Digital Business University of Applied Sciences

Bachelorarbeit zum Thema:

# Entwicklung und Evaluation eines LangGraph-basierten Text-to-SQL-Analysten für Unternehmensdatenbanken

zur Erlangung des Grades Bachelor


**Eingereicht von**

| | |
|---|---|
| Vorname, Name: | Mareike Lass-Hennemann |
| E-Mail: | mareike.lass-hennemann@student.dbuas.de |
| Matrikelnummer: | 190190 |
| Fachsemester: | 8 |
| Studiengang: | Data Science & Business Analytics (B. Sc.) |
| Datum Abgabe: | 28.10.2025 |
| Gutachter:in: | Prof. Dr. Marcel Hebing, Prof. Dr. Claudia Baldermann |

## Executive Summary

Die Arbeit konzentriert sich auf die Entwicklung und Bewertung eines Text-to-SQL-Analysten, der auf dem LangGraph-Framework basiert und natürliche Sprachanfragen (Natural Language, NL) interpretiert, relevante Datenbankstrukturen identifiziert, passende SQL-Abfragen generiert, ausführt und verständliche Antworten in natürlicher Sprache formuliert. Der Analyst ist als strukturierter Workflow mit einer optionalen Iteration implementiert. In einigen Schritten wird ChatGPT 4.0 eingesetzt, z.B. für die Auswahl von relevanten Teil-Schemata, für die Generierung der SQL-Abfrage und für die Formulierung einer passenden Antwort mit dem Abfrage-Ergebnis.

Als Zielumgebung für die Evaluierung dient Google BigQuery mit realen Daten aus den Bereichen Publishing und GA4-Tracking.

Die SQL-Abfragen des Analysts und deren Ergebnisse werden mit idealen SQL-Abfragen als Grundwahrheit verglichen, um zu analysieren, welche Komponenten des Workflows zur Verbesserung der Ausführbarkeit, Ergebnisgenauigkeit und semantischen Übereinstimmung der Abfragen beitragen. Die Varianten der zu beurteilenden Komponenten können gegeneinander ausgetauscht bzw. aktiviert oder deaktiviert werden, sodass über den Vergleich verschiedener Analyst-Versionen Rückschlüsse gezogen werden können, welchen Einfluss die jeweils eingesetzten Komponenten auf das Endergebnis haben. Bewertet werden die Komponenten Kontext, Prompting und Retry.

Die Komponente Kontext ist für die Ausführbarkeit der generierten SQL signifikant wichtig. Die erwartete Verbesserung der Ergebnisgenauigkeit konnte nicht bestätigt werden. Dies könnte im speziellen Aufbau der Komponente begründet liegen, der Schwächen aufweist, sodass es empfehlenswert wäre, den Aufbau mit Hilfe von Prompt-Engineering und einer Minimierung der Anzahl der Einsätze des Sprachmodells zu optimieren oder alternativ entweder über den Aufbau eines RAG-Systems oder über Finetuning des Sprachmodells nachzudenken.

One Shot Prompting liefert bzgl. Ausführbarkeit und semantischer Übereinstimmung signifikant bessere Werte als Chain of Thought Prompting. Dies bestätigt Erkenntnisse aus der Literaturrecherche, wonach Chain of Thought Prompting im Bereich Text-to-SQL erst in Kombination mit weiteren Methoden effektiver ist als One Shot Prompting. Es könnte demnach zukünftig darüber nachgedacht werden, beide Methoden zu kombinieren und ggf. weitere Strategien wie z.B. Decomposition zu ergänzen, um die Leistung weiter zu optimieren.

Die Option auf eine Wiederholung aller Schritte in einem Retry verbessert signifikant die Ausführbarkeit und die Ergebnisgenauigkeit – einen Retry zu ermöglichen erscheint also immer sinnvoll, könnte aber höhere laufende Kosten verursachen durch die wiederholte Anfrage der LLM-Schnittstellen.

Die Fragen, die die Analysten bearbeiten sollen, lassen sich in SQL-Abfragen auf zwei Komplexitätsstufen differenzieren und werden in zwei Formulierungsstilen gestellt – so kann eine Einschätzung erfolgen, ob ein Analyst für den Einsatz bei Stakeholdern oder Daten-Experten und in welchem Umfang geeignet ist. Hierbei hat die Analyse ergeben, dass die Werte der Zielmetriken für die Formulierung der Fragen ohne Hintergrundwissen etwas geringer ausfallen und bei Formulierungen ohne Datenwissen häufiger leere Abfrage-Ergebnisse geliefert werden. Ein Einsatz der Analysten für Stakeholder ist somit noch nicht bedenkenlos möglich.

Die erwartete Komplexität der SQL macht einen deutlichen Unterschied für die Zielmetriken und auch Fehlermeldungen, sodass die Analysten auf ihrem heutigen Stand bei einfachen SQL-Abfragen deutlich zuverlässiger agieren als bei schwierigen.

In der Gesamtsicht sollte zum aktuellen Stand der Entwicklung kein Analyst in Produktion gehen, sondern aufgrund der hohen anteiligen semantischen Übereinstimmung (EM_share) und niedrigen anteiligen Ergebnisgenauigkeit (EA_share) bei allen Analysten diskutiert werden, ob eine menschliche Überprüfung der Abfrage vor ihrer Ausführung, die Schwächen in der Schemaverknüpfung zusätzlich ausgleichen und die Ergebnisgenauigkeit verbessern kann.

Weiterführend könnte diskutiert werden, den Analyst-Workflow in Richtung Agent auszubauen, da für Agents in der Literatur vielversprechende Ergebnisse veröffentlicht wurden. Allerdings sollte berücksichtigt werden, dass in allen Studien mit Benchmarks gearbeitet wurde, auf die die Testfragen genau zugeschnitten waren. So sind die Ergebnisse nicht direkt vergleichbar mit den in dieser Arbeit durchgeführten Tests, die anhand von realen Testfragen auf einer komplexen Unternehmensdatenbank durchgeführt wurden.

Insgesamt können Investitionen in die Entwicklung aller getesteten Komponenten lohnenswert sein, wobei in LangGraph die Implementierung des Retrys am einfachsten erscheint.

Wenn Datenbanken inhouse betreut werden und Schnittstellen für ein LLM vorhanden sind, ist es auf Basis der Analyse dieser Arbeit empfehlenswert, selbst mit Text-to-SQL-Workflows über Open-Source Frameworks wie z.B. LangGraph zu experimentieren, um die Abhängigkeit von Datenteams zu reduzieren, einer breiteren Masse an Mitarbeitenden den selbständigen Datenzugriff zu ermöglichen und so datengetriebene Entscheidungen zu fördern.

# Inhaltsverzeichnis

**Executive Summary** ..... **i**

**Abbildungsverzeichnis** ..... **iv**

**Abkürzungsverzeichnis** ..... **iv**

**1.Einleitung** ..... **1**

*1.1.Kontext* ..... *3*

*1.2.Prompting* ..... *3*

*1.3.Retry* ..... *5*

*1.4.Forschungsfrage und Hypothesen* ..... *5*

**2.Daten und Methoden** ..... **6**

*2.1.Methode und Design* ..... *6*

*2.2. Workflow und einzelne Knoten in LangGraph* ..... *6*

*2.3.Aufbau der Datenbank* ..... *9*

*2.4.Testfragen, Grundwahrheit* ..... *9*

*2.5.Metriken* ..... *10*

**3.Ergebnisse** ..... **13**

**4.Diskussion und Handlungsempfehlung** ..... **18**

**5.Quellenverzeichnis** ..... **21**

**6.Anhang** ..... **24**

*6.1. unterschiedliche Workflows entsprechend Ablation Studies* ..... *24*

## Abbildungsverzeichnis

Abbildung 1: Übersicht Workflow ..... 6
Abbildung 2: Kontext - Valid SQL ..... 14
Abbildung 3: Kontext - Execution Accuracy ..... 14
Abbildung 4: Prompting - Execution Accuracy ..... 15
Abbildung 5: Prompting - Exact Match ..... 15
Abbildung 6: Retry - Valid SQL ..... 15
Abbildung 7: Retry - Execution Accuracy ..... 15
Abbildung 8: Zielmetriken aller Analysten ..... 16
Abbildung 9: Anteilige Berechnung EA und EM aller Analysten ..... 16
Abbildung 10: Auswertung Fragenformulierung ..... 18
Abbildung 11: Auswertung Komplexität SQL ..... 18

## Abkürzungsverzeichnis

| **Abkürzung** | **Bedeutung** |
|---|---|
| SQL | Structured Query Language |
| LLM | LLM Large Language Model |
| COT | Chain of Thought |
| OS | One Shot |
| ICL | In-Context-Learning |
| VA | Valid SQL (Ausführbarkeit) |
| EA | Execution Accuracy (Ergebnisse) |
| EM | Exact Match (SQL) |
| RAG | Retrieval Augmented Generation |

# 1.Einleitung

Text-to-SQL bezeichnet die automatisierte Übersetzung von Anfragen in natürlicher Sprache (Natural Language, NL) in ausführbare SQL-Abfragen (Kanburoglu & Tek, 2024, S.405; Yan et al., 2025, S.1; Nascimento et al., 2024 S.1; Zhu et al., 2024, S.1). Da Organisationen zunehmend auf relationale Datenbanken zur Verwaltung ihrer wachsenden Informationsmengen angewiesen sind, gleichzeitig aber SQL als technische Fähigkeit zur Abfrage der Daten nicht von allen Mitarbeitenden beherrscht wird, hat das Verfahren in den letzten Jahren zunehmend an Bedeutung gewonnen (Zhu et al., 2024, S. 1) und könnte die Lücke zwischen dem Bedarf an Zugang zu Daten und der Möglichkeit, diese Daten zu extrahieren, schließen (Mohammadjafari et al., 2025, S. 1). Der Datenzugriff wird demokratisiert, indem die Abhängigkeit von technischen Teams reduziert wird, zudem können Unternehmen schnellere und fundiertere Entscheidungen treffen, wenn es einer breiten Masse an Mitarbeitenden möglich wird, selbst datengestützte Erkenntnisse zu gewinnen (Ojuri et al., 2025, S.1).

Die Forschung im Bereich Text-to-SQL hat sich von regelbasierten Methoden über Deep Learning und pre-trained Language Models hin zu Large Language Models (LLMs) entwickelt. Durch den Einsatz der aktuellsten, großen Sprachmodelle, wie der GPT-Serie oder LLaMA konnten große Fortschritte erzielt werden, insbesondere im Hinblick auf das grundsätzliche Verstehen von NL-Anfragen und deren Verlinkung auf komplexere Datenbankschemata, sowie durch die Fähigkeit von Sprachmodellen Code zu generieren. Dennoch bleibt Text-to-SQL ein aufstrebender Forschungsbereich mit Potential für weitere Studien (Mohhamadjafari et al., 2025, S.2-4; Hong et al., 2025, S.4; Martra, 2024, S.15).

Aktuelle Studien konzentrieren sich auf die Entwicklung und Evaluierung von Systemen auf großen Benchmarks und heben die Schwierigkeit der Generalisierung hervor (Mohammadjafari et al., 2025, S.1; Hong et al., 2025, S.3-4). Hier soll die vorliegende Arbeit für den Bereich Publishing eine Lücke schließen, indem ein LLM-basierter Workflow entwickelt wird, der auf die realen Unternehmensdaten ausgerichtet ist und im Zielbild Mitarbeitenden des Publishing Unternehmens zukünftig als Chat zur Verfügung stehen soll.

Es bestehen verschiedene Möglichkeiten LLMs für Text-to-SQL einzusetzen. Intelligente Agenten nutzen das LLM als kognitiven Kern zur Steuerung und verfügen über Fähigkeiten wie Planungskompetenz, Gedächtnisleistung und Ausführung von Aktionen z.B. über die Nutzung von Werkzeugen. Das LLM entscheidet demnach dynamisch und eigenständig über die auszuführenden Schritte zur Lösung einer Aufgabe. Agenten können zudem bei Bedarf auf externe Ressourcen wie API´s, Code-Interpreter oder Datenbanken zugreifen (Xi et al.,

2025, S.4; Wang et al., 2024, S.2 ff.; Schluntz & Zhang, 2024). Eine Erweiterung stellen Multi-Agenten-Frameworks aus kollaborierenden Agenten dar wie z.B. MAC-SQL: der „Selector“ zerlegt eine große Datenbank in kleinere Sub-Datenbanken, der „Decomposer“ teilt eine komplexe Frage in einfachere Teilfragen, die schrittweise gelöst werden und der „Refiner“ holt Ausführungs-Feedback ein und kann darauf basierend fehlerhafte SQL-Abfragen reparieren (Wang et al., 2025, S.2 ff.).

Workflows sind dagegen Systeme, in denen LLMs und Tools über vordefinierte Codepfade koordiniert werden (Schluntz & Zhang, 2024). In einem Workflow legt also der Entwickler, nicht das LLM, die Schritte zur Lösung einer Aufgabe vorab fest. Im Bereich Text-to-SQL wird typischerweise mit einem mehrstufigen Ansatz gearbeitet. Dieser besteht unter anderem aus Komponenten wie Verständnis der Frage und Identifizierung der Zielkomponenten aus dem Datenbankschema. Diese können ggf. auch als „Retrieval“- oder „Schema Linking“-Schritt implementiert sein. Es folgt die Generierung der SQL unter Einbeziehung der vorherigen Schritte und optional eine Korrektur nach Ausführungs- oder Modell-Feedback (Maamari et al., 2024, S.1-2; Hong et al., 2025, S.2; Pourreza & Rafiei, 2023, S.4). Weitere Ansätze empfehlen zusätzlich eine Zerlegung („Decomposition“) von bestimmten SQL-Abfragen in kleinere Sub-Abfragen, um die Genauigkeit bei komplexen Abfragen zu verbessern (Pourreza & Rafiei, 2023, S. 5-6).

Im Rahmen dieser Arbeit erscheint die Entwicklung eines strukturierten LLM-basierten Workflows sinnvoller als die Entwicklung eines Agents, da ein Workflow eine klare Definition und modulare Verwaltung aller Schritte ermöglicht. Alle Schritte können kontrolliert und verschiedene Versionen von Abläufen gezielt gestaltet und getestet werden. Workflows bieten lt. Schluntz & Zhang (2024) Vorhersagbarkeit und Konsistenz für klar definierte Aufgaben und können dennoch komplex gestaltet werden.

Im Bereich Text-to-SQL wurden zuletzt große Fortschritte durch den Einsatz moderner Sprachmodelle erzielt. Dennoch bleiben einige Herausforderungen bestehen, die sich ergeben aus a) der Ambiguität und linguistischen Komplexität von NL-Anfragen, b) der Notwendigkeit des Schema-Verständnisses der zugrunde liegenden Datenbankstrukturen und Schwierigkeiten, die sich aus der passenden Präsentation von Schema-Informationen ergeben, c) der Schwierigkeit auch komplexe und seltene SQL Operationen generieren zu lassen bei ggf. fehlenden Trainingsdaten, d) der Notwendigkeit der Generalisierung über verschiedene Domänen hinweg bei unterschiedlichem Vokabular, Strukturen und Mustern (Mohammadjafari et al., 2025, S.1; Hong et al., 2025, S.3-4).

Die o.g. Herausforderungen sollen in dieser Arbeit adressiert werden. Dazu werden nachfolgend die Punkte Kontext, Prompting und Retry genauer betrachtet.

### 1.1.Kontext

Kontext stellt in der vorliegenden Arbeit die Interpretation und Implementierung der in der Literatur als Retrieval, Schema-Verständnis und/oder Schemaverknüpfung benannten Schritte dar und adressiert die Herausforderung b) Schemata in einer Form und einem Umfang zu präsentieren, sodass ein LLM dieses verstehen und die NL-Frage korrekt auf die Datenbank abbilden kann. Kontext wird als kritischer vorbereitender Schritt für eine genaue Übersetzung angesehen (Pourreza & Rafiei, 2023, S.4; Zuckarelli, 2025, S.53). In dieser Arbeit wird sich auf die Zuführung von Schema-Informationen über den Prompt konzentriert. Für Prompting ist zu beachten, dass das Kontextfenster eines LLMs eine begrenzte Token-Anzahl aufnehmen und die Einbeziehung irrelevanter Elemente, Halluzinationen fördern und die Fehleranfälligkeit erhöhen kann (Hong et al., 2025, S.13; Wang et al., 2025, S.3). Gleichzeitig könnten bei der Entfernung irrelevanter Elemente versehentlich auch wichtige Spalten gelöscht werden - hier muss durch Schemaverknüpfung ein ausgewogener Kompromiss gefunden werden (Maamari et al., 2024, S.1) – allerdings wird von Maamari et al. (2024, S.8) auch bereits diskutiert, ob die Bedeutung der Schemaverknüpfung mit sinkenden Kosten, größeren Kontextfenstern und verbesserten Generierungsfähigkeiten abnehmen könnte.

Kontext kann außerdem bei den Herausforderungen a) Mehrdeutigkeit in NL-Fragen überbrücken und d) der Generalisierung bzw. im vorliegenden Fall der Spezialisierung auf eine bestimmte Domäne mit spezifischem Vokabular, Strukturen und Mustern helfen, indem Beschreibungen und Beispiele zur Verfügung gestellt werden, sodass die Lücke fehlender, nicht übereinstimmender oder unbekannter Begriffe geschlossen werden kann (Kanburoglu & Tek, 2023, S.406; Zuckarelli, 2025, S.53-54).

In der vorliegenden Arbeit soll deshalb geprüft werden, ob Kontext in der ausgewählten Form und in welchem Umfang für die spezifische Domäne Publishing hilfreich ist.

### 1.2.Prompting

Prompt Engineering als Teilbereich des In Context Learning (ICL) dient dazu, dem Sprachmodell umfassende Anweisungen zur Aufgabe zu geben und ist ein wichtiger Faktor, um das Potenzial von LLMs freizusetzen und präzise Ergebnisse zu erzielen, ohne das Modell an sich neu/weiter trainieren zu müssen. Dabei beeinflusst die Wahl der Strategie die Leistung des Modells, wobei vor allem bei komplexen Abfragen messbare Unterschiede erwartet werden können (Zhu et al., 2024, S.9-10; Hong et al., 2025, S.6; Yan et al., 2025, S.4). ICL

ist menschlichem Lernen aus Analogien ähnlich. Von Vorteil ist dabei, dass die Anweisungen in natürlicher Sprache verfasst und somit die Interaktion interpretierbar und veränderbar ist. Menschliches Wissen kann auf einfache Weise einbezogen werden. Es sind zudem kein Finetuning oder Parameter-Aktualisierungen nötig, sodass Rechenkosten für Anpassungen an neue Aufgaben erheblich reduziert werden (Xi et al., 2025, S.9).

Prompt Engineering adressiert die Herausforderung c) komplexe und seltene SQL-Operationen generieren zu lassen. Für diese Arbeit wurden die Strategien One Shot und Chain of Thought als relevant identifiziert.

Beim One Shot (OS) / Few Shot Prompting wird dem Modell eine Frage mit einem oder mehreren Fallbeispielen gestellt. Die Antwort des Modells soll auf Grundlage der Beispiele formuliert werden – dies führt zu einem besseren Verständnis der Aufgabe und kann die gesamte Leistung gegenüber zero shot (Frage ohne Beispiele) insbesondere bei komplexen SQL-Aufgaben verbessern (Zhu et al., 2024, S.10; Zuckarelli, 2025, S.56-57; Boonstra, 2024, S.14).

Beim Chain of Thought Prompting (COT) wird das Modell angeleitet, vor Generierung der finalen SQL zunächst zu argumentieren, und ein Problem in eine Reihe logischer Zwischenschritte zu zerlegen, ähnlich wie ein Mensch ein Problem durchdenken würde (Kansal, 2024, S.146; Boonstra, 2024, S.29). Als einfacher Einstieg in COT Prompting können bereits zwei Schritte hilfreich sein, indem man das Modell zunächst anleitet, einen Lösungsansatz in natürlicher Sprache und dann darauf basierend Code zu entwickeln (Zuckarelli, 2025, S. 64-65). Standard COT Prompts zeigen lt. Hong et al. (2025, S.9) begrenzte Effektivität für Text-to-SQL - zusätzliche Erweiterungen bzw. die Kombination von Strategien können diese Lücke aber schließen.

In der vorliegenden Arbeit soll nun geprüft werden, welche der beiden Prompting-Strategien für unterschiedlich komplexe SQL besser geeignet ist. Wobei es Hinweise darauf gibt, dass Few-Shot Prompting für einfache, COT Prompting für komplexe Abfragen besser geeignet sein könnte (mit der Bedingung, dass COT mit Decomposition kombiniert wird (Pourreza & Rafiei, 2023, S.6 und S.10). Diese Differenzierung ist zusätzlich aufgrund unterschiedlich erwarteter Token-Längen und dementsprechenden Kosten relevant. Few-Shot verbraucht Token mit zunehmendem Umfang der Beispiele, weshalb in dieser Arbeit nur ein Beispiel gegeben, also One Shot Prompting, angewendet wird. COT benötigt im Verhältnis meist noch mehr Token aufgrund der einzelnen Zwischenschritte und Argumente (Martra, 2024, S.25; Zuckarelli, 2025, S.65; Boonstra, 2024, S.29).

### 1.3.Retry

Retry stellt in der vorliegenden Arbeit die Interpretation und Implementierung der in der Literatur als *Execution Refinement* (Mohammadjafari et al., 2025, S.8), *Feedback-gesteuerte* Regenerierung (Hong et al., 2025, S.10) oder *Refiner* (Wang et al., 2025, S.4) benannten Schritte dar. Allen Methoden gemein ist die iterative Verfeinerung der Abfrage basierend auf dem Ausführungsergebnis. Auf diese Weise können Performance und Korrektheit optimiert und Ausführungsfehler minimiert werden (Mohammadjafari et al., 2025, S.8; Hong et al., 2025, S.10), sodass potentiell ungültige Abfragen doch noch in valide Abfragen umgewandelt werden können. Die Methode ist von der Zugänglichkeit der Datenbank abhängig (Hong et al., 2025, S.12).

In der vorliegenden Arbeit soll geprüft werden, ob eine Retry-Option insbesondere zur Lösung komplexer SQL-Abfragen hilfreich sein kann, was ebenso wie Prompting Herausforderung c) komplexe und seltene SQL-Operationen generieren zu lassen, adressiert.

### 1.4.Forschungsfrage und Hypothesen

Hergeleitet aus den vorherigen Darstellungen zu Kontext, Prompting und Retry ergibt sich die Forschungsfrage: Wie beeinflussen die Komponenten Kontext, Prompting und Retry eines LLM-basierten Text-to-SQL-Analysten die Qualität der generierten SQL-Abfragen gemessen an Ausführbarkeit (Valid SQL), Ergebnisgenauigkeit (Execution Accuracy) und semantischer Übereinstimmung (Exact Match) bei Ausführung auf einer Unternehmensdatenbank?

Auch die folgenden Hypothesen lassen sich aus den vorherigen Ausführungen ableiten:

- Hypothese 1: Die Einbindung eines angereicherten Schemas (bzgl Tabellen und Attributen), über das relevante Tabellen und Spalten identifiziert werden können, erhöht insbesondere bei komplexen Abfragen sowohl die Ausführbarkeit als auch die Ergebnisgenauigkeit, da korrekte Verknüpfungen mit der zugrunde liegenden Datenstruktur wahrscheinlicher werden.
- Hypothese 2: OS-Prompting ist einem Standard COT-Prompting ohne zusätzliche Maßnahmen bzgl. semantischer Übereinstimmung (EM) insbesondere bei einfachen Abfragen überlegen, da Aufgabenbeispiele im Bereich Text-to-SQL ausreichen, damit das LLM die Aufgabe, Abfragen zu generieren, gut versteht.
- Hypothese 3: Die Implementierung eines Mechanismus zur Fehlerkorrektur und iterativen Verfeinerung bei Ausführungsfehlern trägt insbesondere bei komplexen Abfragen zur Verbesserung von Ausführbarkeit und Ergebnisgenauigkeit bei, da unvollständige oder fehlerhafte SQL-Abfragen gezielt überarbeitet werden können.

# 2.Daten und Methoden

Im folgenden Kapitel wird zunächst das Forschungsdesign und die Methode erläutert. Es folgt die Vorstellung des Workflows bzw. Analysten und der einzelnen Knoten. Daran anschließend werden Datenbank, Testfragen und Grundwahrheit beschrieben. Zum Ende wird auf die Evaluationsmetriken eingegangen.

## 2.1.Methode und Design

Die Arbeit verfolgt ein experimentell-quantitatives Forschungsdesign.

Der geplante Analyst basiert auf dem LangGraph-Framework und wird modular aufgebaut, sodass jede Komponente als eigener Knoten oder Kombination von Knoten im Workflow abgebildet ist. Anhand eines Sets von NL-Testfragen, wird der Analyst in fünf verschiedenen Varianten auf einer realen Unternehmensdatenbank in BigQuery ausgeführt. Die dabei erzeugten SQL-Abfragen und Ergebnisse werden anhand der Metriken Valid SQL, Execution Accuracy und Exact Match im Vergleich zur Grundwahrheit bewertet.

Kern des Vorgehens ist eine Ablation Study: Es werden einzelne Komponenten aktiviert bzw. gegeneinander ausgetauscht, um deren Wirkung auf die definierten Metriken isoliert zu analysieren - orientiert am Vorgehen in diversen Studien (z.B. Pourreza & Rafiei, 2023, S.9; Yan et al., 2025, S.19; Zhang et al., 2023, S.670). Die NL-Fragen werden in zwei Formulierungsstilen und abzielend auf zwei Komplexitätsstufen für die erwartete SQL entwickelt. Jede Frage wird zehn Mal an jede Analyst-Variante gestellt, um die statistische Zuverlässigkeit der Ergebnisse zu erhöhen.  Zusätzlich zur Ablation Study wird eine Fehleranalyse vorgenommen, um Schwächen der Analysten sichtbar zu machen.

## 2.2. Workflow und einzelne Knoten in LangGraph

Der Workflow folgt den in Abb.1 gezeigten Knoten. Oben werden die jeweiligen Dateien genannt, auf die von den Knoten zugegriffen wird. Das LLM-Männchen zeigt an, ob in einem Knoten ein Sprachmodell angebunden ist. Die Schleife von „analyze error“ ist optional und wird nur genutzt, wenn bei einer Abfrage eine Fehlermeldung generiert wird.

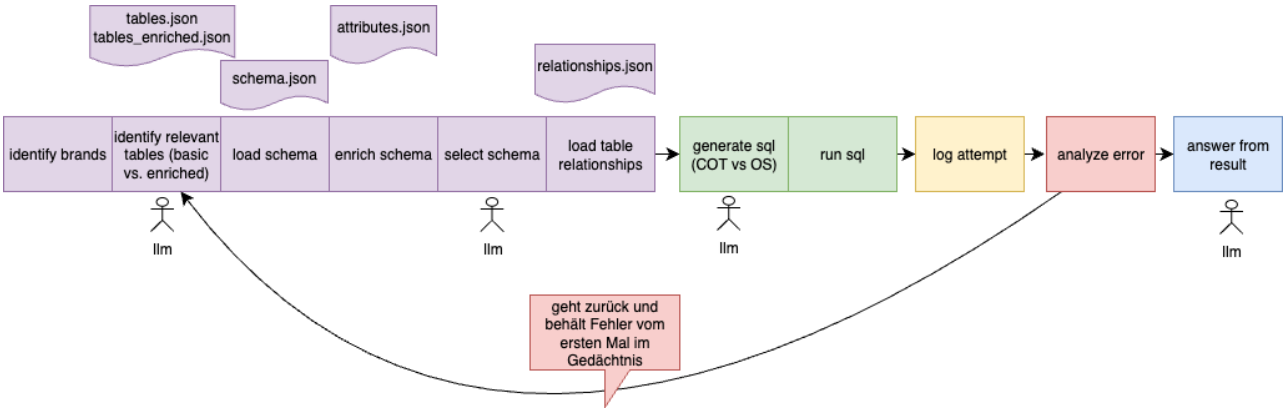



*Abbildung 1: Übersicht Workflow*

Die Abbildung zeigt alle Knoten, die möglich sind. In den verschiedenen Versionen der Analysten sind aber nicht alle Knoten angebunden, entsprechend dem Vorgehen in einer Ablation Study (Workflows jedes Analysten im Detail siehe Anhang 6.1).

Der Analyst bzw. Workflow wird mithilfe von LangGraph entwickelt. LangGraph ist ein Framework, welches die Modellierung von Workflows als Graphen ermöglicht. Das Verhalten des Systems wird dabei nach Langgraph (2025) über die drei Schlüsselkomponenten Zustand (*State*), Knoten (*Nodes*) und Kanten (*Edges*) definiert. Der Zustand ist eine von allen Knoten und Kanten gemeinsam genutzte Datenstruktur, die den aktuellen Arbeitsstand des Workflows darstellt. Dies kann z.B. ein Datentyp sein, der mithilfe eines ebenfalls gemeinsam genutzten Zustandsschemas definiert wird. Knoten enthalten Funktionen, die die Logik des Workflows codieren. Sie empfangen den Zustand als Eingabe, führen z.B. Berechnungen durch und geben einen aktualisierten Zustand zurück. Kanten sind Funktionen, die anhand des aktuellen Zustands bestimmen, welcher Knoten als nächstes ausgeführt werden soll. Dabei kann es sich um eine bedingte Verzweigung oder feste Übergänge handeln. Nachfolgend werden die einzelnen Knoten (und eine optionale Kante) 1. bis 11. beschrieben. Der jeweils zugehörige Code kann im Repository im Ordner /agent/core/nodes eingesehen werden.

1. identify brands: Identifiziert aus der eingegebenen Frage regelbasiert die Marken, um die es gehen soll. Wird in der Frage keine Marke genannt, werden alle Marken im State gespeichert. (Anmerkung: Markennamen sind Teil der Tabellennamen.)
2. identify relevant tables: LLM wählt entsprechend der Frage Tabellen aus, eine basic vs. ausführliche Beschreibung der Tabellen werden im Prompt via JSON-Datei zur Verfügung gestellt (einzusehen im Repository im Ordner /agent/core/context).Tabellennamen werden im State gespeichert.
3. load schema: Lädt regelbasiert das Schema zu den ausgewählten Tabellen. Enthalten sind Name und Datentyp jedes Attributs (einzusehen im Repository im Ordner /agent/core/context). Das Schema wird im State gespeichert.
4. enrich schema: Lädt regelbasiert zusätzliche Informationen (Beispielwerte, semantische Beschreibung) zu den Attributen in den ausgewählten Tabellen (einzusehen im Repository im Ordner /agent/core/context). Hier werden wenige, als grundsätzlich unwichtig definierte Attribute, vorweg regelbasiert aussortiert. Das angereicherte Schema wird im State gespeichert.

5. select schema: LLM wählt idealerweise passend zur Frage maximal 15 Attribute pro Tabelle anhand des angereicherten Schemas. Das fokussierte und angereicherte Schema wird im State gespeichert.
6. load table relationships: Lädt regelbasiert zu den in „identify relevant tables“ ausgewählten Tabellen Informationen zu möglichen Joins. Die Join-Informationen werden im State gespeichert.

Alle zuvor genannten Knoten tragen zum Aufbau von Kontext bei. Es werden Informationen z.T. regelbasiert und z.T. durch ein LLM jeweils aus extern gehaltenen JSON-Dateien, die dynamisch in die LLM-Prompts eingefügt werden, ausgewählt. So kann aktiv und intelligent gefiltert werden. Die Auswahl des zugeführten Kontextes geschieht in mehreren Schritten, um die Komplexität der Schema-Verknüpfung aufzubrechen und die jeweilige Token-Anzahl einzugrenzen. Final erhält das LLM im nun folgenden Knoten die ausgewählten Tabellennamen, die ausgewählten Attributnamen und zu diesen jeweils den Datentyp und Beispielwerte wie von Martra (2024, S.249 ff.) diskutiert, sowie zusätzlich je Attribut eine semantische Beschreibung, angelehnt an die Idee von „*LLM-friendly descriptions*“ von Nascimento et al. (2024, S.3).

7. generate sql: LLM erhält die ausgewählten Schema- und Join-Informationen aus den vorigen Schritten und generiert mithilfe von COT- oder OS-Prompting eine SQL-Abfrage passend zur NL-Frage. Die Abfrage wird im State gespeichert.

Die Prompts für diesen und die anderen Knoten mit LLM-Anbindung wurden entwickelt anhand der Beschreibungen von Boonstra (2024) zu OS-Prompting (ab S.14), zu COT-Prompting (ab S.29) sowie zu System- und Rollen-Prompting (ab S.17).

8. run sql: führt die generierte SQL-Abfrage auf Big Query aus, Abfrage-Ergebnis wird im State gespeichert
9. log attempt: loggt Daten für die Evaluation
10. analyze error: eigentlich eine optionale Kante, die mit zwei Hilfsfunktionen gesteuert wird. Tritt eine Fehlermeldung als Antwort auf die Abfrage auf, wird ein erneuter Einstieg in „identify relevant tables“ möglich. Bei gültigem Ergebnis geht es weiter zu „answer“. Bei einer Wiederholung wird in allen Knoten mit LLM eine „Retry-Notiz“ aktiv, die unter anderem die erste generierte Abfrage und die Fehlermeldung enthält.

Die Möglichkeit nach einer Fehlermeldung den Workflow noch einmal zu durchlaufen, wurde orientiert an Vorbildern aus der Literatur entwickelt (Mohammadjafari et al., 2025,

S.8; Hong et al., 2025, S.10; Wang et al., 2025, S.4). Eine Abfrage, die ein leeres Ergebnis liefert, durchläuft ebenfalls erneut den Workflow wie von Wang et al. (2018, S.2) vorgeschlagen.

11. answer from result: erhält das Ergebnis der Abfrage und gibt eine Antwort auf die Frage in natürlicher Sprache, Antwort wird im State gespeichert

Als Sprachmodell wurde ChatGPT 4.0 angebunden. Diese Wahl wurde vorrangig aus unternehmensinternen Gründen getroffen, wird aber auch durch Studienergebnisse gestützt (z.B. Narasimhan et al., 2024, S.4; Pourreza & Rafiei, 2023, S.2; Wang et al., 2025, S.1; Ojuri et al., 2025, S.9).

### 2.3.Aufbau der Datenbank

Die für das Projekt genutzte Datenbank stellt einen Ausschnitt aus der realen Datenbank des Unternehmens der Autorin dar. Alle Tabellen sind jeweils für die drei Marken ___ und die Monate Juni, Juli, August vorhanden – insgesamt sind 18 Tabellen eingebunden. Die gängigen Benchmarks SPIDER und BIRD enthalten in ihren größten Datenbanken SPIDER 16 bis 25 Tabellen und BIRD 13 bis 30 Tabellen (Oliveira (2025), S.279-280).

Die Tabellen agg_* und rep_* enthalten Rohdaten (agg_*) oder aggregierte Reporting-Daten (rep_*) mit jeweils unterschiedlichem Fokus (z.B. Nutzende, Sitzungen, Inhalte, Treffer) – teilweise finden sich hier ähnliche Attribute wieder. Die Daten stammen aus dem Tracking, welches mittels Google Analytics 4 auf den Publishing-Inhalten stattfindet.

Die Tabellen csp_content_* enthalten Daten, die aus dem Verwaltungssystem der Inhalte stammen.

Eine genaue Tabellenbeschreibung, sowie die Schemata und Beschreibung der Attribute befinden sich im Repository im Ordner /agent/core/context.

### 2.4.Testfragen, Grundwahrheit

Die Fragen für die Tests wurden passend zur Datenbank gestaltet, wobei differenziert wird nach Fragenformulierung und erwarteter Komplexität der SQL.

Fragen simple vs. complex zielen auf dasselbe SQL-Ergebnis ab, werden aber unterschiedlich formuliert. Dies stellt eine Erweiterung zum Vorgehen in gängigen Benchmarks dar, denn hier sind die NL-Fragen lt. Oliveira et al. (2025, S. 279-280) rein auf Grundlagen von Begriffen formuliert, die in den Datenbankschemata verwendet werden, was Ergebnisse verzerren kann. Zudem zielt die unterschiedliche Formulierung in Erweiterung zum Einsatz von Kontext darauf ab, zu testen wie gut die verschiedenen Analysten mit Herausforderung a) sprachlicher Mehrdeutigkeit und linguistischer Komplexität zurechtkommen. Es ist pro

SQL-Abfrage eine einfache und eine komplexe Frage vorhanden – insgesamt sind es 60 Fragen.

- simple: Frage formuliert, wie sie jemand mit Kenntnis der Tabellen und Attribute stellen würde; klar strukturiert, mit Hinweisen, Attributnamen in deutscher Entsprechung oder Englisch, sowie Werte für Filter sind korrekt benannt.
- complex: Frage formuliert, wie sie jemand ohne Kenntnis der Tabellen und Attribute stellen würde; Namen verwässert, keine Hinweise, seltener direktes verknüpfen auf das Datenbankschema möglich.

Die erwartete SQL simple vs. complex zielt auf unterschiedlich komplexe Abfragen ab – dies adressiert nochmals die grundsätzliche Herausforderung c), dass es grundsätzlich weiter Probleme gibt, seltene und komplexe SQL generieren zu lassen und entspricht auch dem Vorgehen in gängigen Benchmarks, wobei SPIDER eine Einordnung in vier Stufen vornimmt, Mondial in drei Stufen (Oliveira et al., 2025, S.279 und S.285). Es sind 15 einfache und 15 komplexe Abfragen vorhanden.

- simple: konkrete Abfragen von eindeutigen Dimensionen und Metriken mit einfachen Operationen wie WHERE-Klauseln, GROUP BY oder ORDER BY
- complex: enthalten z.B. Joins über zwei Tabellen, Aggregationen, window functions, verschachtelte Sub-Abfragen

Alle Fragen und ihre zugehörigen SQL-Abfragen wurden in Anlehnung an Analysefragen, die in den letzten Jahren an das Team der Autorin herangetragen wurden, entwickelt – entsprechen also in etwa den Anforderungen, die auch in der Realität an den Analysten gestellt werden würden.

In der Grundwahrheit (im Detail im Repository im Ordner /agent/notebooks/input) sind jeweils die zwei Fragen, die ideale SQL, das ideale Ergebnis, sowie zusätzlich Einzelteile der idealen SQL gespeichert: die erwarteten Tabellen, die erwarteten Attribute und die idealen SQL-Operationen.

Für die Evaluation werden die generierte SQL, die Anzahl der Versuche (bei Analysten mit Retry Option), der grundsätzliche Erfolg der Ausführung, sowie das Ergebnis geloggt.

So kann in der Evaluation entsprechend der zu erhebenden Metriken abgeglichen werden zwischen generierten Ergebnissen des Analysten und der Grundwahrheit.

### 2.5.Metriken

Die anzuwendenden Metriken werden wie folgt definiert (Code im Notebook evaluation.ipynb, Zeilen 12 bis 16, 35 bis 39):

- Valid SQL (VA) oder auch „Ausführbarkeit“: Bewertet, ob eine Abfrage grundsätzlich gültig und somit ausführbar ist. **VA_rate** wird errechnet aus der Anzahl execution_succes = True, geteilt durch die Gesamtzahl an Abfragen (siehe auch Ojuri et al., 2025, S.5).
- Exact Match (EM) oder auch „semantische Übereinstimmung“: Bewertet die Abfrage an sich und ob sie inhaltlich mit der erwarteten Abfrage übereinstimmt. Es wird hier eine Mischung aus den in der Literatur vorgeschlagenen Metriken Component Matching (CM) und Exact Matching (EM) angewendet. Für CM wird bewertet, ob alle erwarteten Komponenten wie z.B. SELECT, WHERE, GROUP BY in der generierten Abfrage vorhanden sind, unabhängig von der Reihenfolge. EM bewertet, ob die generierte Abfrage komplett bzgl. Elementen, Struktur und Reihenfolge der Grundwahrheit entspricht (Mohammadjafari et al., 2025, S.6; Hong et al., 2025, S.6). In dieser Arbeit wird, bezeichnet als **EM_share,** für jede generierte SQL ein Mittelwert des Anteils an mit der Grundwahrheit-SQL übereinstimmenden Tabellen, Attributen, SQL-Operationen (entspricht Komponenten von CM), Datum, Sortierrichtung und Begrenzung berechnet – unabhängig von der Reihenfolge.

  **EM_rate** ist der binäre Check auf komplette Übereinstimmung aller definierten Elemente, errechnet aus der Anzahl von EM_share = 1 geteilt durch die Anzahl aller generierten Abfragen.

  Das folgende Beispiel anhand Abfrage q02 (Grundwahrheit im Repository im Ordner /agent/notebooks/input) soll das Vorgehen veranschaulichen.

  Von Analyst A wurde die folgende Abfrage mit der session_id 2025-09-24T08:31:57.594617 generiert (Ausführungsergebnisse im Repository im Ordner /agent/notebooks/output): « `SELECT DATE_TRUNC(date, MONTH) as month, SUM(CASE WHEN LOWER(event_name) = 'video_wb_started' THEN events ELSE 0 END) AS video_wb_started,SUM(CASE WHEN LOWER(event_name) = 'video_wb_ended' THEN events ELSE 0 END) AS video_wb_ended,SAFE_DIVIDE(SUM(CASE WHEN LOWER(event_name) = 'video_wb_ended' THEN events ELSE 0 END), SUM(CASE WHEN LOWER(event_name) = 'video_wb_started' THEN events ELSE 0 END)) AS ratio_ended_started FROM `bachelor_mlh.agg_ga4_hits_ __ _web` WHERE LOWER(operating_system) = 'android' AND LOWER(display) = 'app' AND date BETWEEN '2025-06-24' AND '2025-09-24'GROUP BY month ORDER BY month` »

  Im Vergleich mit der Grundwahrheit wurde die Tabelle korrekt gewählt (=1), die Attribute korrekt gewählt (=1), von den Operationen 9 von 14 korrekt gewählt (= 0.64), das Datum nicht korrekt gewählt (=0), keine Begrenzung korrekt gewählt (=1), die Sortierrichtung korrekt gewählt (=1). EM_share = 4.64 / 6 = 0.77

- Execution Accuracy (EA) oder auch „Ergebnisgenauigkeit“: Bewertet das Ergebnis einer ausgeführten Abfrage. **EA_rate** ist ein binärer Check, ob alle Elemente des generierten Ergebnisses nach Ausführung der Abfrage auf die Datenbank, mit allen Elementen der Grundwahrheit übereinstimmen, unabhängig von der Sortierung und Reihenfolge. Es wird die Anzahl von result_share = 1 (bzw. True) gezählt und durch die Gesamtzahl an Ergebnissen geteilt (Ojuri et al., 2025, S.5; Mohammadjafari et al., 2025, S.6; Hong et al., 2025, S.6; Shi et al., 2025, S.6; Wang et al., 2025, S.5; Pourreza & Rafiei, 2023, S.7).

  Zusätzlich wird **EA_share** aus der Anzahl übereinstimmender Elemente zwischen generierten Ergebnissen und Grundwahrheit berechnet, geteilt durch die Gesamtzahl erwarteter Elemente im Ergebnis. EA_share ist aus der Beobachtung entstanden, dass nur wenige Ergebnisse komplett als übereinstimmend markiert werden können, obwohl durchaus auch richtige Werte ausgegeben werden. EA_share macht diesen Anteil sichtbar.

  Das von Analyst A zur Abfrage q02 in der session 2025-09-24T08:31:57.594617 generierte Ergebnis soll analog zu EM_share auch hier das Vorgehen veranschaulichen.

  Das Ergebnis lautet (im Repository im Ordger /agent/notebooks/output):

  ```
  «2025-06-01, 36975, 16820, 0.4549019607843137 (wird gerundet auf 0.5)
  2025-07-01, 198119, 102425, 0.5169872652294832 (gerundet 0.5)
  2025-08-01, 13253, 6079, 0.4586885988078171" (gerundet 0.5) »
  ```

  Die Grundwahrheit zur Abfrage lautet (im Repository im Ordner /agent/notebooks/input):

  ```
  2025-06, app, Android, 156689, 80083, 0.51109522685064046 (gerundet 0.5)
  2025-07, app, Android, 198119, 102425, 0.51698726522948324 (gerundet 0.5)
  2025-08, app, Android, 13253, 6079, 0.45868859880781709" (gerundet 0.5)
  ```

  18 Ergebnis-Elemente sollen vorkommen, 7 stimmen überein. EA_share= 7 / 18 = 0.39

Zwischen den Metriken besteht ein kaskadierender Effekt. Nur gültige Abfragen, die keine Fehlermeldung ausgelöst haben, liefern ein Ergebnis, dessen einzelne Elemente über die Ergebnisgenauigkeit bewertet werden können. Die Abfrage an sich wird unabhängig davon, ob ausführbar oder nicht, auf ihre inhaltliche Übereinstimmung einzelner Elemente mit der Abfrage-Grundwahrheit geprüft. Je mehr Abfrage-Elemente vom LLM korrekt gewählt wurden, desto wahrscheinlicher ist es, dass auch die Elemente des Ergebnisses mit der Ergebnis-Grundwahrheit übereinstimmen, also eine hohe Ergebnisgenauigkeit erwartet werden kann. Die oben genannten Beispiele zeigen andeutungsweise einige Schwierigkeiten in der Bewertung der semantischen Übereinstimmung und Ergebnisgenauigkeit. So wurden im Beispiel der Abfrage nur ein Teil der Operationen korrekt gewählt, dennoch sind nachfolgend richtige Ergebnis-Elemente vorhanden. Es gibt also womöglich nicht die eine ideale SQL-Abfrage,

sondern es sind manchmal Alternativen möglich. In der gezeigten Abfrage ist der zentrale Fehler, dass das Datum nicht korrekt gewählt wurde, sodass die Daten für den Monat Juni nicht stimmen. Zudem gab es ein Missverständnis zum Umfang der Ergebnis-Ausgabe. Es fehlen Ergebnis-Elemente, die in der Grundwahrheit gefordert sind, aber das Ergebnis bzgl. der reinen Zahlen nicht korrekter machen, sondern nur vollständiger für das Verständnis. Der Umfang eines Abfrage-Ergebnisses kann dementsprechend abweichen, was die Ergebnis-Werte nicht falsch macht, aber die Ergebnisgenauigkeit dennoch negativ beeinflusst.

# 3.Ergebnisse

Aus der im vorigen Kapitel beschriebenen Vorgehensweise resultieren die nachfolgenden Analyse-Ergebnisse, welche im Notebook evaluation.ipynb detailliert zu finden sind (im Repository im Ordner /agent/notebooks). Zunächst wird entsprechend der Forschungsfrage untersucht, welchen Einfluss die Komponenten Kontext, Prompting und Retry auf Ausführbarkeit, Ergebnisgenauigkeit und semantische Übereinstimmung haben. Es folgt ein Vergleich über alle Analysten und eine Analyse der gefundenen Fehler. Abschließend wird auf die Auswirkungen der Fragenformulierung und SQL-Komplexitätsstufen auf die Zielmetriken eingegangen.

Zur Beurteilung der Komponente **Kontext** werden die Analysten A und C miteinander verglichen – beide nutzen OS Prompting, A nutzt fokussierten und angereicherten Kontext, C erhält ein Basic Schema zu den ausgewählten Tabellen, welches nicht auf wichtige Attribute reduziert wurde (Details zu den Analysten siehe Anhang 6.1 und Kapitel 2.2). Im Vergleich ergibt sich, dass angereicherter Kontext für die Ausführbarkeit signifikant wichtig ist (Signifikanztest siehe evaluation.ipynb, Codezeile 59). Mit angereichertem Kontext (Analyst A) werden 78% ausführbare Abfragen generiert, während mit wenig Kontext (Analyst C) nur 70% ausführbare Abfragen generiert werden (evaluation.ipynb, Codezeile 55-56). Dies zeigt sich insbesondere für komplexe SQL (Abb.2). Hier generiert Analyst A mit angereichertem Kontext zwischen 63 bis 73% ausführbare Abfragen, Analyst C mit wenig Kontext 14 bis 19% weniger. Die angenommene Verbesserung der Ergebnisgenauigkeit durch Kontext ist nicht signifikant, aber als Tendenz zu erkennen für komplexe Abfragen. So können mit angereichertem Kontext (Analyst A) in der Kombination complex/simple(q) 11% korrekte Ergebnisse generiert werden, mit wenig Kontext (Analyst C) dagegen nur 7%. Im Gegensatz dazu geht für einfache Abfragen die Tendenz zu wenig Kontext. In der Kombination simple/simple(q) werden mit wenig Kontext (Analyst C) 41% genau übereinstimmende Ergebnisse generiert, mit angereichertem Kontext (Analyst A) 6% weniger (Abb.3).

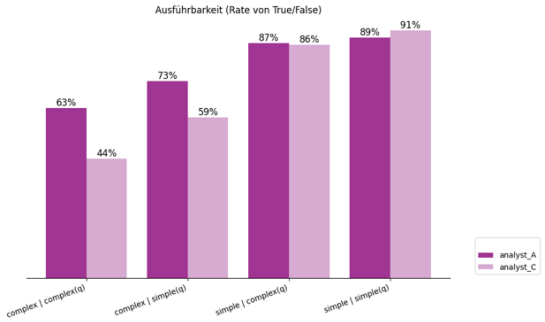



*Abbildung 2: Kontext - Valid SQL*

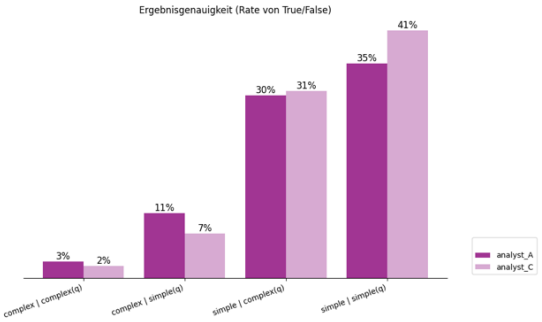



*Abbildung3: Kontext - Execution Accuracy*

Für die Komponente **Prompting** werden die Analysten A und B miteinander verglichen. Beide nutzen die volle Kontext-Komponente, A nutzt den OS-Prompt, B nutzt den COT-Prompt (Details zu den Analysten siehe Anhang 6.1.). COT-Prompting (Analyst B) liefert im Vergleich zu OS-Prompting (Analyst A) signifikant häufiger genaue Ergebnisse in allen Frage-/SQL-Kombinationen, insbesondere aber bei einfachen SQL-Abfragen mit einer Rate von 39 bis 49% (Abb.4) und 9 bis 14% häufiger als OS-Prompting. OS-Prompting (Analyst A) liefert dagegen bei Ausführbarkeit und semantischer Übereinstimmung grundsätzlich signifikant bessere Werte (Signifikanztest siehe evaluation.ipynb, Codezeile 53). Betrachtet man die Werte zur semantischen Übereinstimmung aufgefächert nach Fragestellung und Komplexität der SQL, zeigt sich jedoch, dass OS-Prompting nur bei einfacher SQL deutlich besser abschneidet als COT (31% vs. 25% bis 19%). Als schwierig eingestufte Abfragen werden bei einfacher Fragestellung in 4% der Fälle durch COT-Prompting semantisch übereinstimmend generiert. Inhaltlich übereinstimmende Abfragen der Kombination complex/complex(q) konnten weder durch COT noch durch OS-Prompting generiert werden (Abb.5). Man könnte annehmen, dass sich das Ergebnis in sich widerspricht – richtige Ergebnisse sollten doch durch richtige Abfragen generiert worden sein. Durch OS-Prompting werden etwa so viele richtige Ergebnisse wie richtige Abfragen generiert, bei COT-Prompting sind es deutlich weniger richtige Abfragen als richtige Ergebnisse. Es wurde während der Tests beobachtet, dass für die Abfrage in gewissem Rahmen Abweichungen von der idealen Lösung möglich sind, die dennoch dasselbe Ergebnis z.B. in anderer Sortierung oder Reihenfolge ergeben. Sortierung und Reihenfolge werden bei der Ergebnisbeurteilung allerdings nicht beachtet, entsprechende Operationen gehören aber zur vollständigen semantischen Übereinstimmung dazu.

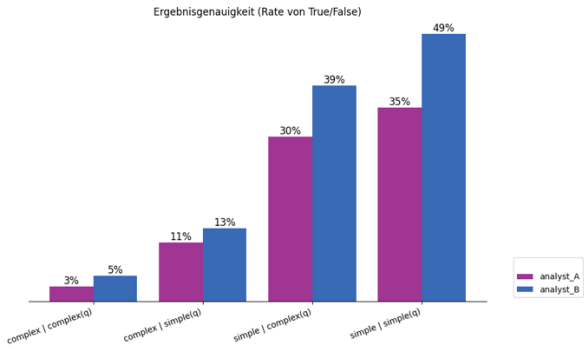



*Abbildung 4: Prompting - Execution Accuracy*

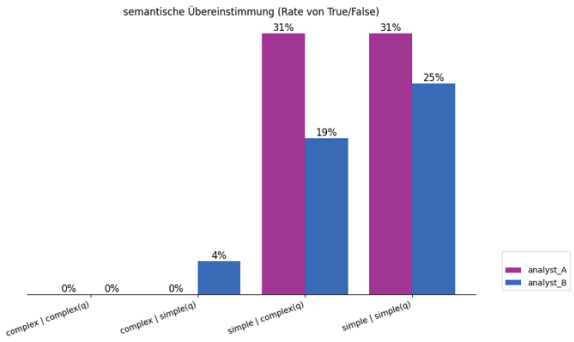



*Abbildung 5: Prompting - Exact Match*

Für die Komponente **Retry** werden die Analysten B und E miteinander verglichen, beide nutzen COT-Prompting und den vollen Kontext, E hat die Option auf einen Retry, B kann die Option nicht nutzen (Details zu den Analysten siehe Anhang 6.1.). Ein Retry findet nur statt, wenn eine Fehlermeldung oder ein leeres Ergebnis vorliegen. Der komplette Workflow wird mit einer Retry-Notiz für die Knoten mit LLM noch einmal durchlaufen, sodass die Sprachmodelle die Möglichkeit bekommen, anhand der Fehlermeldung die erste Abfrage noch einmal zu verfeinern. Diese Möglichkeit verbessert signifikant die Ausführbarkeit und die Ergebnisgenauigkeit (Signifikanztest siehe evaluation.ipynb, Codezeile 65) und zeigt sich sowohl für einfache als auch für komplexe SQL und beide Fragenformulierungen durchgängig mit 81 bis 97% ausführbaren Abfragen (Abb.6) und 2% bis 4% höherer Ergebnisgenauigkeit als ohne Retry (Abb.7). Auf die semantische Übereinstimmung hat die Retry-Option dagegen keinen signifikanten Effekt. Dies könnte so interpretiert werden, dass in den Wiederholungen nur wenige Änderungen vorgenommen werden, sodass sich das in der Bewertung der einzelnen Abfrage-Elemente für die semantische Übereinstimmung nicht bemerkbar macht.

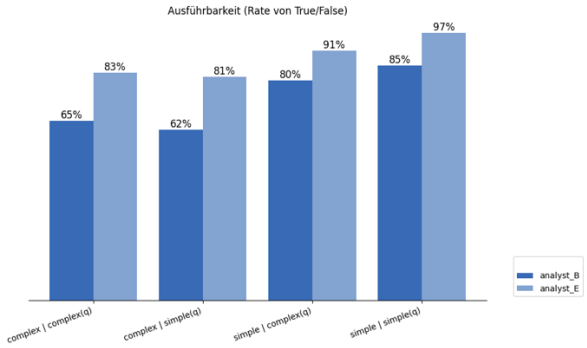



*Abbildung 6: Retry - Valid SQL*

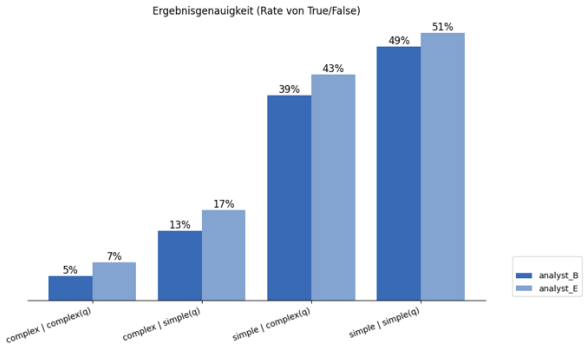



*Abbildung 7: Retry - Execution Accuracy*

Nachfolgend wird der Vergleich aller Analysten gezeigt sowie eine detaillierte Fehleranalyse vorgenommen, um die niedrigen Raten der semantischen Übereinstimmung und Ergebnisgenauigkeit zu ergründen.

Alle Analysten liegen bzgl. der Ausführbarkeit bei Werten über 70%. Analyst E (Kontext, COT, Retry) schneidet mit 88% ausführbaren Abfragen am besten ab. Die durchschnittlichen Werte für die Ergebnisgenauigkeit (EA_rate) und semantische Übereinstimmung (EM_rate) liegen deutlich niedriger: Ergebnisgenauigkeit zwischen 20 bis 30%, semantische Übereinstimmung zwischen 12 bis 18% (siehe Abb.8). Da die Bewertung der Abfrage und Ergebnisse sich, wie vorher schon besprochen, aus der Beurteilung vieler einzelner Elemente zusammensetzt, liegt der Gedanke nahe, zu schauen, ob zumindest anteilig von den Analysten richtige Elemente für die Abfrage gewählt und nachfolgend im Ergebnis generiert wurden. Hierzu können die Metriken EM_share und EA_share herangezogen werden (Abb.9).

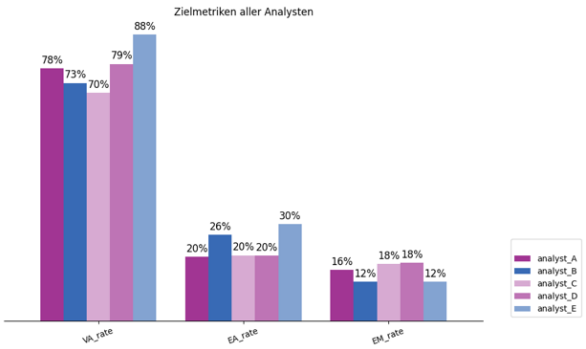



*Abbildung 8: Zielmetriken aller Analysten*

| analyst | EM_share | EA_share |
|---|---|---|
| analyst_A | 0.873985 | 0.33600 |
| analyst_B | 0.825972 | 0.42050 |
| analyst_C | 0.88073 | 0.33150 |
| analyst_D | 0.877246 | 0.35175 |
| analyst_E | 0.829456 | 0.48400 |

*Abbildung 9: Anteilige Berechnung EA und EM aller Analysten*

Im Durchschnitt stimmen zwischen 82 bis 88% der Elemente der generierten Abfragen mit den jeweils erwarteten Abfragen überein (Spalte EM_share). Das bedeutet, das im Durchschnitt nur wenige Elemente falsch gewählt wurden oder nicht in der Abfrage enthalten waren. Für die anteilige Ergebnisgenauigkeit (EA_share) liegen die Werte zwischen 33 bis 48%. Das heißt, im Durchschnitt stimmen ein Drittel bis die Hälfte der Elemente eines generierten Ergebnisses mit dem erwarteten Ergebnis überein.

Nachfolgend werden die Elemente, die zur Erniedrigung der Rate der semantischen Übereinstimmung beitragen, angeschaut. In 78% der Fälle wird das Datum korrekt gesetzt. Es haben aber die Analysten mit COT-Prompting mehr Probleme mit dem Datum als die Analysten mit OS-Prompting: 60% vs. 90%. Die Wahl der Attribute fällt allen Analysten schwer – diese werden durchschnittlich in nur 65% der Fälle komplett korrekt gewählt (evaluation.ipynb, Codezeile 26 – 27). Hier kann auf eine ungenügende Schemaverknüpfung innerhalb der Kontext-Knoten und bei der Generierung der SQL geschlossen werden. Die

niedrigste Rate wird für die Wahl der Operationen erreicht – nur 22% der Fälle enthalten im Durchschnitt alle idealen Operationen. Das heißt, was für die semantische Übereinstimmung jeder Abfrage fehlt, sind meistens Operationen, Attribute oder Datumsangaben. Alle haben entscheidenden Einfluss auf das Endergebnis. Ist nur eine dieser Elementgruppen fehlerhaft, wird auch das Ergebnis fehlerhaft sein. Fehlerhafte Tabellen und Attribute können sogar die ganze Abfrage ungültig machen und zu Fehlermeldungen führen.

Fehlermeldungen sowie Ergebnisse ohne enthaltene Daten sind bekannte Probleme im Bereich Text-to-SQL (Wang, 2025, S.11; Kanburoglu & Tek, 2024, S.411). Diese Probleme zeigen sich im vorliegenden Projekt ebenso. Bei den als schwierig eingestuften erwarteten SQL-Abfragen tauchen mit 26% der Fälle deutlich mehr Fehlermeldungen auf, als bei den als einfach eingestuften erwarteten Abfragen mit 6% der Fälle (evaluation.ipynb, Codezeile 31). Die meisten Fehlermeldungen bei den komplexen Abfragen betreffen die Wahl der Attribute ( z.B. „Unrecognized name“, „No matching“, „Column name“) mit 51% der Fehlermeldungen (evaluation.ipynb, Codezeile 34), was auf Schwierigkeiten bei der Schemaverknüpfung hindeutet – dies spiegelt sich auch wie oben beschrieben schon in der niedrigeren Zahl der korrekt gewählten Attribute wider.

Die Formulierung der Fragen mit Hintergrundwissen (simple) erzeugt weniger Ergebnisse ohne Daten (5%) als die Formulierung der Fragen ohne Hintergrundwissen (complex) mit 8% (evaluation.ipynb, Codezeile 30).

Im Gesamtbild erzeugt Analyst E (Kontext, COT, Retry) die wenigsten Probleme mit 12% Fehlermeldungen und Ergebnissen ohne Daten (entsprechend oben genanntem Wert der Ausführbarkeit von 88 %; evaluation.ipynb, Codezeile 32).

Fragen mit Hintergrundwissen (simple) erreichen für alle Zielmetriken etwas bessere Werte als Fragenformulierungen ohne Hintergrundwissen (complex). Die Differenz liegt zwischen 4 bis 8% und ist, wie in Abb.10 gezeigt, am relevantesten für die Ergebnisgenauigkeit (EA_rate) mit richtigen Ergebnissen in 27% der Fälle für Fragen mit und 19% für Fragen ohne Hintergrundwissen. Die Komplexität der SQL macht einen noch deutlicheren Unterschied für alle Zielmetriken (siehe Abb.11) – die Differenz liegt zwischen 22% bis 31%. Einfache SQL sind zu 89% ausführbar, die Abfragen stimmen zu 28% komplett mit der Grundwahrheit überein, die Ergebnisse zu 39%. Schwierige SQL sind dagegen nur zu 67% ausführbar und es stimmen nur sehr wenige (2%) der Abfragen genau mit dem Ideal überein. Die Ergebnisgenauigkeit liegt mit 8% wieder etwas höher.

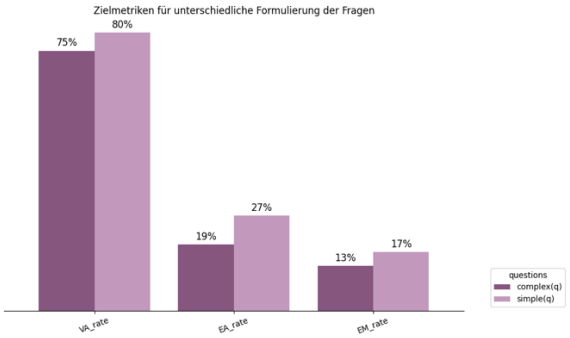



*Abbildung 10: Auswertung Fragenformulierung*

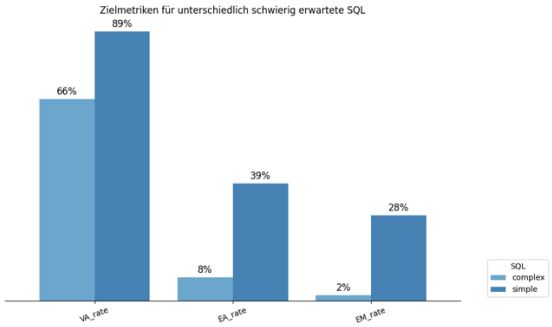



*Abbildung 11: Auswertung Komplexität der SQL*

## 4.Diskussion und Handlungsempfehlung

Im Folgenden werden die zuvor beschriebenen Ergebnisse im Hinblick auf die Hypothesen zu den drei Komponenten diskutiert und Handlungsempfehlungen gegeben.

Dass **Kontext** im hier vorgeschlagenen Umfang für die Ausführbarkeit wichtig ist, bestätigt die Annahmen von Hypothese 1 zum Teil. Nicht bestätigt werden konnte ein positiver Einfluss auf die Ergebnisgenauigkeit. Hier ging die Tendenz für einfache Abfragen sogar eher zu weniger Kontext (Abb.4). Dies kann wohl als Stärke aktueller Sprachmodelle in ihrer Generalisierungs- und Übersetzungsleistung interpretiert werden, wenn die Aufgabe einfach ist – demnach reichen wenige Details aus und ein Schema in voller Länge scheint nicht zu überfordern. Dies würde auch der Diskussion von Maamari et al. (2024, S.8) entsprechen, dass Schemaverknüpfung mit verbesserter Generalisierung unwichtiger wird. Gleichzeitig kann hier eine Schwäche des Workflows vermutet werden. Aufgrund des Projektumfanges wurde sich, wie zuvor beschrieben, für eine individuelle Art der Kontextzuführung entschieden, die insbesondere an den drei Stellen, an denen ein LLM dynamisch Informationen filtert, jeweils Fehlerpotential birgt, solange LLMs zu Halluzinationen neigen und sich nicht immer genau an Anweisungen halten. Zusätzlich können umfangreiche, detaillierte Prompts, wie sie insbesondere bei komplexen Fragen und mit Nutzung der vollen Kontext-Komponente vorliegen, bewirken, dass die Reasoning-Leistung von Sprachmodellen abnimmt und diese dazu tendieren, Anweisungen seltener zu befolgen - wobei ChatGPT 4.0 am robustesten gegen Input-Längen auftritt und mit COT-Prompting negative Tendenzen teilweise ausgeglichen werden können (Levy, Jacoby & Goldberg, 2024, S.2, S.5-7).

Im wissenschaftlichen Diskurs wurde bereits gezeigt, dass Kontext z.B. in Form einer klassischen RAG-Implementierung das Ergebnis von Text-to-SQL-Aufgaben deutlich verbessern kann (Mohammadjafari et al. 2025, S.2). RAG-Systeme suchen mittels numerischer Vektoren, die komplexe Daten wie Text als Zahlenfolgen zusammenfassen („Embeddings“),

in domänenspezifischen Vektordatenbanken und können so Schemainformationen zuverlässig filtern und abrufen (Marta, 2024, S. 40). Es wird daher empfohlen, für eindeutigere Ergebnisse die Komponente Kontext zukünftig entweder noch einmal im Hinblick auf kürzere Prompts und die grundsätzliche Anzahl der Schritte mit Einbindung eines LLM zu überarbeiten, oder über die Implementierung eines klassischen RAG-System mit einer Vektordatenbank nachzudenken. Zusätzlich oder alternativ könnte außerdem eine indirekte Kontextzuführung über ein Training des Sprachmodells auf die spezifischen Daten („Finetuning“) diskutiert werden. Dass *Finetuning* insbesondere für domänenspezifische Anforderungen Sinn machen kann, da es deutlich bessere Ergebnisse als Prompt Engineering Methoden erzielt, wird bereits von Ojuri et al. (2025, S.6) angedeutet.

Zur Analyse der Komponente **Prompting** werden die Strategien COT vs. OS miteinander verglichen. Die Annahme aus Hypothese 2, dass OS-Prompting die richtige Strategie für semantisch übereinstimmende SQL-Abfragen ist, wurde bestätigt. Das Ergebnis tendiert in eine Richtung, die auch in der Literatur zu finden ist, nämlich dass COT-Prompting erst in Kombination mit weiteren Strategien deutlich bessere Ergebnisse als Few-Shot-Prompting liefert (Pourreza & Rafiei, 2023, S.10; Hong (2025), S.9). Demnach könnte für die zukünftige Entwicklung diskutiert werden, der allgemeinen Empfehlung von Boonstra (2024, S. 31) zu folgen, COT mit OS oder Few Shot zu kombinieren, um beide Stärken für bessere Ergebnisse zusammenzuführen und zusätzlich ggf. sogar weitere Strategien wie Decomposition nach Pourreza & Rafiei zu ergänzen.

Die Komponente **Retry** ist relevant für die Ausführbarkeit und Ergebnisgenauigkeit. Dies bestätigt Hypothese 3 vollständig sowohl für einfache als auch für komplexe SQL. Sollte also ein Analyst gewünscht sein, der Abfrage-Ergebnisse liefert, ist die Retry-Schleife immer sinnvoll. Hier ist zu beachten, dass durch die wiederholte Abfrage der LLM-Schnittstellen mit höheren laufenden Kosten zu rechnen ist, als ohne Retry-Option. Die Implementierung einer Iteration ist dafür in LangGraph über optionale Kanten recht einfach zu gestalten.

In der **Gesamtsicht** bietet Analyst E, der den idealen Workflow mit angereichertem Schema, COT-Prompting und Retry nutzt, trotz der vorherigen Diskussion über Kontext und Prompting, die besten Ergebnisse bzgl. Ausführbarkeit und Ergebnisgenauigkeit, aber auch einer niedrigeren Anzahl von Fehlermeldungen. Dagegen liefern die Analysten mit OS-Prompting (Analysten A, C, D) alle drei bessere Werte bzgl. semantischer Übereinstimmung. Dennoch sollte zum aktuellen Stand der Entwicklung kein Analyst in Produktion gehen. Angesichts des Ergebnisses, dass sowohl die Rate der semantischen Übereinstimmung (EM_rate) als auch die Rate der Ergebnisgenauigkeit (EA_rate) für alle Analysten gering

ausfallen, aber die anteilige semantische Übereinstimmung (EM_share) deutlich höher ist, als die anteilige Ergebnisgenauigkeit (EA_share), könnte die Möglichkeit, eine menschliche Überprüfung der Abfrage vor ihrer Ausführung einzuführen, diskutiert werden. Dies könnte zusätzlich zu den Kontext-Empfehlungen die Schwächen in der Schemaverknüpfung bzgl. der Wahl von Attributen und Filterwerten ausgleichen und auch falsche Datumsangaben könnten hier korrigiert werden.

Da die erwartete Komplexität der SQL einen deutlichen Einfluss auf die Zielmetriken hat, ist es zum heutigen Stand erheblich zuverlässiger, die Analysten für einfachen SQL-Abfragen einzusetzen als für schwierige. Auch ist der Einsatz bei Stakeholdern aufgrund der etwas geringeren Werte der Zielmetriken bei der Fragenformulierung ohne Hintergrundwissen noch nicht bedenkenlos möglich.

Die hier vorgestellten Analysten bestehen aus einem strukturierten Workflow mit Iteration, wobei für einzelne Schritte Sprachmodelle eingesetzt werden. Im wissenschaftlichen Diskurs werden agentische und Multi-Agenten-Systeme als vielversprechend im Bereich Text-to-SQL für komplexe Aufgaben diskutiert (Zhu, 2024, S.14; Wang, 2025, S.6). Besonders bei den komplexen Aufgaben schneiden leider auch die vorgestellten Analysten schlecht ab. Es empfiehlt sich demnach, darüber nachzudenken, den vorliegenden Ansatz in Richtung Agent oder Multi-Agent-System weiterzuentwickeln. Dies wäre insbesondere auch mit dem hier genutzten Framework LangGraph möglich, da es auf agentische Systeme ausgelegt ist.

Im vorliegenden Projekt ist allein ChatGPT 4.0 als Sprachmodell getestet worden. Es könnten ergänzend Sprachmodelle von Anthropic oder auch die neueste GPT-Variante 5.0 im Workflow getestet werden. Insbesondere Modelle von Anthropic können lt. Narasimhan et al. (2024, S.4) als nahezu gleichwertige Alternative im Bereich Text-to-SQL gesehen werden.

Die Ergebnisse dieser Arbeit bieten praktischen Nutzen insbesondere für Unternehmen, die KI-gestützte Systeme zur Datenabfrage entwickeln oder einsetzen wollen. Grundsätzlich kann die vorliegende Arbeit Make-or-Buy-Entscheidungen erleichtern, da gezeigt wird, was im Unternehmen mit einem Open-Source-Framework, bereits vorhandenen LLM-Schnittstellen und der eingesetzten Datenbank möglich ist. Durch die gezielte Bewertung einzelner Analystenkomponenten wird zusätzlich deutlich, dass sich Investitionen in die Komponenten Kontext, Prompting und Retry aus verschiedenen Gründen lohnen.

Zwar sind die vorgeschlagenen Versionen von Analysten auf heutigem Stand noch nicht bereit für einen unternehmensweiten Einsatz, es wurden aber Empfehlungen diskutiert, damit dieses Ziel erreicht werden kann.

## 5.Quellenverzeichnis


Schluntz, E. & Zhang, B. (2024). *Building effective agents.* Anthropic. Abgerufen am 10.10.2025 von https://www.anthropic.com/engineering/building-effective-agents

Boonstra, L. (2024). *Prompt Engineering.* Whitepaper von Google. Abgerufen am 05.09.2025 von https://gptaiflow.com/assets/files/2025-01-18-pdf-1-TechAI-Goolge-white-paper_Prompt%20Engineering_v4-af36dcc7a49bb7269a58b1c9b89a8ae1.pdf

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F. & Huang, X. (2025). *Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL*. arXiv. https://arxiv.org/pdf/2406.08426

Kanburoglu, A.B., Tek, F.B. (2024). *Text-to-SQL: A methodical review of challenges and models.* Turkish Journal of Electrical Engineering and Computer Sciences: Vol. 32: No. 3, Article 4. https://doi.org/10.55730/1300-0632.4077

Kansal, A. (2024). *Building Generative AI-Powered Apps - A Hands-on Guide for Developers*. Apress. https://doi.org/10.1007/979-8-8688-0205-8

LangGraph. *Graph API concepts.* Abgerufen am 16.10.2025 von https://langchain-ai.github.io/langgraph/concepts/low_level/ (könnte bei Korrektur der Arbeit bereits nach Release von LangGraph v1.0 entfernt worden sein, dann hier: https://docs.langchain.com/oss/python/langgraph/graph-api)

Levy, M., Jacoby, A., Goldberg, Y. (2024). *Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models.* arXiv. https://arxiv.org/pdf/2402.14848

Maamari, K., Abubaker, F., Jaroslawicz, D. & Mhedhbi, A. (2024). *The Death of Schema Linking - Text-to-SQL in the Age of Well-Reasoned Language Models*. In Table Representation Learning Workshop at NeurIPS 2024.

Martra, P. (2024). *Large Language Models Projects: Apply and Implement Strategies for Large Language Models*. Apress. https://doi.org/10.1007/979-8-8688-0515-8

Mohammadjafari, A., Maida, A. & Gottumukkala, R. (2025). *From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems*. arXiv. https://arxiv.org/pdf/2410.01066

Narasimhan, A., Bhamboo, A.K. & Devnathan, A. (2024). *Benchmarking Large Language Models for NL-to-SQL: A Comprehensive Evaluation of Accuracy, Cost and Throughput.* TechRxiv. DOI: 10.36227/techrxiv.173121325.56335825/v1

Nascimento, E.R., García, G., Izquierdo, Y.T., Feijó, L., Coelho, G.M.C., de Oliveira, A.R., Lemos, M., Garcia, R.L.S., Paes Leme, L.A.P., Casanova, M.A. (2024). *LLM-Based Text-to-SQL for Real-World Databases.* SN Computer Science (2025) 6:130. https://doi.org/10.1007/s42979-025-03662-6

Oliveira, A. *et al.* (2025). *Small, Medium, and Large Language Models for Text-to-SQL*. In: Maass, W., Han, H., Yasar, H., Multari, N. (eds) Conceptual Modeling. ER 2024. Lecture Notes in Computer Science, vol 15238. Springer, Cham. https://doi.org/10.1007/978-3-031-75872-0_15

Ojuri, S., Han, T. A., Chiong, R., & Di Stefano, A. (2025). *Optimizing text-to-SQL conversion techniques through the integration of intelligent agents and large language models. Information Processing and Management*, *62*, 104136. https://doi.org/10.1016/j.ipm.2025.104136

Pourreza, M., Rafiei, D. (2023). *DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction.* arXiv. https://arxiv.org/pdf/2304.11015

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang P., Mao, Y., Polozov, O., Singh, R. (2018). *Robust Text-to-SQL Generation with Execution-Guided Decoding.* arXiv. https://arxiv.org/pdf/1807.03100

Wang, L., Ma, C., Feng, X. et al. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, *18*, 186345. https://doi.org/10.1007/s11704-024-40231-1

Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q., Yin, D., Sun, X. & Li, Z. (2025). *MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL*. arXiv. https://arxiv.org/pdf/2312.11242

Xi, Z., Chen, W., Guo, X. et al. (2025). *The rise and potential of large language model based agents: a survey*. Sci. China Inf. Sci. 68, 121101. https://doi.org/10.1007/s11432-024-4222-0

Yan, L., Su, J., Liu, C., Duan, S., Zhang, Y., Li, J., Han, P. & Liu, Y. (2025). *ExSPIN: Explicit Feedback-Based Self-Play Fine-Tuning for Text-to-SQL Parsing.* Entropy 2025, 27, 235. https://doi.org/10.3390/e27030235 Co

Zhang, K., Lin, X., Wang, Y., Zhang, X., Sun, F., Cen, J., Jiang, X., Tan, H., & Shen, H. (2023). ReFSQL: A Retrieval-Augmentation Framework for Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023* (pp. 664–673).

Zhu, X., Li, Q., Cui, L. & Liu, Y. (2024). *Large Language Model Enhanced Text to SQL Generation: A Survey*. arXiv. https://arxiv.org/pdf/2410.06011

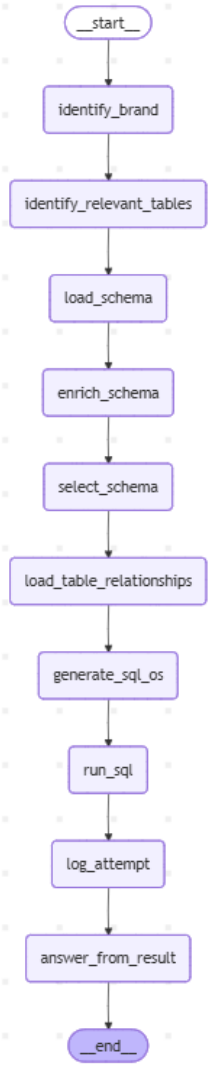
Zuckarelli, J. L. (2025). *Programmieren mit ChatGPT Eine kompakte Einführung*. Springer Vieweg. https://doi.org/10.1007/978-3-662-69433-6

# 6.Anhang

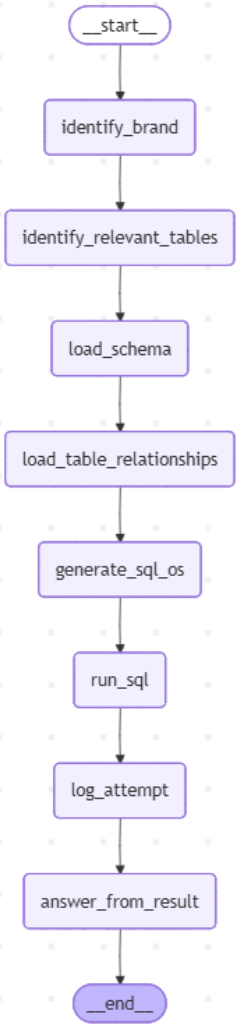
## 6.1. unterschiedliche Workflows entsprechend Ablation Studies

A, B, C mit OS Prompting, vollem Kontext (A) vs. Basic Kontext(C, D), mit (D) und ohne Retry (A, C)

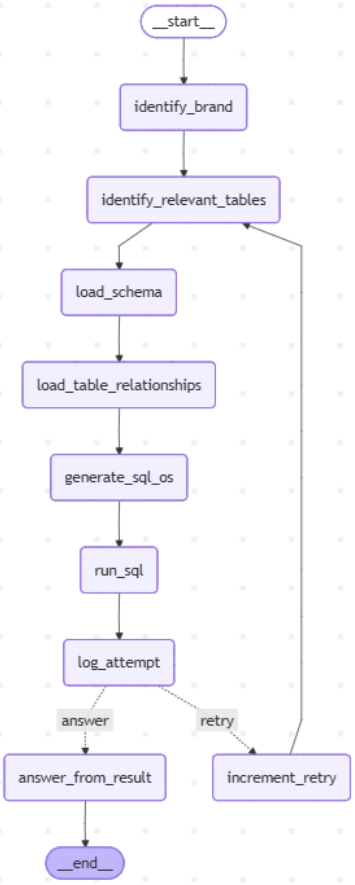
**(A)**




**(C)**

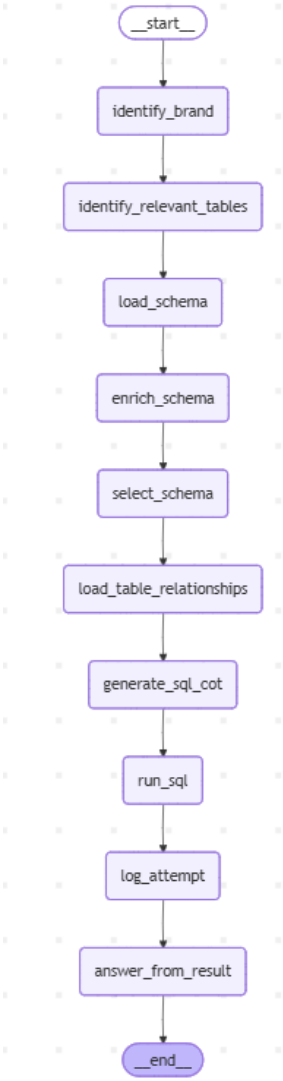



**(D)**



B, C mit COT Prompting und vollem Kontext, sowie mit( E) und ohne (B) Retry

**(B)**




**(E)**