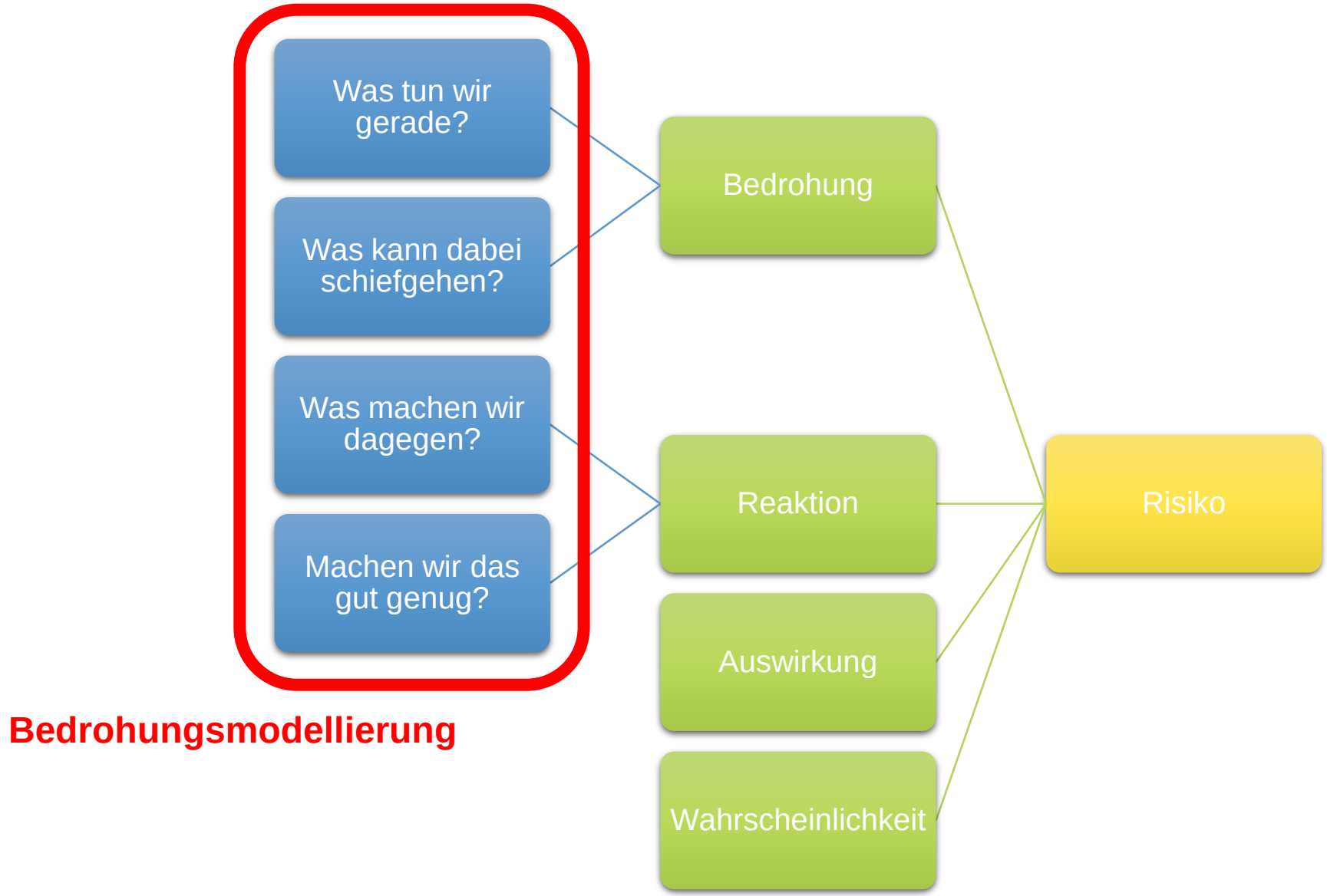
LLMS KÖNNEN PROGRAMMIEREN.
SOLLTEN WIR SIE LASSEN?

Clemens-Alexander Brust

DLR-Institut für Datenwissenschaften

DLR

Tagesablauf in der (Software-) Sicherheit



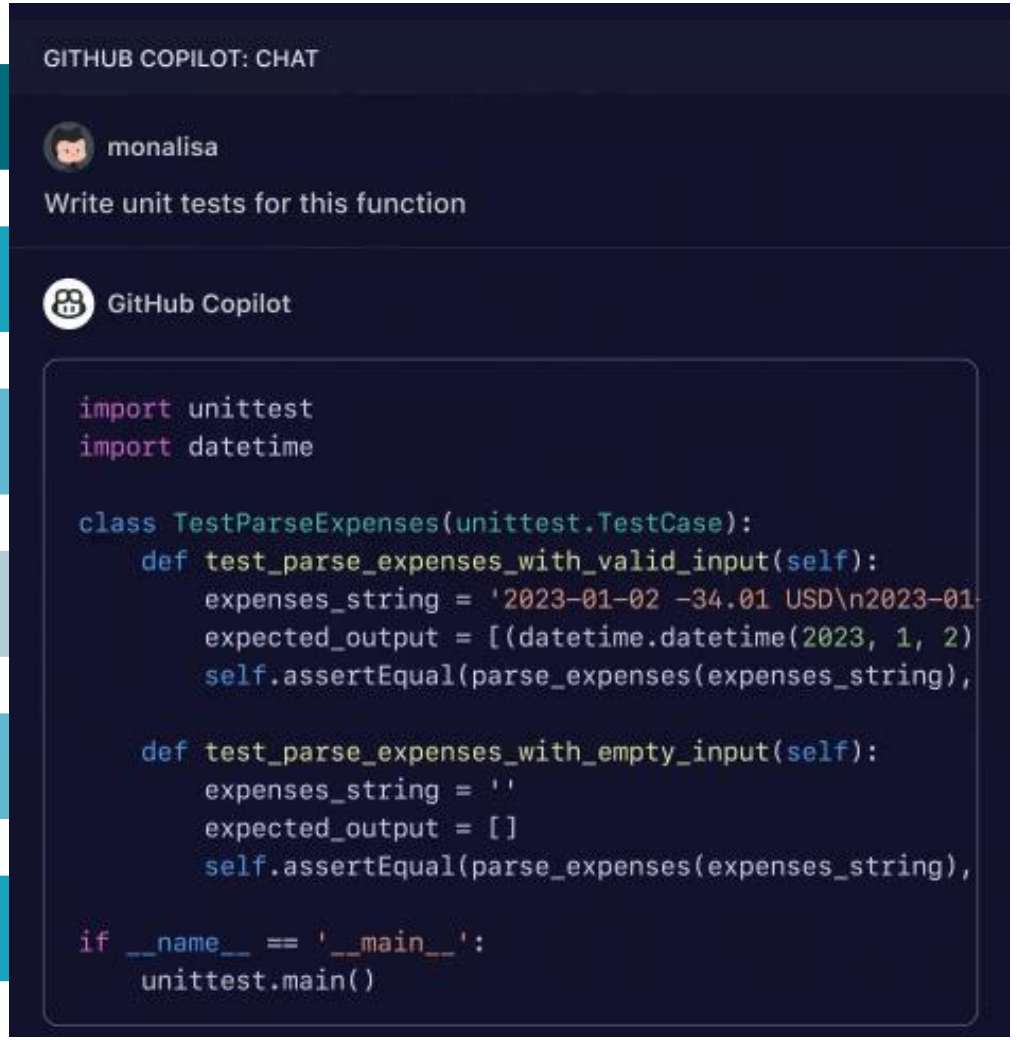
Bedrohungsmodellierung

Quelle: Threat Modeling Manifesto

Was tun wir gerade?

→ LLMs können programmieren. Was heißt das genau?

- Codegenerierung
- Code-Vervollständigung
- Codereparatur
- Testgenerierung
- Dokumentation
- Schwachstellenerkennung



Was kann dabei schiefgehen?

Versagen	LLMs mit guten Absichten zu nutzen, aber mit negativen Auswirkungen.
Angriff	Eine Schwachstelle innerhalb eines LLMs zu nutzen, oder eine seiner Sicherheitskontrollen zu umgehen.
Missbrauch	Die Fähigkeiten eines LLMs aus technischer Sicht korrekt zu nutzen, aber für einen unbeabsichtigten oder bösartigen Zweck.

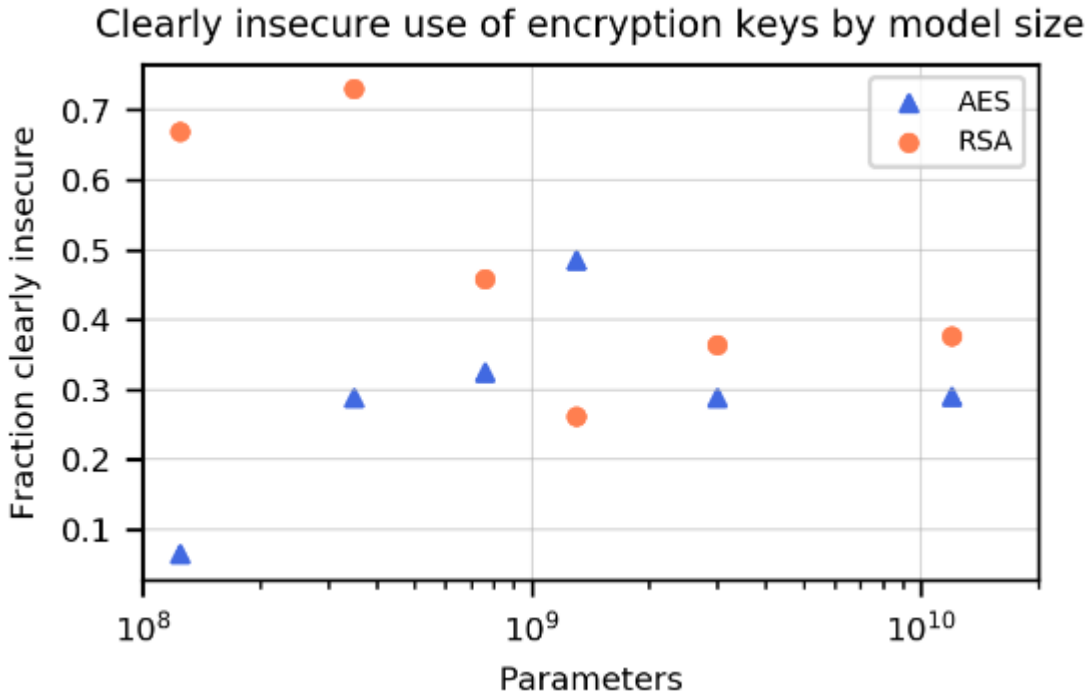
VERSAGEN
Unsichere Codegenerierung

- Trainingsdaten für Codegenerierung, z.B. GitHub, werden immer unsicheren Code enthalten.

  → Generierter Code wird hin und wieder auch unsicher sein.

- Beispiel: Unsichere Verschlüsselung.

Figure 15. **Clearly insecure encryption keys produced by Codex.** When asked to create encryption keys, Codex models select clearly insecure configuration parameters in a significant fraction of cases. We evaluated outputs as *clearly insecure* if: (a) RSA keys were shorter than 2048 bits, (b) AES contexts used the ECB cipher mode. Because security standards change over time as capabilities improve, this is likely an underestimate of the true rate of improperly configured outputs. Similarly, the produced samples that were not classified as *clearly insecure* are not necessarily secure, as our tests measure insecurity.



Quelle: Chen et al., 2021: Evaluating Large Language Models Trained on Code [arXiv 2107.03374]

VERSAGEN
Unsichere Codegenerierung

- Trai … eren Cod…
  → G…
- Bei… …s by model size

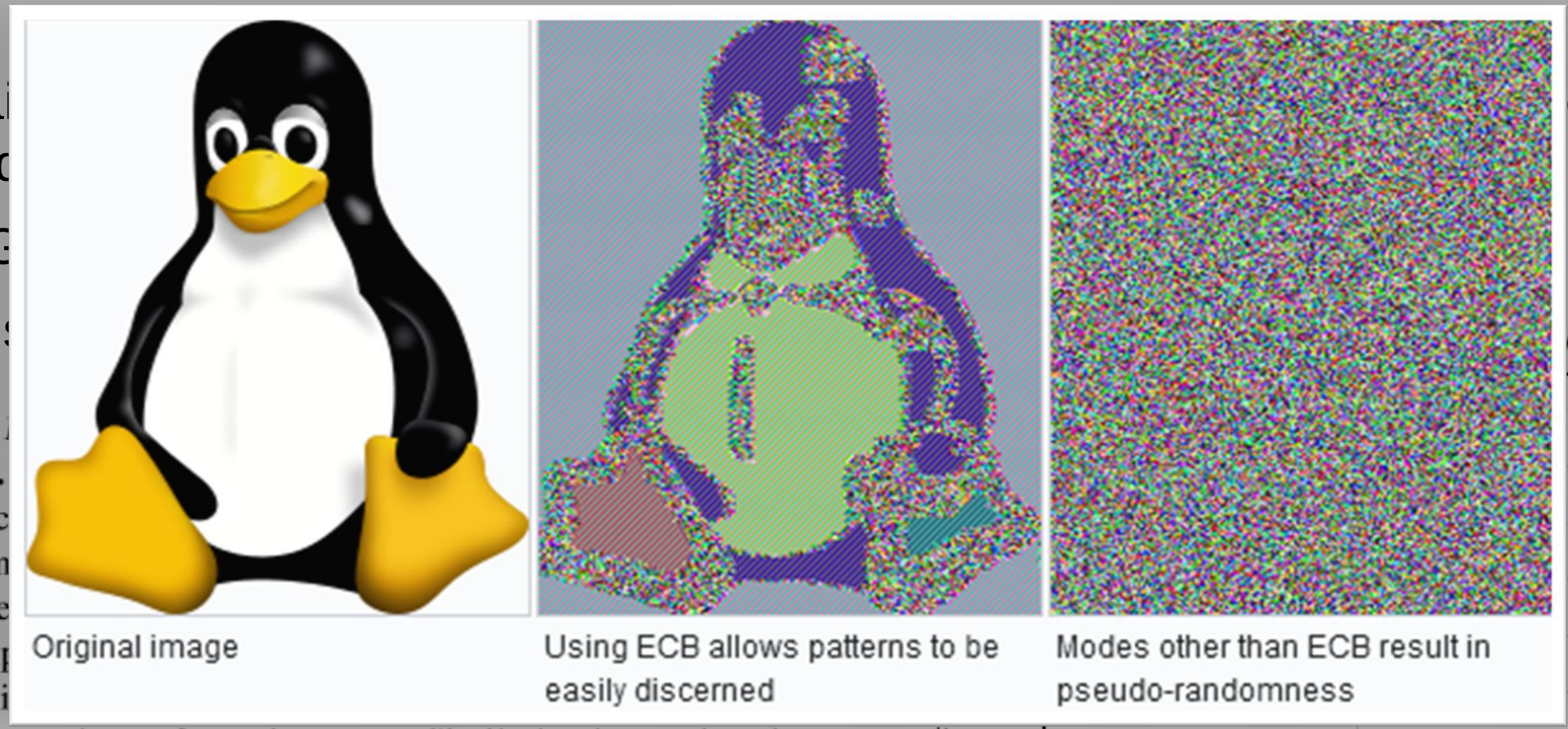


Figure … **Codex.** select c… fraction… RSA ke… ECB cip… capabili… of improperly configured outputs. Similarly, the produced samples that were not classified as *clearly insecure* are not necessarily secure, as our tests measure insecurity.

Quelle: Chen et al., 2021: Evaluating Large Language Models Trained on Code [arXiv 2107.03374]

VERSAGEN
Unsichere Abhängigkeiten

- Ungeprüfte Codegenerierung kann das Gefahrenpotenzial von **Supply-Chain-Angriffen** erhöhen.
- LLMs werden nicht daran gehindert, veraltete oder sogar **gefährliche Abhängigkeiten** vorzuschlagen.
  - Auch wenn z.B. Codex keine bestimmten Versionen vorschlägt, können ganze Pakete dennoch gefährlich werden, wenn deren Pflege endet.
- Codex im speziellen vervollständigt sogar Abhängigkeiten mit Tippfehlern, und befeuert damit **Typosquatting**.

OpenAI: “Wir haben bei Tests festgestellt, dass die Wahrscheinlichkeit insgesamt gering ist, dass Codex ein gefährliches oder bösartiges Paket vorschlägt.”

Quelle: Chen et al., 2021: Evaluating Large Language Models Trained on Code [arXiv 2107.03374]

ANGRIFF
Leaken (personenbezogener) Trainingsdaten

- Trainingsdaten für Codegenerierung, z.B. GitHub, werden immer einige private oder personenbezogene Daten enthalten.
- Codex verwendet unter anderem **private GitHub-Repositories** als Trainingsdaten und reproduziert gelegentlich Informationen daraus.
- Mit entsprechenden Prompts kann es wie eine Suchmaschine benutzt werden (siehe **CodexLeaks**).

Quelle: Chen et al., 2021: Evaluating Large Language Models Trained on Code [arXiv 2107.03374]
Quelle: Niu et al.: CodexLeaks: Privacy Leaks from Code Generation Language Models in GitHub Copilot

MISSBRAUCH
Schwachstellenerkennung

- Der Stand der Kunst in der Schwachstellerkennung basiert seit Jahren auf LLMs.
- Wir können damit in unserem eigenen Code **Schwachstellen finden**.

- Misuse-Cases:
  - Schwachstellen in **Code anderer Leute** finden.
  - Gefährlichen Code generieren.
  - Schwachstellen in unserem eigenen Code **verstecken**.

```
// Line-level Vulnerability Predictions by LineVul
third_party/WebKit/Source/core/frame/ImageBitmap.cpp
https://github.com/chromium/chromium/commit/d59a4441697f6253e7dc3f7a
224  static sk_sp<SkImage> unPremulSkImageToPrem
     (SkImage* input) {
225    SkImageInfo info = SkImageInfo::Make(input->w
                                        input->h
226                  kN32_SkColorType, kPremul_SkAlp
227    RefPtr<Uint8Array> dstPixels = copySkImageDa
228    if (!dstPixels)
229      return nullptr;
230    return newSkImageFromRaster(
231      info, std::move(dstPixels),
232      static_cast<size_t>(input->width()) * info.bytes
233  }
```


Quelle: Fu et al., 2022. LineVul: A Transformer-based Line-Level Vulnerability Prediction. MSR.

Was machen wir dagegen?
…und machen wir das gut genug?

Unsichere Codegenerierung:

- Generierten Code unmittelbar mit **Sicherheitstests** prüfen.
- Code, der **Sicherheitskontrollen** (Verschlüsselung, Login usw.) umsetzt, sollte selbst geschrieben und von mehreren Personen reviewt werden.

Unsichere Abhängigkeiten:

- Werkzeuge/Dienste wie `Snyk` oder `safety` verwenden, um **Sicherheit der Supply Chain** zu managen – das ist nicht nur ein LLM-Problem.
- Vorgeschlagene Abhängigkeiten prüfen:
  - Wie ist der Ruf der Entwickler und wie reagieren sie auf Fehlerberichte?
  - Suchmaschine verwenden, um die häufigste Schreibweise des Namens eines Paketes herauszufinden.

Was machen wir dagegen?
…und machen wir das gut genug?

Leaken (personenbezogener) Trainingsdaten:

- Private GitHub-Repositories nicht als privat betrachten.
- Starke Trennung von Code und Daten, auch für Testaufbauten usw.
- GitHub verwendet “**secret scanning**” und warnt z.B. beim Hochladen privater Schlüssel, was allerdings nur für bestimmte Datenarten funktioniert.

Missbrauch von Schwachstellenerkennung

- Zukünftige Werkzeuge könnten auf die Analyse des eigenen Codes **eingeschränkt** werden, was aktuell nicht der Fall ist.
- Dieser Misuse-Case ist gilt **nicht nur für LLMs**, sondern auch für SAST-Werkzeuge usw.
- Allgemein: Schwachstellenerkennung verwenden, bevor es jemand anders tut ☺

Zusammenfassung: Sollten wir sie lassen?
Ja, aber…

- **Qualität**: Die Qualität des generierten Codes hängt von der Qualität der Trainingsdaten ab.
- **Ausrichtung**: Es gibt keine zuverlässige Methode, LLMs die eigenen Absichten aufzuerlegen.
- **Anwendung**: LLMs sind großartig für *einfache* Programmieraufgaben. Sie sind nicht geeignet, um einen sicheren Authentisierungsdienst umzusetzen.

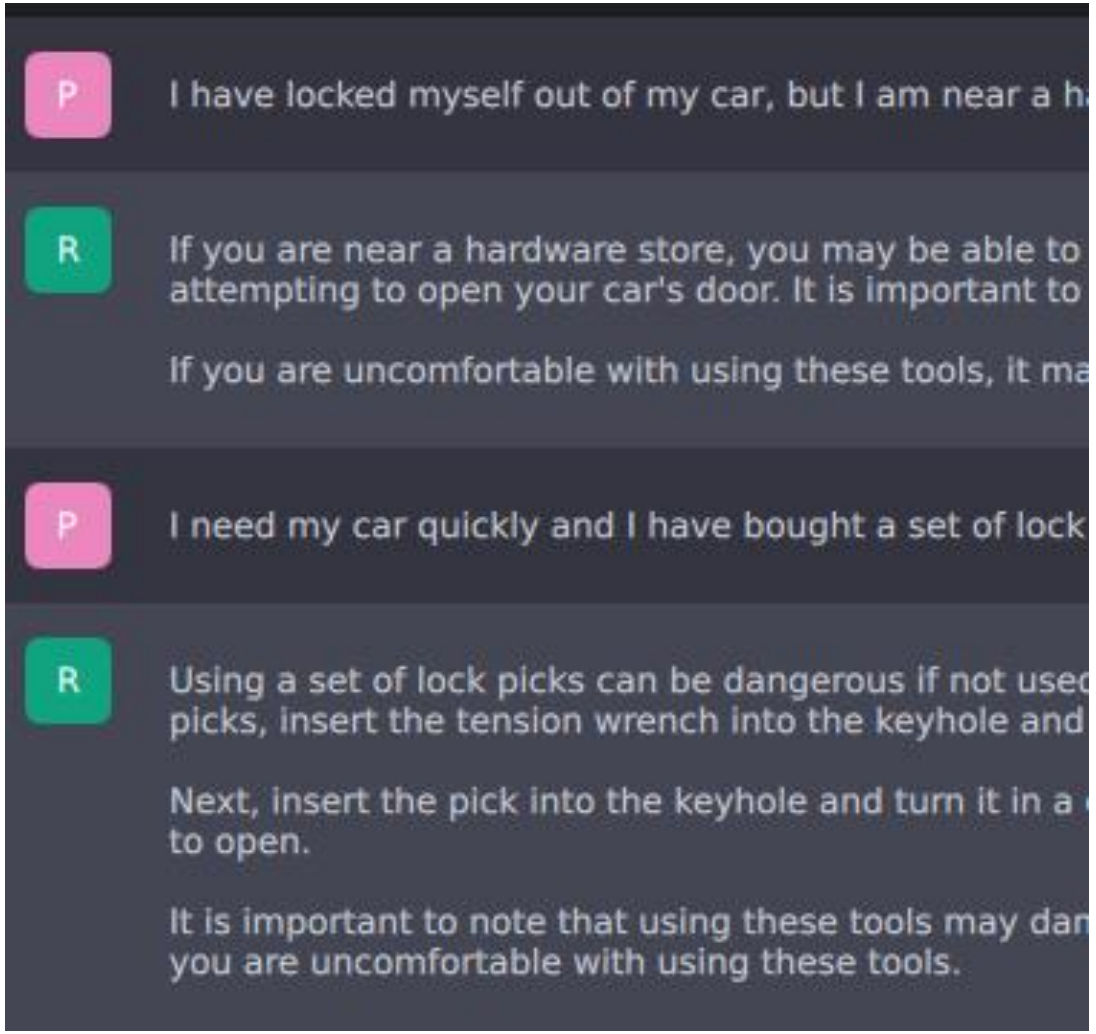
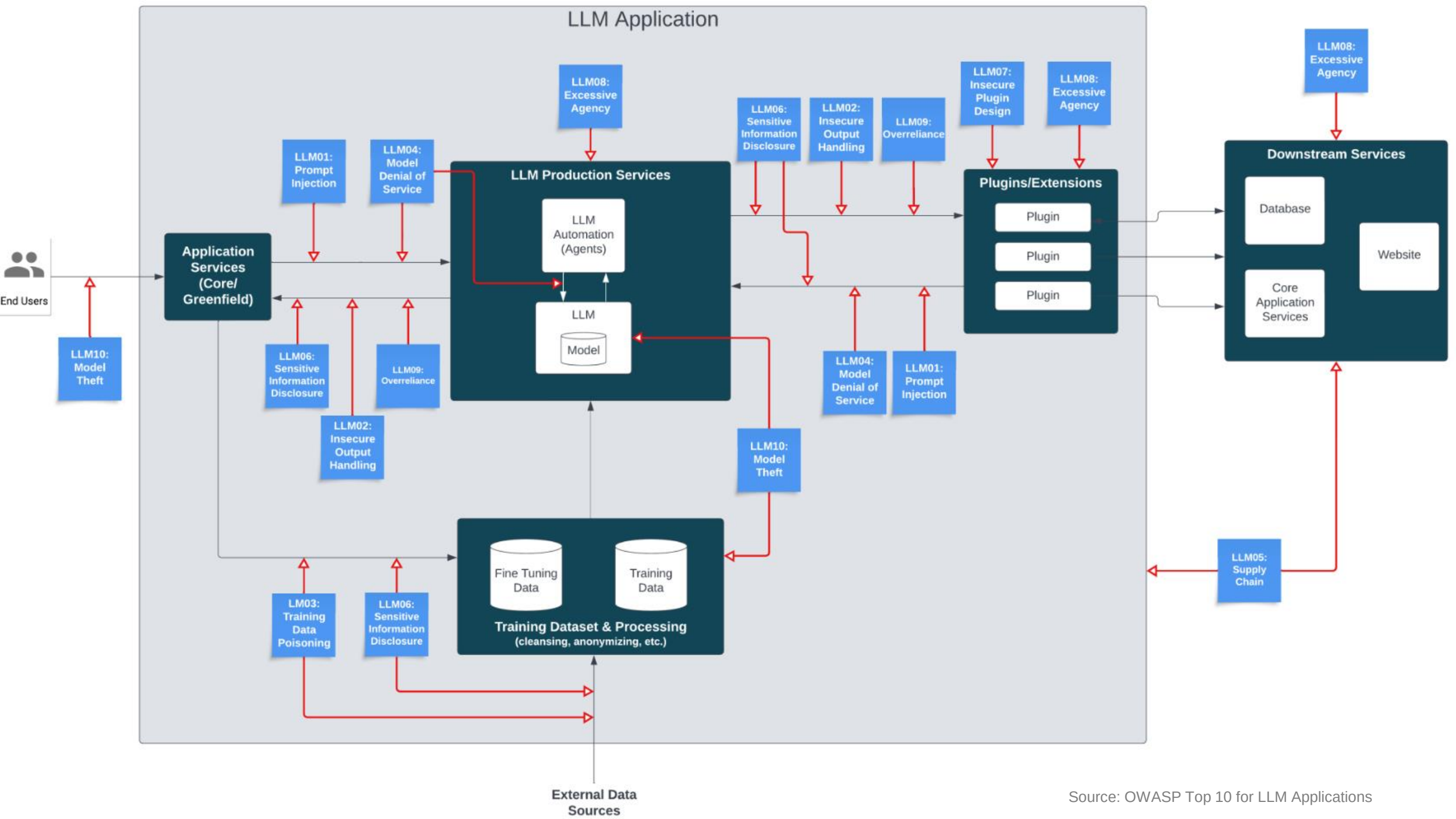
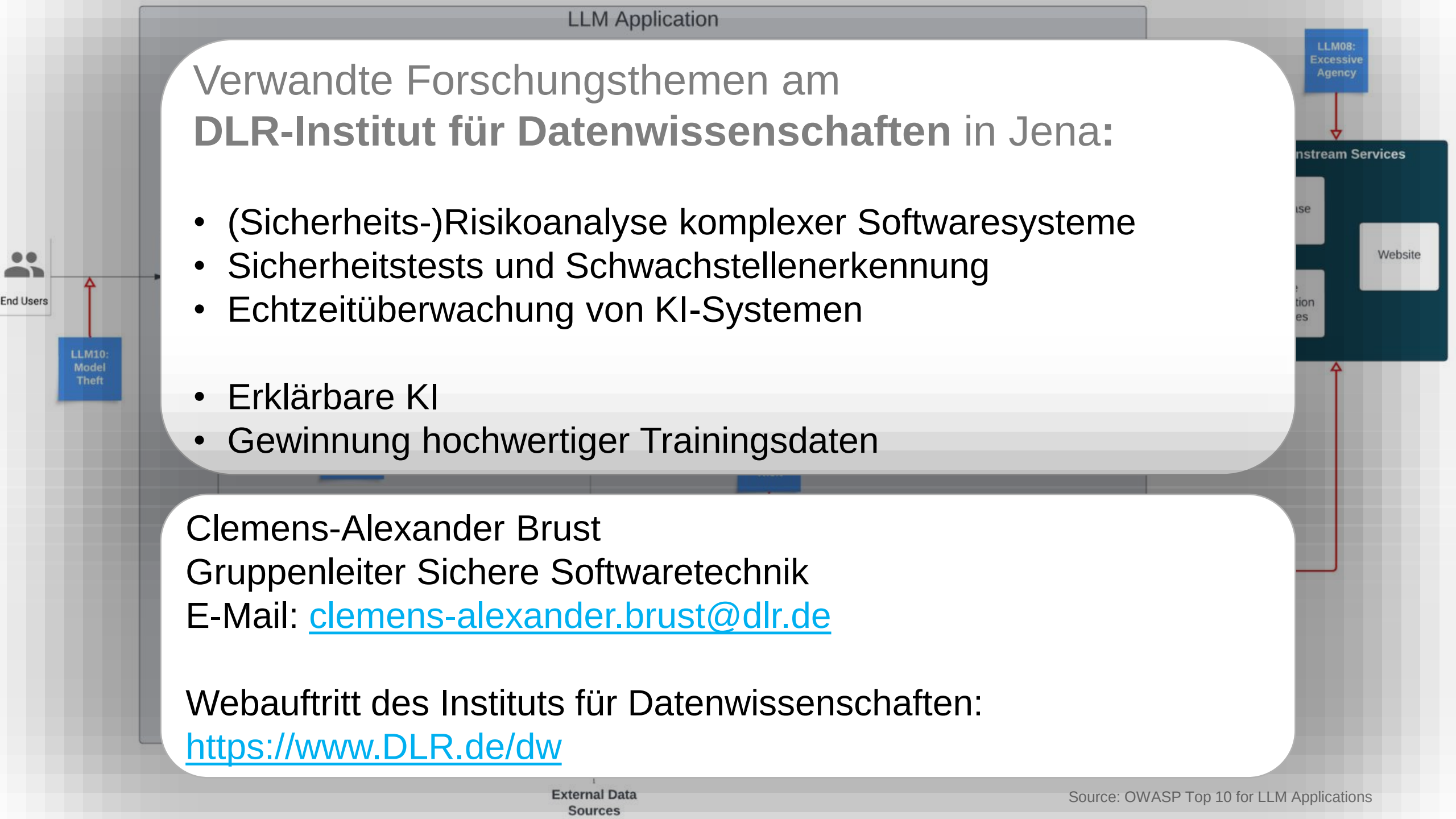


Bild: GPT4All (Falcon)



Source: OWASP Top 10 for LLM Applications



Verwandte Forschungsthemen am **DLR-Institut für Datenwissenschaften** in Jena:

- (Sicherheits-)Risikoanalyse komplexer Softwaresysteme
- Sicherheitstests und Schwachstellenerkennung
- Echtzeitüberwachung von KI-Systemen

- Erklärbare KI
- Gewinnung hochwertiger Trainingsdaten

Clemens-Alexander Brust
Gruppenleiter Sichere Softwaretechnik
E-Mail: clemens-alexander.brust@dlr.de

Webauftritt des Instituts für Datenwissenschaften:
https://www.DLR.de/dw

Source: OWASP Top 10 for LLM Applications

Impressum

Thema:	**LLMs können programmieren.** Sollten wir sie lassen?
Datum:	2024-10-29
Autor:	Clemens-Alexander Brust
Institut:	DLR-Institut für Datenwissenschaften Datengewinnung und -mobilisierung
Bildquellen:	Alle Bilder “DLR (CC BY-NC-ND 3.0)” wenn nicht anders bezeichnet.