Accuracy by model (micro, pooled over all requirements).

Notes. Duplicate-ID detection found all collisions. All cited-parent links were resolved except 2 cases caused by parser mis-segmentation. Parser missed 7 of 225 blocks (∼3.1%), hence 218 parsed.

Models evaluated.—We evaluate five instruction-tuned LLMs. Our selection *prioritizes open-weight models* that can run *on-premises* for reproducibility and data-governance, and includes one *hosted, closed-weight* model to broaden operating styles and licensing regimes. Concretely: DeepSeek v3.1 (**DeepSeek**), GPT-OSS 120B (**GPT-OSS**), Qwen3 235B Instruct (**Qwen**), Llama 3.3 70B Instruct (**Llama**), and Claude Sonnet 4 (**Claude**). All models use the same prompts and decoding settings (temperature 0.2, top-p 0.9, fixed max tokens) without per-model tuning; thresholds are fixed once on a small dev slice and held constant. Hereafter we refer to them by the bold shorthand in parentheses.

Reading guide. We structure the results as compact Research-Questions (RQ) and answer units: each RQ states a short takeaway, cites the evidence (figures/tables/metrics), and ends with a brief implication.

RQ1 — Which models perform best overall on ambiguity detection?—Balanced Accuracy (Fig. 2) provides a prevalence-robust view. The leading group (DeepSeek, GPT-OSS, Claude) sits in the high 0.7–0.8 range; Qwen is slightly lower; Llama lags (mid-0.6). These differences are consistent across documents and foreshadow distinct error profiles.
Matthews Correlation (Fig. 3) yields the same ordering and similar gaps: DeepSeek leads (MCC≈0.59), GPT-OSS and Claude follow (≈0.55), Qwen is close (≈0.50), and Llama trails (≈0.40). Because MCC is prevalence-robust and penalizes one-class behavior, this supports using Balanced Accuracy as the primary ranking metric.
Interpretation: on this corpus, the leading models (DeepSeek, GPT-OSS, Claude Sonnet 4) miss substantially fewer ambiguous requirements than Llama. For example, DeepSeek missed about 15 items versus about 60 for Llama. Because missed issues carry higher risk in our setting, we do not consider the mid-0.6 BAcc model suitable as a primary triager during requirement review.

RQ2 — What is the trade-off between missed issues and false alarms?—The FPR–FNR scatter (Fig. 4) makes the false-alarm vs. miss trade-off explicit. Llama is conservative (fewer flags, more misses); DeepSeek is liberal (more flags, fewer misses); GPT-OSS and Claude are closer to balanced.

[2] "Parser errors" = PDF block mis-segmentation.

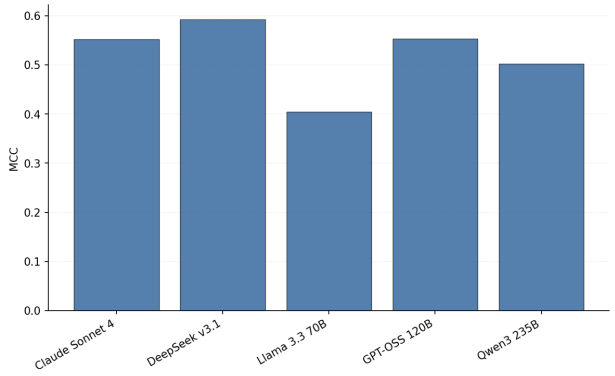


Figure 3: Ambiguity—MCC by model (micro, pooled over all requirements). Higher is better.

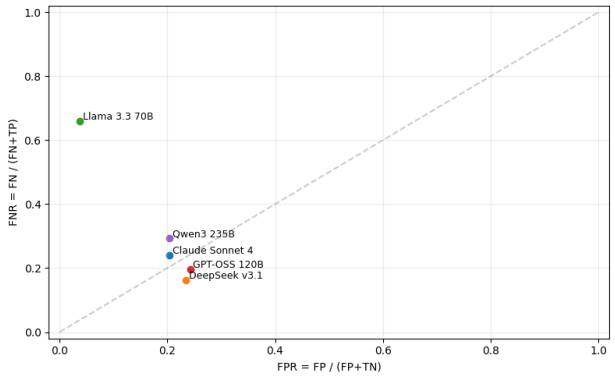


Figure 4: Ambiguity—error trade-off (FPR vs. FNR). Lower/left is better.

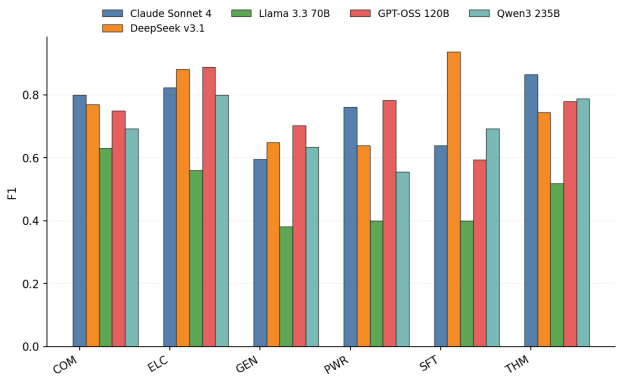


Figure 5: Ambiguity—F_1 by subsystem (per-subsystem scores; macro view).

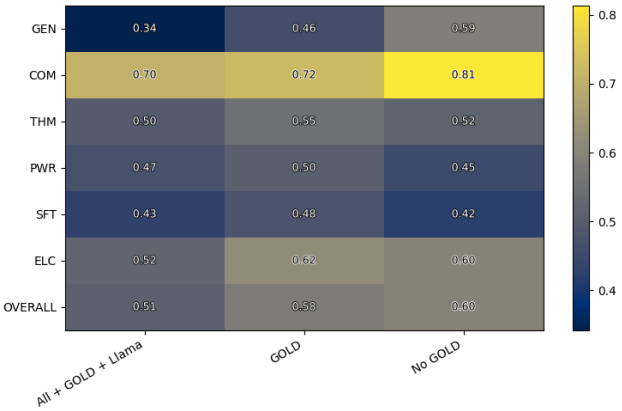


Figure 6: Ambiguity—Fleiss' κ by subsystem and rater pool (descriptive agreement).

Implication: this is a choice between misses and reviewer load. In our results, DeepSeek missed 15 items but raised 31 extra flags; Llama missed 60 but raised 5. During drafting we prefer the recall-leaning behavior to avoid misses; as documents stabilize, a conservative setting/model can be used to reduce flags.

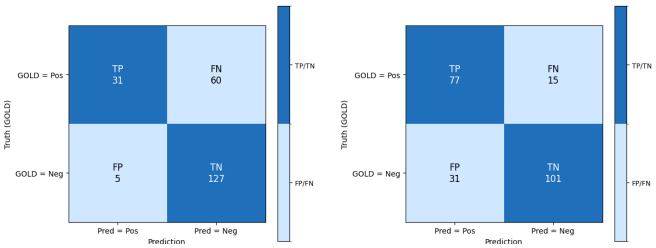
Reliability for our use case: we treat reliability as avoiding dangerous misses with stable behavior across subsystems; by that criterion, Llama-style conservative operation is not appropriate as the sole triager.

RQ3 — Where are the models strong or weak across subsystems?— Performance is not uniform across documents. The F_1 breakdown by subsystem (Fig. 5) shows *Power* and *Electrical* as more straightforward (scores around 0.9 across models), while *System* and *Thermal* exhibit larger spread and lower centers. *Software* is mixed: one model is notably lower while others remain strong, suggesting that domain phrasing and document style shape ambiguity cues. Precision and recall figures are in the Appendix 5, (Fig. 16-17).
Likely cause: Power/Electrical feature crisper local patterns (units, numeric bounds, interface tokens) with fewer cross-sentence dependencies; *System/Thermal* contain broader policy wording and multi-condition clauses whose meaning relies on surrounding prose, which increases ambiguity even for human readers.
implication: allocate more reviewer attention (and use recall-leaning settings) for *System/Thermal*; default settings typically suffice for *Power/Electrical*.

RQ4 — How much do raters agree on ambiguity judgments?—To contextualize model–gold disagreements, Fig. 6 reports Fleiss' κ by subsystem for several rater pools. *Communications* attains the highest agreement (around 0.7–0.8 depending on the pool), whereas *System* and *Software* are lower ($\sim$ 0.4–0.5). The "No GOLD" column shows that models alone can still align strongly in some domains, highlighting prevalence and rubric effects rather than pure noise.
Interpretation: agreement is moderate overall—good enough for triage but not for auto-acceptance. High agreement in *Communications* implies lower adjudication cost; lower agreement in *System/Software* justifies mandatory reviewer pass or stricter acceptance rules in those domains.

RQ5 — What do the errors look like in practice?—Contrasting confusion matrices (Fig. 7) make the behaviors concrete. Llama shows many *misses* (e.g., FN = 60 vs. FP = 5 in this corpus), while DeepSeek shows the opposite tendency (e.g., FN = 15 vs. FP = 31), consistent with the trade-off plot. This pattern also matches the Balanced-Accuracy ranking and the subsystem spread above.
Interpretation: the paired matrices confirm complementary tendencies: conservative models under-flag borderline ambiguity, liberal models over-flag it. In practice we mitigate this without retraining by adding light score calibration, using domain-specific flagging rules in *System/Thermal*, and prioritizing flags that touch safety-relevant tokens (limits, modes, inhibits).

Testability (summary).— Results broadly mirror *Ambiguity* but with a different operating trade–off. As shown by the FPR–FNR scatter in Fig. 8, **GPT-OSS 120B** and **DeepSeek**



(a) Llama 3.3 70B — more FN (conservative).

(b) DeepSeek — more FP (liberal).

Figure 7: Ambiguity—contrasting confusion matrices. Dark = correct (TP/TN), light = errors (FP/FN).

deliver the best balanced operating points (low–moderate FNR with comparatively low FPR), while **Llama 3.3 70B** is the weakest due to many missed flags (high FNR, low FPR). **Qwen3 235B** sits close to the leaders and **Claude Sonnet 4** skews conservative (higher FNR, very low FPR). By subsystem (Appendix, Fig. 10), *Electrical* and *Power* are easiest ($F_1 \approx 0.9$ across models), while *Thermal* is hardest ($\sim$0.75). Agreement analysis (Appendix, Fig. 11) shows moderate Fleiss' κ overall ($\sim$0.5–0.6), highest in *Communications* and lowest in *System/Thermal*. Paired confusion matrices (Appendix, Fig. 13) illustrate the contrast: GPT-OSS is more liberal (higher FP, lower FN) whereas Llama is more conservative (lower FP, higher FN).

Policy: favor recall-first thresholds during drafting (flag potential non-testables aggressively), then shift to precision-first near verification planning to avoid noise in test procedure generation.

Interpretation: testability rewards explicit acceptance conditions and coherent A/T/R/I tags, which favors structure-sensitive models. During drafting we run in a recall-first mode to surface potentially non-testable statements early; near verification planning we tighten the policy to avoid noise in procedure generation.

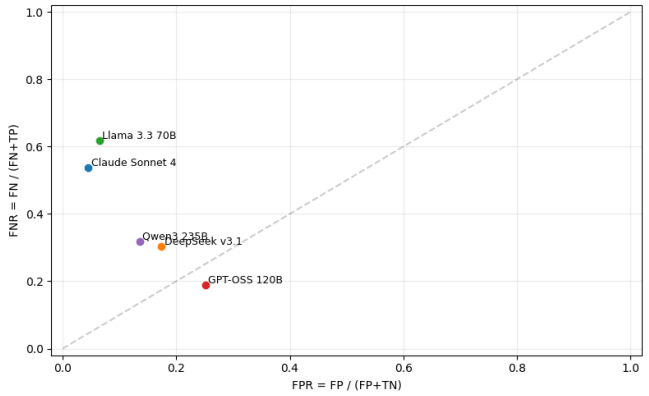


Figure 8: Testability—error trade-off (FPR vs. FNR).

5. Conclusion

We presented an end-to-end, human-in-the-loop pipeline that parses different requirement documents, enforces structural integrity, and audits semantics (ambiguity, testability). On a 225-requirement corpus with adjudicated gold labels, the deterministic stage reliably surfaced duplicate ReqIDs and SRD→TRD traceability gaps, while five instruction-tuned LLMs achieved balanced accuracy of about 0.65–0.80 with distinct false-positive/false-negative profiles and measurable variation across subsystems. These results support a hybrid *structure-first, semantics-second* design: rules provide reproducible, auditable corpus structure at scale, and LLMs assist reviewers by pre-screening statements for potential issues without replacing expert judgment. The pipeline is document-agnostic, versioned and auditable end-to-end, and integrates with RM/MBSE workflows.

Strengths include portable rule induction that adapts across document families, normalized storage for cross-document analytics, and interpretable outputs (confusion summaries, per-subsystem metrics, agreement analyses). The main limitations are evaluative scope and data quality: we used a synthetic corpus with limited adjudication (a more definitive study would involve larger, multi-expert labeling), and we observed occasional parser mis-segmentations (7 of 225, 3%) that can perturb reported scores; performance is also sensitive to domain phrasing. Immediate applications include pre-commit checks, review screening, supplier-document intake, change-impact via traces, and continuous coverage/redundancy audits, with outputs consumable by RM/MBSE tools. Future work will strengthen robustness across larger and more diverse documents and layouts, study sensitivity to prevalence and subsystem style, align prompts and rubrics with ECSS/Consultative Committee for Space Data Systems (CCSDS), develop a standards-aware model via domain-adaptive fine-tuning on ECSS/CCSDS guidelines [17] to enable direct conformance checks against the standards themselves, and normalize statements into templates that compile to temporal logics for tighter MBSE coupling. Overall, the pipeline advances automated quality assurance for aerospace requirements and opens a clear path from reliable parsing and analytics to formal verification.

Appendix

```
{
  "docs": [{
    "meta": { "name": <str> },
    "toc": [{
      "level": <int>,
      "title": "<str>",
      "page": <int>,
      "number": <int>,
      "full_title": "<str>"
    }],
    "samples": [{
      "page_index": <int>,
      "text": "..."
      "signals": {
        "id_token_candidates": [<str>],
        "id_contexts": [{
          "tok": <str>,
          "left_has_norm": <bool>,
          "right_has_norm": <bool>,
          "has_version": <bool>,
          "at_line_start": <bool>,
          "snippet":<str>
        }],
        "label_like_candidates": [<str>],
        "literal_separator_counts": {<char>:<int>},
        "shall_must_should_present": <bool>
      }
    }]
  }]
}
```

Listing 1: Peek – minimal example

```
{
  "schema": "regex_profile/v3",
  "id_core": "ASTRA-[A-Z]{3}-REQ-[\\d]{5}",
  "header_regex": "^ASTRA-[A-Z]{3}-REQ-[\\d]{5}(?:\\s*v:\\s*(\\d
      +(?:\\.\\d+)?))?(?:\\s*(?:(\\w+)))?",
  "inline_id_regex": "ASTRA-[A-Z]{3}-REQ-[\\d]{5}",

  "canonical_fields": [
    "statement", "rationale", "Verification Method",
    "Standard Origin Source", "ECSS DOORS Req ID", "Mother REQ"
  ],
```

```
  "field_label_map": {
    "statement": ["The", "shall"],
    "rationale": ["Rationale"],
    "Verification Method": ["Verification Method"],
    "Standard Origin Source": ["Standard Origin Source"],
    "ECSS DOORS Req ID": ["ECSS DOORS Req ID"],
    "Mother REQ": ["Mother REQ"]
  },

  "field_split_regex": "(?:^|\\s)(?:The|shall|must|should)(?=\\s*[
      a-zA-Z])",
  "statement_regex": "(?:shall|must|should)(?=\\s*[a-zA-Z])",

  "verification_tokens_regex": "^(?:A|T|I|R)(?:,\\s*(?:A|T|I|R))*$
      ",
  "page_noise_regexes": ["^\\d+$", "^\\s*$"],
  "external_ref_regexes": ["ECSS-[A-Z]{2}-[A-Z]{2}-\\d{2}C"]
}
```

Listing 2: regex profile – minimal example

```
ROLE: You are an expert auditor of space/ECSS-style requirements.
TASK: For each requirement block, decide Ambiguous (Y/N), Testable
      (Y/N),
      enumerate issues (taxonomy), and give a minimal fix. No
      external knowledge.

INPUT FIELDS:
  ReqID, STATEMENT, (optional) STANDARD, PARENT, PARENT_TEXT, DOC,
      SECTION, VERIF(A/T/R/I)

CONTEXT PRIORITY:
  1) If PARENT_TEXT closes acceptance (limits/tests), use it.
  2) If STATEMENT delegates to STANDARD and says "per <STD> limits
      /tests", use it.
  3) Else evaluate STATEMENT alone.

DELEGATION & PLACEHOLDERS:
  - Valid delegation to ICD/ICDR/ECSS/config/data pack; don't
      restate numbers.
  - If delegation clear but artifact missing in fields  "Cross-ref
      missing" (not Ambiguous=Y).
  - Explicit TBD/TBR/TBC: not ambiguous by itself  tag "
      Placeholder"; if pass/fail depends on it  Testable=N + "
      Verification gap".

DECISION LOGIC:
  Ambiguous=Y if acceptance cannot be determined without
      clarification:
     missing closing value and no delegation/managed placeholder;
     vague/undefined term with no measurable acceptance or
      reference;
     open-ended range with no closing reference.
  Ambiguous=N if closed by explicit limits, cited standard/test,
      PARENT_TEXT, clear artifact, or managed placeholder.
  Testable=Y if measurable predicate exists or pass/fail defined
      by standard/test/artifact (incl. binary R/I).
  Testable=N if qualitative with no binding reference or decisive
      value is TBD ("Verification gap").
  Multiple predicates: Ambiguous adopts worst case; Testable=Y if
      any predicate verifiable.

TAXONOMY (multi-label):
  Missing number; Missing unit; Vague term; Undefined term; Cross-
      ref missing;
  Contradiction; Range/open-ended; Verification gap; Environment
      unspecified; Placeholder (TBD/TBR/TBC).

OUTPUT: STRICT JSON only (schema ambig_test/v2), one item per
      block.
```

Listing 3: Ambiguity/Testability prompt

```
{
  "schema": "ambig_test/v2",
  "items": [{
    "id": "<ReqID>",
    "ambiguous": true,
    "testable": false,
    "issues": [
      {"type":"<taxonomy>", "quote":"<exact phrase>", "explain":"<
      why>"}
    ],
    "suggestion": "<minimal rewrite>",
    "rationale": "<24 lines; cite PARENT/STANDARD/SECTION when
      used>",
    "off_taxonomy_ambiguity": "<optional or empty>",
    "confidence": 0.0
  }]
}
```

Listing 4: Ambiguity/Testability output schema

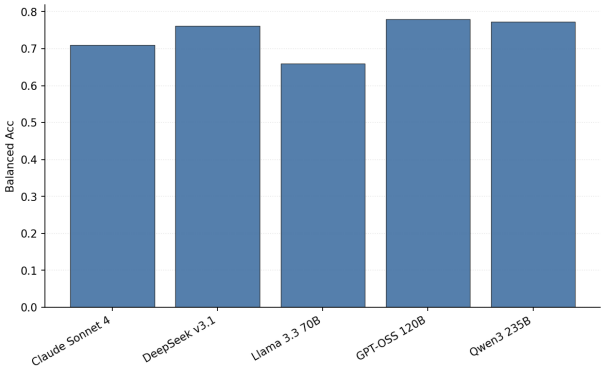


Figure 9: Testability—Balanced accuracy by model.

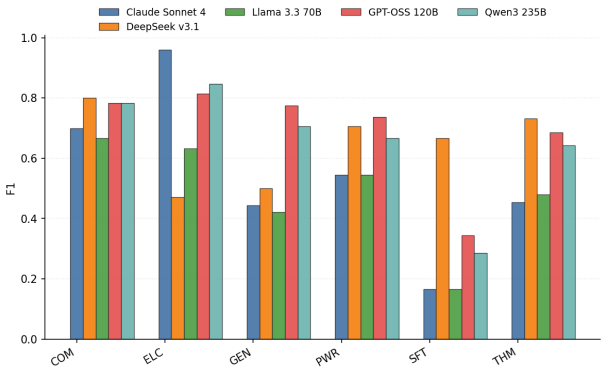


Figure 10: Testability—F_1 by subsystem (grouped by model).

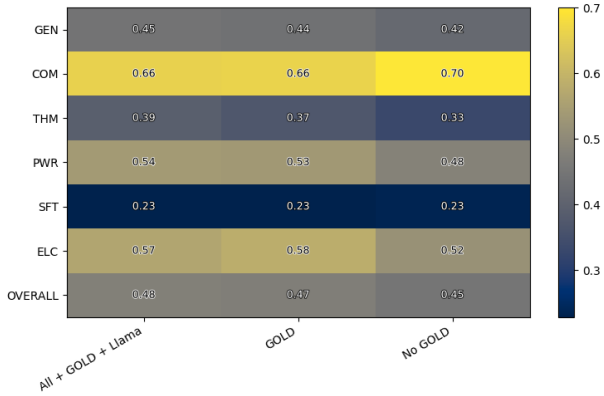
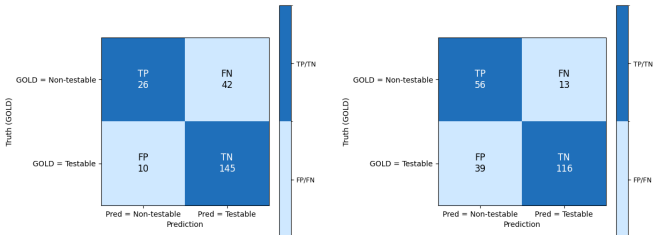


Figure 11: Testability—Fleiss' κ by subsystem and rater pool.



(a) Llama 3.3 70B — conservative (more FN).

(b) GPT-OSS 120 — liberal (more FP).

Figure 13: Testability—contrasting confusion matrices. Dark = correct (TP/TN), light = errors (FP/FN).

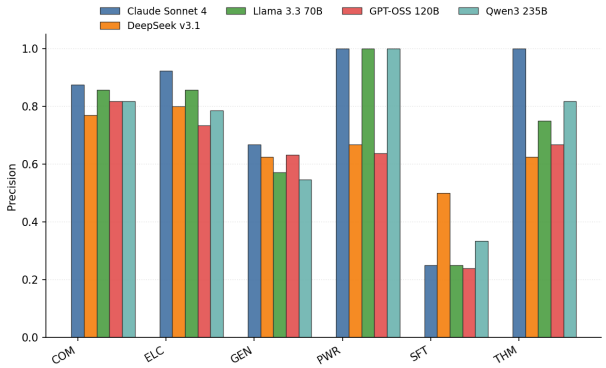


Figure 14: Testability- Precision by subsystem

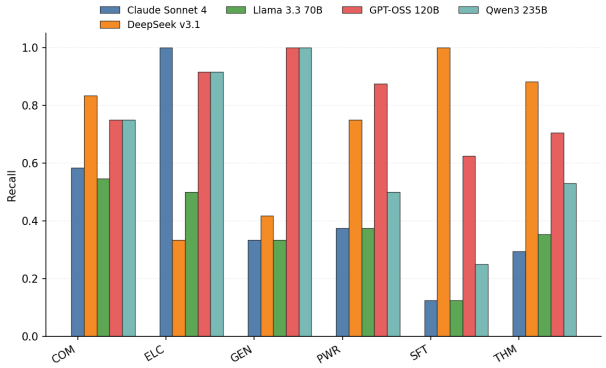


Figure 15: Testability- Recall by subsystem

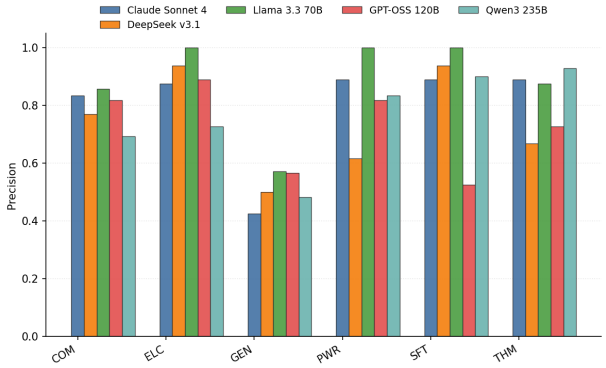


Figure 16: Ambiguity- Precision by subsystem

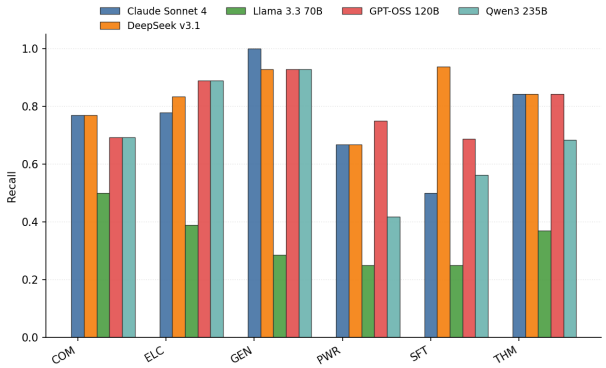


Figure 17: Ambiguity- Recall by subsystem

REFERENCES

[1] M. M. Kapoor and M. Mehta, "Importance of requirements analysis traceability to improve software quality and reduce cost and risk," 2010. [Online]. Available: https://hdl.handle.net/2014/44688

[2] IBM. (2025) Ibm engineering requirements management doors next. Accessed 2025-09-16. [Online]. Available: https://www.ibm.com/products/requirements-management

[3] Jama Software. (2025) Jama software- requirement engineering management. Accessed 2025-09-23. [Online]. Available: https://www.jamasoftware.com/

[4] Siemens Polarion. (2025) Polarion® application lifecycle management solutions. Accessed 2025-09-23. [Online]. Available: https://polarion.plm.automation.siemens.com/

[5] Eclipse Capella Project, "Capella requirements viewpoint / reqif importer," https://github.com/eclipse-capella/capella, 2025, allows ReqIF import from DOORS and other RM tools; accessed 2025-09-16.

[6] Object Management Group, "Requirements interchange format (reqif), version 1.2," Object Management Group (OMG), Needham, MA, USA, Tech. Rep. formal/2016-07-01, Jul. 2016. [Online]. Available: https://www.omg.org/spec/ReqIF/1.2/PDF

[7] A. Tikayat Ray and B. F. Cole, "Classification of aerospace requirements using bert," *Aerospace*, vol. 10, no. 3, p. 279, 2023. [Online]. Available: https://www.mdpi.com/2226-4310/10/3/279

[8] Y. Liu *et al.*, "Llm-acnc: Aerospace requirement texts knowledge graph construction with llms," *Aerospace*, vol. 12, no. 6, p. 463, 2025, accessed 2025-09-16. [Online]. Available: https://www.mdpi.com/2226-4310/12/6/463

[9] A. Bajceta *et al.*, "Using nlp tools to detect ambiguities in system requirements: A comparison study," in *Proceedings of the 2nd International Workshop on NLP for Requirements Engineering (NLP4RE)*, ser. CEUR Workshop Proceedings, vol. 3122, 2021. [Online]. Available: https://ceur-ws.org/Vol-3122/NLP4RE-paper-3.pdf

[10] R. Degiovanni *et al.*, "A genetic algorithm for goal-conflict identification," in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3238147.3238220

[11] J. Mucha *et al.*, "A systematic literature review of pre-requirements traceability," *Requirements Engineering*, 2024. [Online]. Available: https://link.springer.com/article/10.1007/s00766-023-00412-z

[12] Q. Chen, A. Banerjee, Çağatay Demiralp, G. Durrett, and I. Dillig, "Data extraction via semantic regular expression synthesis," 2023. [Online]. Available: https://arxiv.org/abs/2305.10401

[13] A. Mavridou, H. Bourbouh, P.-L. Garoche, D. Giannakopoulou, T. Pressburger, and J. Schumann, "Bridging the gap between requirements and model analysis: Evaluation on ten cyber-physical challenge problems," NASA Technical Reports Server (NTRS), Document ID 20200002241, 2020, accessed 2025-09-27. [Online]. Available: https://ntrs.nasa.gov/api/citations/20200002241/downloads/20200002241.pdf

[14] Beijing Academy of Artificial Intelligence, "Baai/bge-m3 — model card," 2024, multi-function, multi-lingual, multi-granularity embedding model. [Online]. Available: https://huggingface.co/BAAI/bge-m3

[15] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," *arXiv:2402.03216*, 2024. [Online]. Available: https://arxiv.org/abs/2402.03216

[16] R. Artstein and M. Poesio, "Inter-coder agreement for computational linguistics," *Computational Linguistics*, vol. 34, no. 4, pp. 555–596, 12 2008. [Online]. Available: https://doi.org/10.1162/coli.07-034-R2

[17] Consultative Committee for Space Data Systems (CCSDS). (2025) Ccsds blue books (standards). Accessed 2025-10-03. [Online]. Available: https://ccsds.org/publications/bluebooks/

BIOGRAPHY

Imed Eddine Ben Slimene received his National Engineering Diploma from the National Institute of Applied Science and Technology (INSAT), Tunisia. He joined the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR), Oberpfaffenhofen, in 2023 as a Research Engineer, where he develops robotics software frameworks and middleware for on-orbit-servicing missions. He is currently investigating methods to optimize and improve requirements engineering for space projects.

Maximo A. Roa received his doctoral degree in robotics in 2009 from Universitat Politecnica de Catalunya (UPC), and the Project Management Professional (PMP) Certification in 2016. He joined the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR) in 2010 as Senior Research Scientist, leading the group on Robotic Planning and Manipulation. He is an IEEE Senior Member, and member of ASME and IAU.

Daniel Leidner received his degree in communications engineering in 2010, and his M.Sc. degree in information technology in 2011 with distinction from the Mannheim University of Applied Sciences, Mannheim, Germany. In 2017 he received the Ph.D. degree in artificial intelligence from the University of Bremen, Bremen, Germany. His dissertation was honored with the Georges Giralt PhD Award as well as the Helmholtz Doctoral Prize. He is a Professor at the Institute of Artificial Intelligence at the University of Bremen and leads the department of Autonomy and Teleoperation at the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Wessling, Germany. As Co-Investigator of the Surface Avatar experiments, he investigates artificial intelligence in the context of astronaut-robot collaboration.

Martin Stelzer studied computer science at FH Ingolstadt and the University of Hagen and received his M.Sc. Degree in 2012. Since 2007 he has been working at the German Aerospace Center in the field of onboard software frameworks and was involved in the space projects ROKVISS, Kontur-2, and EROSS.