

Replikation und Reproduktion von CHESS

Data Mining and Machine Learning

Felix Feist, Erik Jonas Hartnick und Stephan Mitte

31. März 2025

1 Einleitung

Diese Arbeit repliziert den Artikel *CHESS: Contextual Harnessing for Efficient SQL Synthesis* von Talaei u. a. [9], welcher sich mit der Generierung von SQL Anfragen aus Fragen in natürlicher Sprache beschäftigt. Im Bereich des Machine Learning existieren bereits viele Arbeiten zu solchen Text-To-SQL Problemen [2, 11, 12]. Durch den immensen Zuwachs an Daten und die steigende Komplexität von Datenbanksystemen können viele dieser Ansätze, selbst mit viel Rechenkapazität, der Problemstellung nicht mehr gerecht werden. In [5] wurde ein Unterschied in der *Execution accuracy* von 40% zwischen generierten und händisch geschriebenen SQL-Anfragen auf komplexen Datenbanken ermittelt. Aller Voraussicht nach wird die Menge an komplexen Daten in den nächsten Jahren nicht abnehmen. Die replizierte Arbeit stellt einen effektiveren Ansatz zur Lösung dieses Problems vor.

Neue Text-To-SQL Lösungen müssen den folgenden Herausforderungen gerecht werden [vgl. 9]:

- (i) Großer Umfang der Daten
- (ii) Generieren der SQL-Anfragen für große Datenbank-Schemata
- (iii) Verifikation der generierten Anfragen
- (iv) Mehrdeutigkeiten in den Fragestellungen auflösen

Zur Lösung dieser Probleme wird in [9] ein neues Multi-Agent-Framework präsentiert, welches mithilfe von Large Language Models (LLMs) eine effektivere Generierung der SQL-Anfragen unter den genannten Herausforderungen ermöglichen soll. Zudem wird die Adaptivität dieses Ansatzes hervorgehoben, in Anwendungsszenarien mit unterschiedlichen Budgets und Rechenkapazität der LLMs, verschiedene Systemkonfigurationen zu ermöglichen. Dadurch sollen SQL-Anfragen auch mit limitierten Ressourcen auf komplexen Datenbanken generiert werden können.

Wir sehen in diesem Ansatz viel Potenzial, neuen Herausforderungen mit aktuellen Technologien zu begegnen. Die experimentellen Untersuchungen in [9] erzielten wettbewerbsfähige Ergebnisse bei anerkannten Text-To-SQL Benchmarks. Diese Ergebnisse wollen wir zunächst auf einem kleinen Subsampled Development Set (SDS) basierend auf dem BIRD-Datensatz [5], später auf dem Spider-Datensatz [10] reproduzieren.

Aufgrund monetärer Beschränkungen ist es uns nicht möglich, Experimente mit proprietären Modellen, wie sie in [9] genutzt wurden, zu replizieren. Dies führt dazu, dass wir den vollständigen

BIRD-Datensatz und den synthetisch erzeugten industrial-scale Datensatz aus [9] nicht testen können. Außerdem entfallen für unsere Experimente die Tests mit Konfigurationen für leistungsstarke LLMs. Daher möchten wir in dieser Arbeit einen qualitativen Vergleich der Ergebnisse aus [9] mit Open-Source Modellen durchführen, wobei wir hoffen, ähnliche Ergebnisse zu erzielen.

Darüber hinaus testen wir verschiedene Konfigurationen des Candidate Generators, einem der vier Agenten des Frameworks. Dadurch werden wir neue Einblicke über die Konfigurationsparameter der genutzten LLMs erhalten.

Nachfolgend beschreiben wir in Abschnitt 2 ausführlich den Umfang unserer Replikation und formulieren die untersuchten Behauptungen. In Abschnitt 3 gehen wir detailliert auf den Aufbau der Experimente und die genutzten Ressourcen ein. Die Ergebnisse präsentieren wir schließlich in Abschnitt 4 und diskutieren diese dann in Abschnitt 5.

2 Umfang der Replikation/Reproduktion

Zunächst erklären wir das Multi-Agent-Framework und stellen die experimentellen Ergebnisse aus [9] vor. Danach motivieren wir die Auswahl der Experimente für unsere Replikation und formulieren schließlich die konkreten Behauptungen, welche wir experimentell untersuchen werden.

2.1 Experimente der Autoren

In [9] wird ein neuer Lösungsansatz für Text-To-SQL Probleme präsentiert, welcher insbesondere für komplexe Datenbanken mit vielen Tabellen und Spalten entwickelt wurde. Dieser Ansatz versucht das Problem mit einem Multi-Agent-Framework und vier spezialisierten Agenten zu lösen. Die vier Agenten übernehmen dabei folgende Aufgaben:

1. **Information Retriever (IR):**

- Analysieren der gestellten Frage des Nutzers mit LLM-Anfragen
- Ermitteln der relevanten Entitäten und deren Kontext (DB-Schema, DB-Katalog)
- Syntaktische- und semantische Ähnlichkeiten mit Lokalitätssensitivem Hashing (LSH) bestimmen

2. **Schema Selector (SS):**

- Reduzieren von großen DB-Schemata in leichter zu erfassende Sub-Schemata
- Auswählen benötigter Tabellen und Spalten mit LLM-Anfragen

3. **Candidate Generator (CG):** Generieren, ausführen und überarbeiten von Kandidaten mit LLM-Anfragen

4. **Unit Tester (UT):**

- Bewerten der Kandidaten mithilfe generierter Unit-Tests
- Auswählen der finalen SQL-Anfrage

Methode \ Datensätze	BIRD test	BIRD dev	Spider
Beste Methode (Stand 03/2025)	77.14 [5]	75.36 [5]	91.2 [10]
$CHES_{(IR,CG,UT)}$	68.31	71.10	-
$CHES_{(IR,SS,CG)}$ + proprietär	66.69	65.00	87.2
$CHES_{(IR,SS,CG)}$ + Open LLMs	-	61.5	-
$\Delta EX = EX_{\text{proprietär}} - EX_{\text{OpenLLM}}$	-	3.5	-

Tabelle 1: *Execution accuracy* verschiedener Framework-Konfigurationen auf unterschiedlichen Benchmarks [siehe 9, Tabelle 1,2,3]

Text-To-SQL Modelle müssen in realen Anwendungsszenarien mit unterschiedlichen Ressourcenkapazitäten zurecht kommen. Dies schränkt die Nutzung einiger Modelle ein, da diese leistungsstarke LLMs benötigen. Aus diesem Grund bietet das Multi-Agent-Framework verschiedene Konfigurationsmöglichkeiten an. Für die experimentellen Untersuchungen werden in [9] die folgenden zwei Varianten des Frameworks abgeleitet:

- $CHES_{(IR,CG,UT)}$ - Konfiguriert für maximale Genauigkeit (mehr Rechenkapazität und stärkere LLMs)
- $CHES_{(IR,SS,CG)}$ - Konfiguriert für maximale Effizienz (limitierte Rechenkapazität und schwächere LLMs)

Die Bewertung der erzielten Ergebnisse auf unterschiedlichen Text-To-SQL Benchmarks erfolgt anhand der *Execution accuracy* (EX)¹. Die experimentellen Ergebnisse aus [9] sind in Tabelle 1 zusammengefasst. Beide Varianten des Frameworks können in dem jeweiligen Benchmark mit anderen Arbeiten mithalten.

2.2 Motivation

Das erste Ziel unserer Arbeit soll die Replikation der Ergebnisse aus Tabelle 1 auf den verschiedenen Benchmarks sein. Da wir die Experimente nur mit beschränkten Ressourcen durchführen können und keinen Zugang zu proprietären LLMs haben, sind wir insbesondere an der Replikation der Konfigurationsvariante $CHES_{(IR,SS,CG)}$ mit quelloffenen LLMs interessiert und müssen andere ressourcenintensive Konfigurationen weitestgehend ausschließen. Zudem wird in [9] mehrfach die Adaptivität des vorgestellten Ansatzes an ressourcenbeschränkte Umgebungen hervorgehoben. Daher erwarten wir für die genannte Konfiguration mit quelloffenen LLMs ähnliche Ergebnisse zu erzielen.

In den Ablations-Studies des replizierten Artikels wird zudem ein signifikanter Einfluss des revise-Tools beschrieben [siehe 9, Tabelle 4], welches fehlerhaft generierten SQL-Anfragen überarbeitet. Dies wollen wir unter Verwendung quelloffener LLMs verifizieren.

Für den Spider-Datensatz wird in [9] nur ein Ergebnis mit proprietären LLMs angeführt. Wir wollen darüber hinaus auch einen Wert mit quelloffenen LLMs beitragen und erwarten, wie schon

¹Die Execution accuracy ist eine übliche Metrik für Text-To-SQL Benchmarks. Für jede Frage wird eine Liste von sogenannten *gold-values* bereitgestellt, welche im Ergebnis der generierten Anfrage enthalten sein müssen. Die EX stellt den Anteil der korrekt berechneten *gold-values* im Verhältnis zu allen generierten SQL-Anfragen dar.

bei den proprietären LLMs gezeigt, eine höhere *Execution Accuracy*, als für den BIRD-Datensatz zu erhalten.

2.3 Behauptungen und eigener Beitrag

Mit den beschriebenen Experimenten wollen wir die folgenden Behauptungen überprüfen:

- **Behauptung 1:** *Das CHESS-Framework in der Konfiguration $CHESS_{(IR,SS,CG)}$ erzielt mit quelloffenen LLMs eine EX auf dem SDS-Datensatz von annähernd $EX = 61.5$.*
- **Behauptung 2:** *Das CHESS-Framework in der Konfiguration $CHESS_{(IR,SS,CG)}$ mit quelloffenen LLMs wird durch das revise-Tool signifikant beeinflusst und erzielt auf dem SDS-Datensatz eine Differenz in der *Execution Accuracy* (ΔEX) zwischen einmaliger Revision und keiner Revision von annähernd $\Delta EX = 61.22 - 57.82 = 3.40$.*
- **Behauptung 3:** *Das CHESS-Framework in der Konfiguration $CHESS_{(IR,SS,CG)}$ erzielt mit quelloffenen LLMs eine Differenz in der *Execution Accuracy* (ΔEX) zwischen dem BIRD-dev und Spider-Datensatz von annähernd $\Delta EX = 65.00 - 61.5 = 3.50$.*

Zu Behauptung 1, 2: Unsere Ressourcenbeschränkung beinhaltet auch eine zeitliche Komponente, weshalb wir nicht den vollen BIRD-dev Datensatz testen können. Daher beschränken wir uns bei Behauptung 1 und Behauptung 2 auf den Subsampled Development Set (SDS) aus [9] mit 147, statt 1553 Frage-Anfrage Paaren, des BIRD-dev Datensatzes. Neben den unterschiedlichen Datenbanken sind in diesen reduzierten Datensatz auch unterschiedliche Schwierigkeitsgrade enthalten. Dennoch hoffen wir bei der Untersuchung von Behauptung 1 annähernd die *Execution accuracy* von $EX = 61.5$ aus [9] auf dem BIRD-dev Datensatz zu erreichen.

Zu Behauptung 2: In den Ablations-Studies von [9] wurde der Einfluss der verschiedenen Agenten und deren Tools auf die EX der generierten Anfragen untersucht. Demnach soll das revise-Tool des Candidate Generators den größten Einfluss haben. Dieses überarbeitet generierte Anfragen, wenn bei deren Ausführung ein Fehler aufgetreten ist (z.B. Syntaxfehler), das Ergebnis leer oder unerwartet war. Der `sampling_count` gibt dabei die Anzahl durchgeführter Revisionen an. Mit `sampling_count = 1` konnte in [9] eine Verbesserung um $\Delta EX = 3.40$ gegenüber keiner Revision (`sampling_count = 0`) erzielt werden. Diese Verbesserung versuchen wir mit quelloffenen LLMs zu replizieren.

Zu Behauptung 3: In einem dritten Experiment wollen wir Tabelle 1 um einen Wert für $CHESS_{(IR,SS,CG)}$ mit quelloffenen LLMs auf dem Spider-Datensatz ergänzen. Wir erwarten eine Differenz zwischen der EX der proprietären Modelle und der EX der quelloffenen Modelle von annähernd $\Delta EX = 3.50$ zu erreichen, wie es bei dem BIRD-dev Datensatz in [9] ermittelt wurde. Die Behauptung 3 ergibt sich damit nicht aus den Behauptungen des Originalpapers, sondern ist der Beitrag unserer Arbeit.

3 Methoden

In diesem Abschnitt beschreiben wir den Aufbau unserer Experimente und die genutzten Ressourcen. Dazu gehen wir zunächst näher auf die einzelnen Agenten des Frameworks und deren Verwendung in den abgeleiteten Varianten ein.

3.1 Modellbeschreibung

Multi-Agent-Framework Die größte Herausforderung bei Text-To-SQL Problemen auf komplexen Datenbanken ist der Umgang mit den vielen Informationen und verschiedensten Dateiformaten, welche den LLMs zur Verfügung gestellt werden. Hinzu kommen Mehrdeutigkeiten in den Fragestellungen und innerhalb der bereitgestellten Informationen (z.B. Spalte *id*). Aus diesem Grund wird in [9] ein Multi-Agent-Framework präsentiert, welches die Aufgabenstellung in vier Teilprobleme zerlegt und diese mit spezialisierten Agenten löst. Dadurch können die vielen Informationen zielgerichteter genutzt und die Komplexität großer Datenbank-Schemata leichter erfasst werden. Jeder Agent bearbeitet die ihm zugeteilten Aufgaben mit speziellen Anfragen an ein LLM [siehe 9, Anhang C].

Im folgenden beschreiben wir kurz die Funktionsweise der vier Agenten, deren Aufgaben wir bereits in Kapitel 2 ausführlicher dargestellt haben.

Der **Information Retriever** extrahiert aus der Fragestellung zunächst wichtige Schlüsselwörter, Entitäten und erwähnte Spaltennamen mit einer *few-shot* Anfrage an ein LLM. Anschließend werden ähnliche Entitäten, Spalten und Werte in der Datenbank gesucht. Dazu wird die syntaktische Ähnlichkeit von Werten mithilfe von Lokaltätssensitivem Hashing (LSH) bestimmt. Semantische Informationen über den Kontext der Daten können dabei helfen, Mehrdeutigkeiten aufzulösen und nur relevante Daten an die anderen Agenten weiterzugeben. Der Kontext der Daten wird mithilfe des Datenbank Katalogs, Metadaten des Schemas und Beschreibungen der Spalten ermittelt.

Um einen geordneten Informationsfluss zwischen den Agenten zu ermöglichen, wird eine interne Repräsentation der Datenbank erzeugt und beliebig verändert. Der **Schema Selector** wählt die benötigten Tabellen und Spalten zum Generieren der SQL-Anfrage mit Hilfe von *few-shot* Anfragen aus. Dabei wird die interne DB-Repräsentation stark reduziert.

Nachdem nun von den vielen Daten vom Anfang nur noch die relevanten Tabellen und Spalten übrig geblieben sind generiert der **Candidate Generator** mithilfe eines weiteren LLM Aufruf die SQL-Anfrage. Anschließend wird die SQL-Anfrage auf der internen DB-Repräsentation ausgeführt und gegebenenfalls einem revise-Tool übergeben, welches mit einem LLM Aufruf die fehlerhafte Anfrage überarbeitet.

Der CG kann auch mehrere mögliche SQL-Anfrage generieren. Dann ist es die Aufgabe des **Unit Tester** die passenste Anfrage auszuwählen. Dazu generiert der Unit Tester für jede Frage k viele Unit Tests, sodass nur eine korrekte SQL-Anfrage diese Tests erfüllt. Dann werden die verschiedenen Kandidaten mithilfe von LLM Aufrufen getestet und bewertet. Schließlich wird die finale SQL-Anfrage ausgewählt.

Framework Varianten In dem replizierten Artikel wird die Notwendigkeit hervorgehoben, Systeme für verschiedene Anwendungsszenarien konfigurieren zu können. Dabei wird vor allem die verfügbare Rechenkapazität und Stärke der LLMs betrachtet. Die Experimente in [9] werden mit zwei konträr konfigurierten Varianten des Frameworks durchgeführt: eine für maximale Effektivität ($CHES_{(IR,CG,UT)}$) und eine für maximale Effizienz ($CHES_{(IR,SS,CG)}$). Diese unterscheiden sich im wesentlichen darin, dass sie für viel/wenig Rechenkapazität und starke/schwache LLMs ausgelegt sind. Wie die Namen der Varianten bereits erkennen lassen, unterscheiden sie sich auch in den verwendeten Agenten.

Überraschenderweise nutzt die $CHES_{(IR,CG,UT)}$ Variante für viel Rechenkapazität nicht den Schema Selector Agent. Dies wird in [9] damit erklärt, dass aktuelle Benchmarks wie der BIRD-Datensatz eher kleinere Datenbanken beinhalten, wodurch der Overhead des Schema Selector Agent nicht benötigt wird und sogar zu einer Reduzierung der Leistung führt. Die Bedeutung des Schema-Linkings für komplexe Datenbanken wird in [9] mit einem anderen Experiment auf einem synthetisch erzeugten Datensatz gezeigt, für den alle vier Agenten benötigt werden.

In unserer Replikation nutzen wir nur die $CHES_{(IR,SS,CG)}$ Variante, bei der auf den Unit Tester verzichtet wurde. Dadurch werden insgesamt weniger LLM Anfragen benötigt, wodurch unsere limitierten Ressourcen besser genutzt werden. Zudem generiert der Candidate Generator nur einen Kandidaten, welcher dann auch als finale SQL-Anfrage genutzt wird. Das revise-Tool innerhalb des CGs wird aber trotzdem verwendet.

LLM Modelle In unserer Konfiguration nutzen wir analog zu [9] *Llama-3-70B* [1] für:

- Die Agenten Information Retriever, Schema Selector und Candidate Generator selbst,
- Die Tools *extract_keywords* und *retrieve_context* des Information Retrievers,
- Die Tools *filter_column*, *select_tables* und *select_columns* und
- Das Tool *revise*.

Im Candidate Generator verwenden wir für das Tool *generate_candidate* das in [9] nachtrainierte Modell *NL2SQL_DeepSeek_33B* [7].

Zusätzlich werden in der Vorberchnung und im Tool *retrieve_entity* des Information Retrievers Embedding Modelle verwendet. Diese werden in [9] nur für die proprietären Modelle beschrieben, weshalb wir für die quelloffenen Modelle solche nutzen, die mit den Embedding Modellen der proprietären Modellen vergleichbare sind: In der Vorberechnung verwenden wir *mxbai-embed-large* [3] und für das *retrieve_entity* Tool *nomie-embed-text* [6].

3.2 Datenbeschreibung

BIRD Der BIRD Benchmark (BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation) [5] beinhaltet 12.751 einzigartige Fragen, verteilt über 95 Datenbanken. Die Daten stammen aus realen Anwendungen und enthalten mitunter komplizierte Formate. Zusätzlich werden Kontextinformationen und externes Wissen (Evidence) bereitgestellt, welche für die Generierung der Anfragen genutzt werden darf. Die Datensätze können von der offiziellen Webseite² herunter geladen werden.

²BIRD: <https://bird-bench.github.io/>

Subsampled Development Set Dieser Datensatz wurde in [9] für weitere Experimente aus dem BIRD *development-set* generiert. Er enthält 10% der Fragen aus dem *dev*-Set und hat 147 Frage-Anfrage Paare, wovon 81 leicht, 54 moderat und 12 schwer sind. Diese Daten werden über das offizielle Github-Repository³ des Artikels bereitgestellt.

Spider Der Spider-Datensatz [10] ist eine umfangreiche, komplexe und domänenübergreifende Sammlung für die Text-to-SQL-Forschung. Er umfasst 10.181 natürlichsprachliche Fragen sowie 5.693 einzigartige und anspruchsvolle SQL-Abfragen, die auf 200 verschiedenen Datenbanken basieren. Insgesamt decken diese Datenbanken 138 unterschiedliche Domänen ab. Der Datensatz steht über das offizielle GitHub-Repository⁴ zum Download bereit. Aus dem Spider-Datensatz verwenden wir das *test*-Set.

Subsampled Spider Bei der Durchführung unserer Experimente mussten wir feststellen, dass wir mit dem kompletten Spider-Datensatz die bereitgestellten Ressourcen über mehrere Tage blockieren würden. Mit Rücksicht auf andere Nutzer haben wir uns dazu entschieden, zunächst nur eine Teilmenge des Datensatzes für unsere Experimente zu nutzen. Dabei haben wir uns an dem SDS-Datensatz orientiert und von allen Datenbanken jeweils 10% der Fragen zufällig ausgewählt. Damit reduziert sich der Datensatz auf 229 Frage-Anfrage Paare.

3.3 Hyperparameter

Die standardmäßige Konfiguration von Ollama sieht eine Kontextlänge von 2048 Token vor. Diese haben wir für *Llama-3-70B* auf die maximal mögliche Kontextlänge von 8192 erhöht, basierend auf den Modell-Ablationen der Autoren: „[...] we can utilize an open-source LLM with a small context window size, specifically Llama-3 with only 8K token“ [9, Anhang D.2] Die Kontextlänge des *NL2SQL_DeepSeek_33B* Modells haben wir auf die im Quellcode beschriebene Länge von 8192 Token gesetzt.

Im Quellcode der vorhandenen Konfigurationen der Autoren wurden die Temperaturen der Modelle vorgegeben – für den LLM Aufruf von *extract_keywords* der Wert 0.2 und für den LLM Aufruf von *generate_candidate* der Wert 0.01. Die übrigen Temperaturen sind auf den Standardwert 0.0 gesetzt.

Als Datentyp für *NL2SQL_DeepSeek_33B* haben wir das im Quellcode gegebene *bfloat16* genutzt [8]. Sonstige Hyperparameter haben wir auf den Standardwerten von *Ollama* und *vLLM* belassen, sofern diese nicht ebenfalls durch den Quellcode der Autoren beeinflusst werden.

3.4 Implementierung

Das von den Autoren bereitgestellte Github-Repository [8] wurde für diese Replikation in ein eigenes Repository⁵ übernommen. Für die lokale Ausführung der Embedding-Modelle und *Llama-3-70B* nutzen wir das Framework *Ollama* in der aktuell verfügbaren Version und für *NL2SQL_DeepSeek_33B* die Bibliothek *vLLM* in der von den Autoren vorgeschlagenen und mit

³SDS: <https://github.com/ShayanTalaie/CHESS>

⁴Spider: <https://yale-lily.github.io/spider>

⁵GitLab-Repository: <https://gitlab.informatik.uni-halle.de/aktxt/re-chess>

```

1 # In $CHESS/src/database_utils/db_catalog/preprocess.py
2 ...
3 from langchain_ollama import OllamaEmbeddings
4 EMBEDDING_FUNCTION = OllamaEmbeddings(model="mxbai-embed-large")
5 ...
6
7 # In $CHESS/src/workflow/agents/information_retriever/tool_kit/retrieve_entity.py
8 from langchain_ollama import OllamaEmbeddings
9 ...
10 def __init__(self):
11     self.embedding_function = OllamaEmbeddings(model="nomic-embed-text")
12     ...

```

Listing 1: Ergänzung von LangChain um das Plugin für Ollama

den übrigen Python-Paketen kompatiblen Version 0.3.3. Dazu mussten wir *LangChain* um das Plugin für *Ollama* ergänzen, wie es in Listing 1 dargestellt ist.

Neben den Konfigurationen der Framework-Varianten haben wir zusätzlich einige Stellen im Quellcode anpassen müssen, ohne die unsere Experimente nicht mit dem Code der Autoren durchführbar wären.

Für unsere Experimente mit dem revise-Tool mussten wir dieses konfigurierbar machen, da in der aktuellen Version der Autoren der `sampling_count` auf eins festgesetzt ist und die generierte Anfrage stets durch die Überarbeitete ersetzt wird. Unsere Änderungen sind in Listing 2 farblich hervorgehoben. Neben der Einführung des Parameters `sampling_count`, mussten wir von den generierten Revisionen eine finale Anfrage auswählen. Dazu haben wir die bereits vorhandene Methode `aggregate_sqls()` genutzt, welche die Revisionen in ein Cluster einfügt und von dem größtem Cluster die kürzeste Anfrage auswählt. Dieses Verfahren kam in einer älteren Version des CHESS-Frameworks zum Einsatz.

Eine weitere Änderung betrifft die Schnittstelle zum Laden der Datensätze der verschiedenen Benchmarks. Hier wird explizit ein `evidence` Parameter angefordert. Dieser wird aber nur im BIRD-Datensatz genutzt. Für unsere Tests mit dem Spider-Datensatz mussten wir diesem Parameter optional machen. Außerdem wird nun das *Retrieve_Context* Tool des IR nicht mehr benötigt. Die Änderungen sind in Listing 3 dargestellt.

Schließlich haben wir aus den geschlossenen Issues des original Repository erfahren, dass es einer Änderung am GC Agenten benötigt, damit dieser das reduzierte Datenbank Schema nutzt und nicht auf das Ursprüngliche zurück greift. Andernfalls wären die Berechnungen des Information Retrievers und Schema Selectors nutzlos. Die Änderungen in Listing 4 sind gering, haben aber einen enormen Einfluss auf die *Execution accuracy*. Erste Testläufe ohne diesen Parameter erreichten eine *Execution accuracy* von ungefähr $EX = 30$. Diese fehlerhaften Testläufe haben wir verworfen und nur jene mit gesetztem Parameter in diese Arbeit übernommen.

3.5 Aufbau der Experimente

Die Experimente führen wir mit der $CHESS_{(IR,SS,CG)}$ Variante durch, welche mithilfe einer yaml-Datei konfiguriert wird. Dort haben wir die zu verwendenden Agenten, deren Tools, sowie


```

1 def __init__(self, ... sampling_count: int = 1 ):
2     ...
3     self.sampling_count = sampling_count
4
5 def _run(self, state: SystemState):
6     ...
7     response = async_llm_chain_call(
8         prompt = ...
9         ...
10        sampling_count=self.sampling_count
11    )
12    response = [r[0] for r in response]
13
14    revised_sqls = [res["revised_SQL"] for res in response]
15    for target_SQL_meta_info in target_SQL_meta_infos:
16        if target_SQL_meta_info.need_fixing:
17            revised_sql = DatabaseManager().aggregate_sqls(sqls=revised_sqls)
18            chosen_res = next(res for res in response if res["revised_SQL"] == revised_sql)
19            refinement_response = { "refined_sql_query": chosen_res["revised_SQL"] }
20        else
21            ...

```

Listing 2: Änderungen an der Implementierung des revise-Tools

```

1 def initialize_tasks(self, dataset: List[Dict[str, Any]]):
2     ...
3     if "evidence" not in data:
4         data = {"evidence": "(No hint provided)", **data}
5
6     if "SQL" not in data and "query" in data:
7         data = {"SQL": data["query"], **data}

```

Listing 3: Änderungen an der Schnittstelle zum Laden der Datensätze

```

1 class GenerateCandidate(Tool):
2     def _run(self, state: SystemState):
3         ...
4         request_kwargs = {
5             "DATABASE_SCHEMA": state.get_schema_string(schema_type="tentative",
6             "QUESTION": state.task.question,
7             "HINT": state.task.evidence,
8         }
9         ...

```

Listing 4: Änderungen an der Implementierung des GC Agenten

die verwendeten LLMs und deren Parameter angegeben. Hier wird unter anderem auch die Anzahl der Kandidaten und Revisionen festgelegt. In allen Experimenten generieren wir nur einen Kandidaten. Zur Überprüfung von Behauptung 1 und Behauptung 3 lassen wir wie in [9] drei Revisionen durchführen.

Für die verschiedenen Experimente benötigen wir die folgenden Konfigurationen der $CH\text{ESS}_{(IR,SS,CG)}$ Variante, welche in unserem Repository unter `$Re-CH\text{ESS}/CH\text{ESS}/run/configs` zur Verfügung stehen:

- `CH\text{ESS}_IR_SS_CG_BIRD_OSS.yaml` - BIRD-Datensatz mit quelloffenen LLMs
- `CH\text{ESS}_IR_SS_CG_SPIDER_OSS.yaml` - Spider-Datensatz mit quelloffenen LLMs

Die Durchführung der Experimente erfolgt auf einem GPU-Server, welcher in zwei verschiedene Knoten unterteilt ist. Für die Ausführung von *Ollama* und *vLLM* wurde jeweils eine NVIDIA A100-SMX4-80GB Grafikkarte auf einem der Knoten reserviert. Dieser Knoten des GPU-Servers hat weiterhin zwei AMD Epyc 7662 Prozessoren mit je 64 physischen und 128 logischen Kernen, Grundtaktfrequenz von 2 GHz und Boostfrequenz von 3.3 GHz sowie 1024 GB Arbeitsspeicher. Davon haben wir 12 Kerne und 128 GB in unserer Job-Konfiguration mit Slurm reserviert. Der Knoten ist an ein Netzwerklaufwerk für das Cluster mit 1.8 TB Speicherplatz angebunden. Dieses Setup wurde für alle Experimente dieser Replikation verwendet.

Die Durchführung steuern wir mithilfe selbst geschriebener Skripte, welche unter `$Re-CH\text{ESS}/scripts` ebenfalls zur Verfügung stehen. Dabei stellt `copyrepo.sh` den Einstiegspunkt für einen Durchlauf dar. Dieses Skript downloaded und entpackt sowohl *Ollama*, als auch die verwendeten Datensätze. Anschließend wird das `runchess.sh` Skript aufgerufen, welches zunächst die Vorverarbeitung und im Anschluss die angegebene Konfiguration startet.

3.6 Ressourcen für Berechnungen

Das verwendete Modell *Llama-3-70B* benötigt mit der genutzten Kontextlänge von 8192 Token etwa 40 GB Grafikspeicher, während *NL2SQL_DeepSeek_33B* etwa 70 GB benötigt. Die Embedding-Modelle werden von *Ollama* gleichzeitig im Grafikspeicher gehalten, damit das Laden und Entladen die Ausführung nicht beeinflusst. Mit einer Größe von ca. 300 MB ist dies für *nomic-embed-text* kein Problem, das Embedding Modell aus der Vorberechnung *mxbai-embed-large* benötigt ebenfalls nur etwa 600 MB. Die Verwendung einer kleineren Grafikkarte für *Llama-3-70B* ist mit *Ollama* möglich, jedoch sind kleinere Grafikkarten nicht auf demselben Knoten des Clusters verfügbar. Daher haben wir uns dafür entschieden, den Kommunikationsoverhead zu reduzieren und die zwei Grafikkarten mit 80GB auf demselben Knoten zu nutzen.

Im Vorfeld der Experimente haben wir damit gerechnet, dass die Laufzeit pro Durchlauf wenige Stunden beträgt, da wir kein *fine tuning* betreiben und auch kein aufwendiges Training durchführen. Jedoch stellten wir fest, dass die Kommunikation zwischen den verschiedenen LLMs und der Einsatz der vielen verschiedenen Agenten zu einer deutlich höheren Laufzeit führen, als ursprünglich von uns angenommen. Für die benötigten CPU/GPU-Stunden bei den verschiedenen Experimenten ergeben sich die folgenden Werte:

Experiment zu Behauptung 1: Bei diesem Experiment wird, wie zuvor beschrieben, der SDS-Datensatz genutzt. Dabei wird pro Anfrage immer ein Kandidat generiert und sollte dieser nicht korrekt sein, drei Revisionen durchgeführt. Die totale Laufzeit betrug hier 18:26:42. Die

sampling_count	EX_{SDS}
0	53.74
1	54.42

Tabelle 2: *Execution Accuracy* für unterschiedliche Werte des *sampling_count* im revise-Tools

akkumulierte Laufzeit über alle CPUs betrug bei diesem Experiment 9 Tage 05:20:24. Damit lag die für dieses Experiment benötigte Laufzeit deutlich über der im Vorfeld erwarteten Laufzeit.

Experiment zu Behauptung 2: Auch bei den Experimenten zu Behauptung 2 hatten wir eine große Abweichung zu der erwarteten Zeit. Das Reduzieren der Revisionen brachte keine große Zeitersparnisse. Für das Experiment mit *sampling_count* = 1 wurden eine Laufzeit von 18:19:26 benötigt. Akkumuliert über alle CPUs beträgt die Laufzeit 9 Tage 03:53:12. Dies ist etwas weniger als zuvor bei Experiment 1, jedoch immer noch deutlich über der von uns erwarteten Laufzeit. Werden keine Revisionen generiert, steigt die Laufzeit sogar wieder leicht an auf 18:22:14, was einer akkumulierten Laufzeit von 9 Tagen 04:26:48 entspricht.

Experiment zu Behauptung 3: Die Experimente von Behauptung 3 haben wir zunächst auf dem Subsampled Spider Datensatz durchgeführt, welcher nochmal kleiner als der SDS-Datensatz ist. Deshalb rechneten wir auch mit einer deutlich geringeren Laufzeit als zuvor. Der Ausgang der Experimente bestätigte diese Annahme. Bei einem generierten Kandidaten und drei Revisionen, zusammen mit *Llama-3-70B* als Embedding-Modell, wurde eine Laufzeit von 09:47:04 benötigt, was einer akkumulierten Laufzeit von 4 Tagen 21:24:48 entspricht. Verwendet man die Embedding-Modelle *mxbai-embed-large* und *nomie-embed-text* beträgt die Laufzeit 09:03:21, was einer akkumulierten Laufzeit von 4 Tagen 21:24:48 entspricht.

Bei den Tests auf dem vollständigen Spider-Datensatz war zu erwarten, dass die Laufzeit bedeutend höher ist, als zuvor. {...**Hier muss dann noch die Laufzeit für den Spider Datensatz hin**}

4 Ergebnisse

In diesem Abschnitt beschreiben wir die Ergebnisse unserer experimentellen Untersuchungen. Diese überprüfen die in Abschnitt 2 aufgestellten Behauptungen über den replizierten Artikel.

Unsere erste Untersuchung repliziert ein Experiment aus [9], bei dem das CHESS-Framework auf dem BIRD-Datensatz mit quelloffenen LLMs getestet wurde. Wir nutzen dabei die gleiche $CHESS_{(IR,SS,CG)}$ Variante. Während in [9] eine *Execution Accuracy* von $EX = 61.5$ erreicht wurde, können wir nur einen Wert von $EX = 54.42$ erreichen. Folglich kann mit diesem Experiment die Behauptung 1 nicht bestätigt werden.

Mit dem zweiten Experiment wurde der Einfluss des revise-Tools untersucht. Dafür wurde das CHESS-Framework in der Konfiguration $CHESS_{(IR,SS,CG)}$ mit quelloffenen LLMs auf dem SDS-Datensatz getestet, wobei das revise-Tool mit *sampling_count* $\in \{0, 1\}$ konfiguriert wurde. Die Ergebnisse sind in Tabelle 2 aufgelistet. Die absoluten Werte der EX weichen, wie auch schon im ersten Experiment, von denen aus [9] ab. Tatsächlich können wir aber auch einen Anstieg der *Execution Accuracy* bei Verwendung des revise-Tools feststellen. Jedoch ist der Einfluss des Tools

Embedding-Modelle	$EX_{SubSpider}$
LLama-3-70B	62.88
mxbai-embed-large & nomic-embed-text	60.70

Tabelle 3: *Execution Accuracy* mit unterschiedlichen Embedding-Modellen auf dem Subsampled Spider Datensatz

Methode \ Datensätze	BIRD test	BIRD dev	Spider
Beste Methode (Stand 03/2025)	77.14 [5]	75.36 [5]	91.2 [10]
$CHES_{(IR,CG,UT)}$	68.31	71.10	-
$CHES_{(IR,SS,CG)}$ + proprietär	66.69	65.00	87.2
$CHES_{(IR,SS,CG)}$ + Open LLMs	-	61.5	???
$\Delta EX = EX_{\text{proprietär}} - EX_{\text{OpenLLM}}$	-	3.5	???

Tabelle 4: Überblick verschiedener CHES-Konfigurationen auf unterschiedlichen Benchmarks

mit einer Änderung von $\Delta EX = 0.68$ nicht signifikant, weshalb wir Behauptung 2 ebenfalls nicht bestätigen können.

Das dritte Experiment liefert einen Wert für die *Execution Accuracy* des CHES-Frameworks mit quelloffenen LLMs auf dem Spider-Datensatz. Da die ersten Experimente bereits eine deutlich höhere Laufzeit hatten, als zuvor erwartet, haben wir zunächst nur den Subsampled Spider Datensatz genutzt und zusätzlich zu den von uns gewählten Embedding-Modellen, auch *Llama-3-70B* als Embedding-Modell getestet. Die Ergebnisse auf dem Subsampled Spider Datensatz mit den verschiedenen Embedding-Modellen sind in Tabelle 3 aufgeführt. Im weitere Verlauf konnten wir dann aber auch den vollständigen Spider-Datensatz testen und einen Wert von $EX = ???$ ermitteln. Die Differenz zu der *Execution Accuracy* von $CHES_{(IR,SS,CG)}$ mit proprietären LLMs auf dem Spider-Datensatz beträgt $\Delta EX = 87.2 - ??? = ???$. Diese Abweichung ist deutlich größer, als von uns erwartet. Mit diesem Ergebnis liefern wir aber einen Wert für Tabelle 4, bei dem $CHES_{(IR,SS,CG)}$ mit quelloffenen LLMs auf dem Spider-Datensatz getestet wurde.

Die Ergebnisse unserer Experimente zeigen, dass keine unserer formulierten Behauptungen zutrifft. Jedoch sehen wir verschiedene Hinweise, die für die geringere *Execution Accuracy* verantwortlich sein können. Daher werden wir im Folgenden näher auf die Ergebnisse der Experimente eingehen.

Ergebnisse zu Behauptung 1 Das erste Experiment sollte die allgemeine Leistungsfähigkeit des CHES-Frameworks zum Bearbeiten von Text-To-SQL-Aufgaben bestätigen. Dazu wurde ein Experiment aus [9] repliziert, bei dem die $CHES_{(IR,SS,CG)}$ Variante mit quelloffenen LLMs getestet wurde. In [9] wurde mit dieser Variante auf dem vollständigen BIRD-dev Datensatz ein Wert von $EX = 61.5$ erreicht. Mit Behauptung 1 sollte dieser Wert für den SDS-Datensatz, einer Teilmenge des BIRD-dev Datensatzes, bestätigt werden. Jedoch konnten wir nur einen Wert von $EX = 54.42$ erreichen. Die Abweichung führen wir auf die genutzten Embedding-Modellen zurück. Diese werden in der Arbeit für die Experimente mit quelloffenen LLMs nicht genauer benannt. Im Quelltext werden für die Embedding-Modelle nur proprietäre LLMs genutzt. Mit *mxbai-embed-large* und *nomic-embed-text* haben wir versucht ähnliche Modelle zu wählen. Nichtsdestotrotz könnten diese aber für die geringere *Execution Accuracy* verantwortlich sein.

	$sc = 0$	$sc = 1$
correct	79	80
incorrect	65	56
error	3	11

Tabelle 5: Anteil der korrekten, inkorrekten und fehlerhaften SQL-Anfragen

Ergebnisse zu Behauptung 2 In [9] wird dem revise-Tool des CG-Agenten ein signifikanter Einfluss auf die *Execution Accuracy* zugeschrieben. In diesem Experiment wurde der Einfluss dieses Tools genauer untersucht. Die Ergebnisse sind in Tabelle 2 aufgelistet. Aus den ermittelten Werten kann geschlussfolgert werden, dass der Einsatz des revise-Tools zu einer höheren *Execution Accuracy* führt. Jedoch konnten nicht der gleiche signifikante Einfluss feststellen werden, wie in [9]. Auch hier sehen wir den Einfluss der Embedding-Modelle als mögliche Fehlerquelle.

Eine weitere interessante Beobachtung bei diesem Experiment ist die Verteilung der korrekten, inkorrekten und fehlerhaften SQL-Anfragen, wie sie in Tabelle 5 dargestellt sind. Anhand der absoluten Werte kann festgestellt werden, dass nur eine inkorrekte Anfrage in eine korrekte Anfrage umgewandelt werden konnte und die anderen zu fehlerhafter Anfragen umgewandelt wurden. Das zeigt, dass der Einfluss des revise-Tools wirklich nicht signifikant ist. Andererseits bedeutet das Ergebnis auch, dass durch die Revision neue Syntaxfehler entstehen. Eine Erklärung dieses Phänomens sehen wir in der Auswahl der finalen Anfrage, bei der stets die kürzeste Anfrage ausgewählt wird. Dieses Verfahren erscheint bei komplexen Datenbank-Anfragen nicht besonders erfolgversprechend zu sein. Da bei den Ablation-Studies in [9] aber das gleiche Verfahren verwendet wurde, haben wir hier keine Änderungen vorgenommen.

Ergebnisse zu Behauptung 3 Zusätzlich zu den in [9] durchgeführten Experimenten wurde das CHESS-Framework auf dem Spider-Datensatz getestet. Für die $CHESS_{(IR,SS,CG)}$ Variante gibt es bereits einen Wert für proprietär LLMs. Mit diesem Experiment wird zusätzlich einen Wert für quelloffene LLMs bereitgestellt. Aufgrund von Ressourcenbeschränkungen und Rücksicht auf andere Nutzer des GPU-Servers haben wir zunächst nicht den kompletten Spider-Datensatz getestet, da wir andernfalls den GPU-Server für mehrere Tage blockieren würden. Zudem haben wir aufgrund der Ergebnisse von Experiment 1 und 2 die genutzten Embedding-Modelle hinterfragt und zusätzliche *Llama-3-70B* als Embedding-Modell getestet. Die Ergebnisse in Tabelle 3 zeigen, dass mit *Llama-3-70B* ein leicht höherer *EX* Wert erreicht werden konnte. Abseits der Arbeit durchgeführte Tests mit *Llama-3-70B* als Embedding-Modell für Experimente auf dem BIRD-Datensatz haben aber keine Verbesserungen ergeben.

Später konnte dann aber doch der vollständige Spider-Datensatz getestet werden, wobei eine *Execution Accuracy* von ??? erreicht werden konnte. Die Differenz zu der *Execution Accuracy* von $CHESS_{(IR,SS,CG)}$ mit proprietären LLMs auf dem Spider-Datensatz beträgt $\Delta EX = 87.2 - ??? = ???$. Diese Abweichung ist deutlich größer, als von uns erwartet. Mit den vorangegangenen Tests auf dem Subsampled Spider Datensatz mit unterschiedlichen Embedding-Modellen können wir sicher behaupten, dass die Wahl der Embedding-Modelle einen großen Einfluss auf die ermittelte *Execution Accuracy* hat. Wir gehen davon aus, dass mit einem anderen Embedding-Modell auch noch bessere *EX* Werte möglich sind.

5 Diskussion

Aus unseren experimentellen Untersuchungen können wir eine Reihe von Schlussfolgerungen über den in [9] präsentierten Text-To-SQL-Ansatz im Allgemeinen und die Replizierbarkeit dessen im Speziellen ziehen.

Zunächst möchten wir festhalten, dass der Artikel mit Einschränkungen replizierbar ist. Einerseits werden in dem Artikel ausführliche Erklärungen über die Funktionsweise des Frameworks und den Ablauf der Experimente gegeben. Außerdem ist das Repository gut strukturiert, der Code kommentiert und aussagekräftige Bezeichner verwendet. Andererseits mussten wir aber feststellen, dass wichtige Informationen und Parameter zur Replikation der Experimente fehlen und das revise-Tool anders implementiert wurde, als im Artikel beschrieben. Einen Grund für die geringeren *EX* Werte sehen wir in der Wahl der Embedding-Modelle, welche in [9] nicht beschrieben werden. Dies erschwert die Replikation des Artikels deutlich.

Mithilfe unserer Experimente können wir jedoch die Behauptungen in [9] größtenteils stützen. Bessere Replikationen wären durch eine ausführlichere Dokumentation der Experimente möglich. In [9] wird nicht darauf eingegangen, wie die einzelnen Ergebnisse ermittelt werden. Zudem stehen nur für ein Experiment die Rohdaten zur Verfügung. Wichtige Informationen wie zum Beispiel die genutzten Embedding-Modelle müssen geraten werden.

Aufgrund der deutlich geringeren *EX* Werte unserer Experimente stellen wir die hervorgehobene Adaptivität des Frameworks an ressourcenbeschränkte Umgebungen in Frage. Jedes unserer Experimente zeigt einen qualitativen Unterschied zwischen quelloffenen Modellen und den leistungsstarken proprietären Modellen aus [9]. Daher können wir nicht bestätigen, dass das Framework mit limitierten Ressourcen die gleichen Ergebnisse erzielen kann.

5.1 Was war einfach?

Zu Beginn unserer Arbeit mussten wir zunächst den Artikel erfassen, dessen Kernaussage ermitteln und die Experimente nachvollziehen. All dies war durch die ausführlichen Erklärungen und den übersichtlichen Code gewährleistet. Die Beschreibung der Experimente ermöglichte einen schnellen Überblick über dessen Schwerpunkt und Ergebnisse.

Bis auf wenige Ausnahmen stellen die Autoren alle nötigen Informationen bereit, um eine Reproduktion der Experimente durchzuführen. Alle genutzten Datensätze sind online verfügbar, oder werden bereitgestellt, wie es bei dem SDS-Datensatz der Fall ist. Eine Setup-Anleitung wird mitgeliefert und der Code ist ausführbar. Die verwendeten Konfigurationen sind ebenfalls im Repository enthalten.

Da wir in unseren Experimenten andere Modelle genutzt haben, mussten wir die Konfigurationen anpassen. Die entsprechende Schnittstelle ist gut dokumentiert, sodass wir die LLMs schnell einbinden und die Parameter setzen konnten.

5.2 Was war schwer?

Die Experimente in [9] wurden bis auf eine Ausnahme ausschließlich auf proprietären LLMs durchgeführt. Für die Reproduktion dieser Arbeit werden also in jedem Fall leistungsstarke LLMs und monetäre Ressourcen benötigt. Dies schränkt die Reproduzierbarkeit im akademischem Umfeld sehr stark ein und hat auch unsere Arbeit nachteilig beeinflusst. Zudem werden nur die Experimente mit proprietären LLMs detailliert beschrieben. Für unsere Replikation mit quelloffenen LLMs fehlten uns wichtige Informationen und Parameter, welche wir deshalb raten mussten. Konkrete Fragen haben sich zu den genutzten Embedding-Modellen ergeben, welche nicht weiter benannt werden. Im Repository konnten wir nur proprietäre Embedding-Modelle finden. Daher stellt sich für uns die Frage, welches leistungsstärkere Embedding-Modell in [9] für das Experiment mit den quelloffenen LLMs genutzt wurde, da die Ergebnisse mit unserer Wahl deutlich schlechter ausgefallen sind.

Bei der genaueren Untersuchung des revise-Tools haben wir gravierende Unterschiede zwischen der Beschreibung des Tools in [9] und der Implementierung im Repository entdeckt. Während in dem Artikel beschrieben wird, dass *„For each faulty candidate, the agent repeatedly attempts to revise the query, until the issue is resolved or a maximum number of allowed revisions is reached.“* [9], stellten wir fest, dass die Implementierung eine parallele Revision, statt einer Wiederholten durchführt. Wir sehen die Autoren in der Pflicht diese Inkonsistenz zu beheben und gegebenenfalls die Experimente neu durchzuführen, da dieses Tools einen signifikanten Einfluss auf die *Execution Accuracy* hat. Zudem ist die Anzahl der durchgeführten Revisionen nicht konfigurierbar, obwohl sie es laut [9] sein sollte. Dies führte dazu, dass wir zur Überprüfung von Behauptung 2 zunächst die Implementierung entsprechend erweitern mussten.

Bei der Untersuchung von Behauptung 3 haben wir den Spider-Datensatz verwendet, welcher für Experimente in [9] bereits genutzt wurde. Jedoch mussten wir auch hier wieder feststellen, dass dieser nicht ad-hoc verwendet werden kann, da Spider eine andere Schnittstelle als die BIRD-Datensätzen nutzt und die passende Implementierung im Code nicht enthalten ist. Die entsprechenden Anpassungen im Code sind zwar trivial, aber gerade aus diesem Grund sehen wir hier die Autoren in der Pflicht, die Implementierung der Schnittstelle so zu gestalten, dass auch andere Datensätze, wie zum Beispiel die neue Version des Spider-Datensatz [4], nutzbar sind.

5.3 Empfehlungen für die Replizierbarkeit / Reproduzierbarkeit

Replikationen werden typischerweise im akademischem Umfeld erstellt, in dem häufig nur limitierte Ressourcen zur Verfügung stehen. Zudem können proprietäre Modelle aufgrund von Lizenzrechten nur selten verwendet werden. In [9] wird zwar die Adaptivität an ressourcenbeschränkte Umgebungen hervorgehoben und auch ein Experiment mit quelloffenen LLMs durchgeführt, jedoch hat unsere Replikation gezeigt, dass es trotzdem qualitative Unterschiede gibt. Aus diesem Grund sollten Experimente auch mit schwächeren LLMs durchführbar sein.

Text-To-SQL Ansätze stehen im ständigen Wettbewerb untereinander und eine gute Performance in anerkannten Benchmarks ist entscheidend für deren Verwendung. Aus diesem Grund sollte die Schnittstelle zum Nutzen der Benchmarks mehr Beachtung finden und für verschiedenste Benchmarks genutzt werden können.

6 Kommunikation mit den Autoren

Es gab den Versuch der Kontaktaufnahme mit den Autoren der Originalarbeit, um einen guten Fortschritt, sowie eine hohe Qualität für die Replikation zu erreichen. Dabei wurde versucht, offene Fragen, welche sich weder mithilfe der Originalarbeit noch der dazugehörigen Dokumentation beantworten ließen, zu klären. Die dafür verfasste E-Mail fokussierte sich auf 3 Fragestellungen, welche das Projekt maßgeblich vorangebracht hätten.

Zum Einen ist an keiner Stelle dokumentiert, welche *temperatures* für die LLMs bei den Open-Source-Experimenten verwendet wurden. Es gibt lediglich die in der standardmäßigen Konfiguration angegebenen *temperatures*, an denen man sich orientieren kann. Aus diesem Grund wurden die Autoren gefragt, ob sie für ihre Open-Source-Experimente von dem Standard abweichende *temperatures* verwendet haben. Sollte dies der Fall gewesen sein, haben wir die Autoren der Originalarbeit darum gebeten, uns diese weiterzugeben.

Die zweite Fragestellung bezieht sich auf die in den Open-Source-Experimenten der Originalarbeit verwendeten Embedding-Modelle. Aus [9] geht nicht eindeutig hervor, welche Embedding-Modelle für die Experimente der Originalarbeit verwendet wurden. Aus diesem Grund entschieden wir uns dazu, noch einmal nachzufragen, ob tatsächlich *Llama-3-70B* für das Embedding-Modell verwendet wurde, oder doch andere Modelle zum Einsatz kamen.

Des Weiteren gab es bei den Experimenten zu den Ablation-Studies Probleme, das Framework ohne Revisionen zu nutzen. Dies ist für die Experimente zu den Ablation-Studies zwingend notwendig, weshalb wir die Autoren nach ihrer Lösung für dieses Problem befragten. Leider gab es bis zum aktuellen Zeitpunkt zu keiner gestellten Frage keine Rückmeldung, weshalb unserer Lösungen für die beschriebenen Probleme mit dem Vorgehen aus der Originalarbeit abweichen können.

Literatur

- [1] Aaron Grattafiori u. a. *The Llama 3 Herd of Models*. 2024. DOI: [10.48550/ARXIV.2407.21783](https://doi.org/10.48550/ARXIV.2407.21783).
- [2] Jiaqi Guo u. a. „Towards complex text-to-sql in cross-domain database with intermediate representation“. In: *arXiv preprint arXiv:1905.08205* (2019). DOI: [10.48550/ARXIV.1905.08205](https://doi.org/10.48550/ARXIV.1905.08205).
- [3] Sean Lee u. a. *Open Source Strikes Bread - New Fluffy Embedding Model*. 2024. URL: <https://www.mixedbread.com/blog/mxbai-embed-large-v1> (besucht am 27.02.2025).
- [4] Fangyu Lei u. a. „Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows“. In: *arXiv preprint arXiv:2411.07763* (2024). DOI: [10.48550/ARXIV.2411.07763](https://doi.org/10.48550/ARXIV.2411.07763).
- [5] Jinyang Li u. a. „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* 36 (2023), S. 42330–42357. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf (besucht am 24.02.2025).

- [6] Zach Nussbaum u. a. „Nomic Embed: Training a Reproducible Long Context Text Embedder“. In: *arXiv* (2024). DOI: [10.48550/ARXIV.2402.01613](https://doi.org/10.48550/ARXIV.2402.01613). arXiv: [2402.01613](https://arxiv.org/abs/2402.01613) [cs.CL].
- [7] Shayan Talaei und Mohammadreza Pourreza. *AI4DS/NL2SQL_DeepSeek_33B*. 2024. URL: https://huggingface.co/AI4DS/NL2SQL_DeepSeek_33B (besucht am 27.02.2025).
- [8] Shayan Talaei und Mohammadreza Pourreza. *CHESS. README – CHESS: Contextual Harnessing for Efficient SQL Synthesis*. 2024. URL: <https://github.com/ShayanTalaei/CHESS> (besucht am 27.02.2025).
- [9] Shayan Talaei u. a. *CHESS: Contextual Harnessing for Efficient SQL Synthesis*. 2024. DOI: [10.48550/ARXIV.2405.16755](https://doi.org/10.48550/ARXIV.2405.16755).
- [10] Tao Yu u. a. „Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task“. In: *arXiv preprint arXiv:1809.08887* (2018). DOI: [10.48550/ARXIV.1809.08887](https://doi.org/10.48550/ARXIV.1809.08887).
- [11] Tao Yu u. a. „Typesql: Knowledge-based type-aware neural text-to-sql generation“. In: *arXiv preprint arXiv:1804.09769* (2018). DOI: [10.48550/ARXIV.1804.09769](https://doi.org/10.48550/ARXIV.1804.09769).
- [12] Victor Zhong, Caiming Xiong und Richard Socher. „Seq2sql: Generating structured queries from natural language using reinforcement learning“. In: *arXiv preprint arXiv:1709.00103* (2017). DOI: [10.48550/ARXIV.1709.00103](https://doi.org/10.48550/ARXIV.1709.00103).