Bachelorarbeit

Text-to-SQL für die Abfrage und Analyse räumlicher Daten: Eine Fallstudie mit LLMs, PostgreSQL und PostGIS

Vorgelegt von

Weber Marc

E073056872

Dozent

Gellrich Mario

Institution

ZHAW- Zürcher Hochschule für angewandte Wissenschaften

School of Management and Law

Studiengang Wirtschaftsinformatik

Abgabedatum

28.5.2025

Inhaltsverzeichnis

I. Abbildungsverzeichnis

II. Tabellenverzeichnis

III. Begriffe und Abkürzungsverzeichnis

Begriff	Bedeutung
Goldtabelle	Eine Goldtabelle ist eine vordefinierte, korrekte Tabelle, die in einer Datenbank als Referenzpunkt dient.
LLM	Ein KI-Modell, das auf grossen Textmengen trainiert wurde und natürliche Sprache in SQL-Abfragen umwandeln kann.
SQL	Structured Query Language ist die Standardabfragesprache für relationale Datenbanken.
GIS	Ein Geoinformationssystem ist eine Software oder Datenbank, die dazu dient, geospatiale Daten zu speichern, zu analysieren und zu visualisieren.
CSV	Dateiformat Comma-Separated Values
AWS	Amazon Web Services
ACID	Atomicity, consistency, isolation und durability
EC2	Amazon Web Services: Elastic Compute Cloud (Server)
WAL	Write Ahead Logs
Tokens	Grösse Input / Output wird anhand Tokens gemessen bei den LLM

1 Einleitung

Die Nutzung und Verbreitung von Large Language Models wie GPT-4 hat die automatische Generierung von SQL-Abfragen durch natürliche Spracheingaben erheblich verbessert. Während klassische Text-to-SQL-Modelle einfache SQL-Abfragen in strukturierten Datenbanken aufgrund regelbasierter Systeme beschränkt waren, ermöglichen LLMs wie GPT-4 eine flexiblere und präzisere Abfrageerstellung mit besseren Ergebnissen. In vielen Anwendungsfällen haben LLMs bereits leistungsstarke Resultate gezeigt für klassische relationale Datenbanken. Unklar bleibt, wie leistungsstark die LLMs sind bei der Generierung von SQL-Abfragen mit Datenbank-Erweiterungen. Unklar bleibt wie die Leistungsfähigkeit der Modelle sind bei der Generierung von geospatialen SQL-Abfragen wie zum Beispiel anhand von der PostGIS Datenbank Erweiterung.

Diese Bachelorarbeit untersucht die Leistungsfähigkeit von LLMs, komplexe PostGIS Abfragen zu generieren. Anhand eines eigenen Benchmarks werden die Qualität und Genauigkeit der Text-to-SQL-Abfragen in räumlichen Datenbanken systematisch geprüft. Zudem werden die LLMs anhand eines Feedback-Loops weiter bereichert, um die Modelle iterativ zu verbessern. Als Resultat werden mehrere Benchmark-Durchläufe mit verschiedenen Feedback-Loops verglichen, um den Effekt von Feedback-Loops festzuhalten und die Leistung der LLMs zu determinieren.

1.1 Ausgangslage

Text-to-SQL Systeme wandeln natürliche Texteingaben in SQL-Abfragen um. Dies erlaubt nicht-technischen Benutzern komplexe Datenbankabfragen auszuführen ohne vorherige SQL-Kenntnisse zu verfügen (Li et al., 2023, S. 1). Datenbanken erstellen die Basis für die meisten technologischen Anwendungen. Der Bedarf an SQL-basierten Datenbanken ist hoch, über 60% aller Datenbanken sind SQL-basiert (Scalegrid, 2019). Ferner werden Geodatenbanken für Datenanalysen, Business Intelligence und automatisierte Abfragen verwendet (Zhang et al., 2024, S. 1). Durch die weitverbreitete Benutzung von LLMs wird der Bereich Text-to-SQL anhand von Benchmarks wie Spider oder BIRD ausgewertet. LLMs erzielen auf den Benchmarks bereits eine sehr hohe Leistung (Jiang & Yang, 2024, S. 2). Während Text-to-SQL Benchmarks bereits existieren, fehlt ein Benchmark und eine systematische Analyse im Bereich geospatialer SQL, insbesondere für PostGIS.

1.2 Problemstellung

Obwohl LLMs in klassischen Text-to-SQL-Benchmarks beeindruckende Leistungen erzielt, ist es unklar, wie zuverlässig komplexe PostGIS-basierte Fragen LLMs generieren können. PostGIS Funktionen wie ST_Within, ST_Distance oder ST_Intersects setzen ein tiefes Verständnis von SQL und geographischen Konzepte, die in SQL-Abfragen umgeschrieben werden. Zudem ist nicht klar, wie genau die Auswertung von PostGIS-Abfragen aussehen kann, weil es keinen Benchmark gibt, der für PostGIS-Abfragen optimiert ist. Weil Benchmarks wie BIRD oder Spider es an PostGIS-Aufgaben mangelt, kann keine systematische Analyse im Bereich geospatialer SQL-Abfragen gemacht werden. Sprich die Resultate lassen sich nur schwer mit anderen Benchmarks vergleichen, weil die Basis der Fragen sich von Benchmark zu Benchmark unterscheidet.

1.3 Forschungslücke

Es existieren wenige Untersuchungen zur Leistungsfähigkeit von Text-to-SQL-Abfragen, welche spezifisch auf der PostGIS-Datenbank-Erweiterung fokussiert sind. Die Benchmarks, die bisher für Text-to-SQL-Abfragen verwendet werden für Text-to-SQL-Abfragen beinhalten nur vereinzelt geospatiale SQL-Abfragen. In den Resultaten der bisherigen Arbeiten wird die Leistungsfähigkeit von Datenbank Erweiterungen nur selten betrachtet, sondern entweder der Fokus auf die Leistung der LLMs gelegt oder wie ein Modell anhand von Inputs und Feedback-Loops verbessert werden kann. Der Einsatz von Feedback-Loops zur iterativen Verbesserung der geospatialen Abfragen ist bislang nicht in einem PostGIS-Kontext untersucht worden.

1.4 Zielsetzung

Ziel dieser Bachelorarbeit ist es, die Leistungsfähigkeit von LLMs bei der Generierung von PostGIS-SQL-Abfragen zu evaluieren. Die Qualität der LLMs wird anhand des BIRD-Benchmarks gemessen, einem umfassenden Bewertungssystem für Text-to-SQL-Modelle. Der BIRD-Benchmark ermöglicht es, die von LLMs generierten SQL-Abfragen direkt mit den vorgegebenen Musterlösungen zu vergleichen (Li et al., 2023, S. 1). Besonders relevant für diese Arbeit ist, dass der BIRD-Benchmark nicht nur allgemeine SQL-Abfragen, sondern auch spezifische PostGIS-Abfragen umfasst. Der Fokus dieser Untersuchung liegt daher gezielt auf der Eignung der LLMs für räumliche Datenbankabfragen, um zu prüfen, inwieweit Modelle die komplexe geospatiale SQL-Befehle korrekt interpretieren und umsetzen können.

1.5 Forschungsfrage

Die zentrale Forschungsfrage dieser Bachelorarbeit lautet: „Text-to-SQL für die Abfrage und Analyse räumlicher Daten: Eine Fallstudie mit LLMs, PostgreSQL und PostGIS“. Um diese Forschungsfrage zu beantworten, werden folgende Teilfragen untersucht:

- Wie genau sind die von LLMs generierten SQL-Abfragen im Vergleich zu manuell erstellten Abfragen?
- Welche technischen und semantischen Herausforderungen ergeben sich bei der Umsetzung von Text-to-SQL im Kontext geospatialer Daten?
- Wie wirkt sich ein Feedback-Mechanismus auf die Verbesserung der SQL-Abfragen aus?

1.6 Abgrenzung

Die Arbeit limitiert sich ausschliesslich auf die Anwendung von bestehende LLMs, die für SQL-Abfragen verwendet werden können. Es wird kein eigenes Sprachmodell trainiert, sondern wie gut bestehenden LLMs mit der räumlichen Datenbank-Erweiterung PostGIS umgehen können. Für die Evaluation wird ein eigener Benchmark erstellt, worin die Leistung der LLMs, sowie den Einfluss von Feedback-Loops auf PostGIS Text-to-SQL Aufgaben ermittelt wird. Der Mehrwert liegt in der Fokussierung auf eine unterforschte Subdomäne, und zwar die Leistung von LLMs, inwiefern PostGIS-Text-to-SQL-Aufgaben von LLMs korrekt gelöst werden können.

1.7 Aufbau der Arbeit

Die Arbeit ist in 7 Kapiteln aufgeteilt und baut aufeinander auf. Im ersten Kapitel wird eine Einleitung und die Relevanz der Arbeit beschrieben. Das zweite Kapitel beschreibt die methodische Vorgehensweise zur Evaluierung und Verbesserung der Modelle. Das dritte Kapitel beschreibt den Stand der Forschung im Bereich Datenbanken, Text-to-SQL, LLMs, PostGIS und Benchmarks vorgestellt. Im vierten Kapitel wird die Methode und Vorgehensweise beschrieben. Dabei werden essenzielle Fragen beantwortet, etwa wie die Feedback-Loops implementiert wurden, wie der eigene Benchmark erstellt wurde, wie die Benchmark Resultate bewertet wurden, wie Prompts zusammengebaut werden und wie die Datenbank in einer Cloud-Umgebung bereitgestellt wurde. Im fünften Kapitel werden die Resultate dargestellt. Zum einen wird die Webapplikation erklärt und zum anderen werden die Resultate durch den Benchmark erläutert. Im sechsten Kapitel werden die Resultate meiner Arbeit diskutiert und verglichen mit den Resultaten von anderen Arbeiten. Im siebten Kapitel wird die Schlussfolgerung mit Handlungsempfehlungen präsentiert. Danach befindet sich das Literaturverzeichnis und im letzten Kapitel befindet sich der Anhang, wo sich alle Benchmark-Text-to-SQL Aufgaben befinden mit Musterlösungen.

1.8 Verwendung von Künstlicher Intelligenz

Künstliche Intelligenz wurde für diese Arbeit zur Hilfe gezogen in verschiedenen Prozessen. Es wurde nicht nur Künstliche Intelligenz verwendet, um SQL-Abfragen zu erstellen innerhalb der Applikation, sondern assistiert auch während des Schreibprozesses.

Tabelle 1 Verwendung von Künstlicher Intelligenz

KI-System	Anwendung und Bemerkung
ChatGPT	ChatGPT wurde für die Formatierung, Korrektur der Arbeit und zur Zusammenfassung von wissenschaftlichen Papieren in der Literaturrecherche verwendet. ChatGPT hat mit dem Erstellen des Benchmarks assistiert.
Consensus	Consensus wurde für die Literaturrecherche benutzt. Die Nutzung erfolgte auf der Consensus Webseite.
Microsoft Copilot	Microsoft Copilot wurde als Code Assistenten innerhalb Visual Studio Code verwendet. Copilot hat die meisten Kommentare in der Applikation versehen.

In der Tabelle 1 kann entnommen werden, dass drei verschiedene KI-Systeme bei dem Assistieren der Artefakte mitgeholfen haben. ChatGPT ist hauptsächlich als ein Schreib-Coach bei dieser Arbeit dazugekommen. Der ganze Text der Bachelorarbeit wurde zuerst ohne ChatGPT geschrieben und wurde am Ende speziell für eine akkurate Wortwahl sowie die Verbesserung von Grammatik- und Rechtschreibfehlern zur Hilfe genommen. Consensus ist ein KI-basiertes Recherche Tool, das relevante wissenschaftliche Arbeiten zu einer spezifischen Fragen findet. Consensus wurde oft für Literatur Recherche verwendet. Microsoft Copilot ist innerhalb meiner Visual Studio Code Umgebung installiert und dient als einen Code-Assistenten.

2 Forschungsdesign

Diese Arbeit orientiert sich am Design Science Research (DSR)-Ansatz, der sich auf die Entwicklung und Evaluation eines innovativen Artefakts konzentriert. Im Gegensatz zu rein explorativen oder empirischen Ansätzen verfolgt DSR das Ziel, eine konkrete Lösung für ein bestehendes Problem zu entwerfen und deren Wirksamkeit systematisch zu überprüfen (Hevner et al., 2004). In dieser Arbeit wird untersucht, inwiefern LLMs wie GPT-4 in der Lage sind, korrekte und effiziente SQL-Abfragen für PostGIS-Datenbanken zu generieren.

Tabelle 2 Angewandte DSR-Phasen

Phase	Anwendung in dieser Arbeit
Problemidentifikation	Untersuchung der Herausforderungen von LLMs bei der Generierung von PostGIS-SQL-Abfragen.
Anforderungsanalyse	Definition der Anforderungen an das System: LLM-gestützte SQL-Generierung, Feedback-Loops und Benchmarking.
Artefakt-Design	Entwicklung einer Webapplikation, die LLMs nutzt, um SQL-Abfragen aus natürlicher Sprache für PostGIS zu generieren.
Implementierung	Umsetzung des Systems mit React (Frontend), Java Spring Boot und PostgreSQL/PostGIS (Datenbank).
Evaluation	Vergleich der LLM-generierten Abfragen anhand eines eigenen Benchmarks. Die Resultate mit anderen Text-to-SQL Benchmarks wie BIRD oder Spider vergleichen.
Iterative Verbesserung	Optimierung durch Feedback-Loops, korrekt generierte Abfragen werden in einer Tabelle in der Datenbank gespeichert und dient als Anhaltspunkt für LLMs für zukünftige SQL-Abfragen.

Die Tabelle 2 listet alle Phasen auf, die aus Sicht von Design Science Research in dieser Arbeit relevant sind. Während der Problemidentifikation werden die Herausforderungen von LLMs bei der Generierung von PostGIS-SQL-Abfragen untersucht. Anforderungsanalyse definiert, was ein System benötigt, um LLM-gestützte SQL-Generierung, Feedback-Loops und einen eigenen Benchmark zu erstellen. Unter Artefakt-Design wird in dieser Arbeit eine Webapplikation entwickelt und ein Benchmark erstellt. Die Webapplikation wird durch ein Frontend und ein Backend implementiert. Das Backend basiert auf Spring basierend und das Frontend basiert auf React. Die Evaluation besteht darin, die Ergebnisse des Benchmarks

auszuwerten und anhand von Feedback-Loops eine Applikation zu entwickeln, die sich iterativ verbessert.

3 Stand der Forschung

Die Forschung im Bereich Text-to-SQL hat in den letzten Jahren, mit der rapiden Entwicklung von neuen LLMs, erhebliche Fortschritte gemacht. Während früher regelbasierte Methoden basierten, ermöglichen LLMs die automatische Generierung von SQL-Abfragen aus natürlicher Sprache. Für diese Arbeit wird spezifisch auf Text-to-SQL für eine PostgreSQL-Datenbank mit der PostGIS-Erweiterung eingegangen. Es werden LLMs verwendet, um SQL-Abfragen zu generieren. Zuerst wird ein Überblick über Text-to-SQL gegeben, wie geospatiale Daten verarbeitet werden und wie es sich über die Jahre entwickelt hat. Unter anderem welche Datenbank Optionen in Frage kommen für geospatiale Daten. Anschliessend werden die Herausforderungen bei der Generierung von PostGIS geospatiale SQL-Abfragen erläutert und benennt und die bestehende Text-to-SQL Benchmarks diskutiert. Folgend werden die Resultate der Benchmark dienen einen Überblick zu verschaffen, wie weit die Forschung bereits ist, Text-to-SQL Lösungen zu erstellen.

3.1 Begriffe

Datenbanken legen die Basis für moderne Anwendungen, mit denen wir täglich interagieren, indem sie Informationen organisiert und zugänglich macht. Eine Datenbank ist eine strukturierte Sammlung von Daten. Datenbanken speichern, verwalten und rufen Daten ab. Unter anderem werden Datenbanken in Unternehmen, Wissenschaft, Gesundheitswesen und E-Commerce eingesetzt (Martins, 2019, S. 154). Moderne Datenbanken werden anhand von Eigenschaften in verschiedenen Datenbank-Kategorien unterteilt.

Tabelle 3 Datenbank Variationen

Typ	Beschreibung
Relationale Datenbanken	Basiert auf tabellenstrukturierten Daten und Abfragen werden anhand der SQL-Sprache durchgeführt. Beispiele sind PostgreSQL oder MySQL.
NoSQL-Datenbanken	Sind für unstrukturierte und semi-strukturierte Daten optimiert. Es werden Daten abgespeichert anhand Key-Value Pairs. NoSQL Datenbanken werden häufig in Big-Data-Anwendungen genutzt (Silva et al., 2016, S. 413). Ein Beispiel ist DynamoDB von AWS.
NewSQL-Datenbanken	Kombinieren die Skalierbarkeit von NoSQL mit der ACID Transaktionssicherheit relationaler Datenbanken (Silva et al., 2016, S. 414).
Dokument-Datenbanken	Ist ein Typ von NoSQL-Datenbanken, die Informationen in Dokumenten abspeichert. Beispiel für eine Dokument-Datenbank ist MongoDB.

In Tabelle 3 werden vier verschiedene Datenbankentypen aufgelistet und beschrieben. Relationale Datenbanken sind weit verbreitet und finden eine Vielzahl an Anwendung. Relationale Datenbanken verwenden SQL als eine standardisierte Sprache zur Abfrage von Resultaten. SQL ist eine deklarative Sprache, sprich der Benutzer gibt an, was er abrufen möchte. Dabei ist es nicht wichtig, wie es abgerufen wird (Silva et al., 2016, S. 413). Diese Eigenschaft macht es einfacher für nicht technische Benutzer SQL zu lernen und zu verstehen. SQL wird mittlerweile in anderen Datenbanken verwendet, wie zum Beispiel das Big-Data-System Hadoop (Silva et al., 2016, S. 414). In dieser Arbeit werden wir mit PostgreSQL, einer Relationalen Datenbank arbeiten, die SQL basiert ist. Das Besondere an PostgreSQL ist, dass verschiedene Erweiterungen für die Datenbank installiert werden kann. Diese Arbeit wird sich mehrheitlich mit der Erweiterung PostGIS befassen. PostGIS ist eine räumliche Datenbank-Erweiterung (Hsu & Obe, 2021, S. 45). Es dient für eine effiziente Speicherung und Verwaltung von Daten innerhalb einer PostgreSQL Datenbank (Hsu & Obe, 2021, S. 45). Neben SQL-Datenbanken gibt es zwei andere Datenbank Typen, die nicht in dieser Arbeit verwendet werden, aber auch ihre spezifischen Anwendungsbereiche finden. Nämlich NoSQL Datenbanken und Dokument Datenbanken. NoSQL Datenbanken eignen sich besonders für Anwendungen mit vielen unstrukturierten und schnell ändernden Daten (Corbellini et al., 2017,

S. 2). Dokument Datenbanken sind ideal für flexibel hierarchisch strukturierte Informationen, zum Beispiel JSON-Dokumente.

Ein LLM ist ein neuronales Netzwerk, das trainiert ist, natürliche Sprache zu verstehen und Antworten als Bilder, Videos, Text und weiteres zu generieren (Head et al., 2023, S. 34). LLMs brauchen riesige Mengen an Textdaten aus Büchern, wissenschaftlichen Artikeln und Webseiten, um ein Sprachverständnis zu entwickeln. Anhand des Pretrainings lernt ein Modell, welche Wörter es im richtigen Kontext vorhersagen kann, indem es ein Muster in den Daten erkennt. Nach einem Pretraining wird das Modell auf spezifische Angaben verbessert, durch überwachtes Lernen oder Feedback-Loops (Head et al., 2023, S. 35). Ein grosser Vorteil sind Transformer-Architekturen, Selbstaufmerksamkeit verleihen. Durch die Selbstaufmerksamkeit kann ein LLM den Kontext von Worten in einem Satz verstehen, anstatt nur Wort-für-Wort-Analysen (Yang et al., 2024, S. 3). LLMs finden Anwendung in der Textgenerierung, Datenanalyse, Code-Generierung, Übersetzung. Diese Arbeit geht in die Richtung von Code-Generierung und Übersetzung bei der Generierung von SQL-Abfragen. Durch eine natürliche Spracheingabe ermöglicht LLMs SQL-Abfragen zu generieren, diese spezifische Art von Applikation wird Text-to-SQL genannt. In dieser Arbeit wird spezifisch auf Text-to-SQL Abfragen fokussiert, welche anhand Feedback-Loops genaue PostGIS SQL-Abfragen generiert.

Ein Goldtabelle ist eine vordefinierte korrekte Tabelle, die für die Antwort verwendet werden kann (Han et al., 2024, S. 1). Goldtabellen werden verwendet, um dem LLM mitzuteilen, wie die Datenbank aussieht, wie die Kardinalitäten zwischen den Tabellen aussehen und was die Tabellen für Spalten haben. Der Nutzen von Goldtabellen ist gross und wird in den meisten Text-to-SQL Methoden verwendet. Die Nützlichkeit von Goldtabellen ist weniger hoch, wenn ein LLM bereits das Datenbankschema kennt, weil es eine bekannte Datenbank ist.

3.2 Räumliche Datenbanken

Eine Räumliche Datenbank ist eine Datenbank, die anhand von Erweiterungen optimiert ist für geographische Daten. Räumliche Datenbanken haben vielfältige Anwendungen in der Geodatenanalyse, Smart Cities und Navigationssystemen gefunden. Durch Räumliche Datenbanken können komplexe räumliche Berechnungen, die weit über reine Kartenvisualisierung hinausgehen (Hsu & Obe, 2021, S. 48). PostGIS ist eine Räumliche Datenbankerweiterung, die es ermöglicht, räumliche Daten effizient zu speichern, verwalten und analysieren (Hsu & Obe, 2021, S. 45). PostGIS kann als eine Datenbank Erweiterung auf PostgreSQL-Datenbanken installiert werden. Nach Installation sind die PostGIS Funktionen für Datenbank-Abfragen frei verfügbar. Geodatenanalysen sind rechenintensiv, insbesondere wenn komplexe räumliche Operationen wie Flächenberechnung oder Entfernungsanalysen durchgeführt werden. Ohne spezielle Optimierung für räumliche Daten stossen reine SQL-Datenbanken schnell an Leistungsgrenzen (Hsu & Obe, 2021). Anhand der GIST-Indexierung und parallele Verarbeitung Optimierungstechniken in PostgreSQL und der PostGIS Erweiterung, können SQL-Abfragen schneller ausgeführt werden. Die Performance wird sich speziell bei grossen Geodatenmengen erheblich verbessern (Ilba, 2021, S. 4). Die Kombination PostgreSQL mit der PostGIS Erweiterung ist die de-facto SQL-basierte Geodatenbank (Hsu & Obe, 2021, S. 45). PostGIS Abfragen sehen wie SQL-Abfragen mit weiterer Funktionalität aus, folgendes simples Beispiel sollte ein besseres Bild von PostGIS verschaffen.

```
SELECT p.name AS post_office, s.name AS shopping_center
FROM planet_osm_point p
JOIN planet_osm_polygon s ON ST_Within(p.way, s.way)
WHERE p.amenity = 'post_office'
 AND s.shop = 'mall'
ORDER BY p.name;
```

Im obigen Beispiel wird die OpenStreetMap Datenbank verwendet, um alle Einkaufszentren zu identifizieren, welche eine Postfiliale beinhalten. In diesem Beispiel ist ST_Within(X, Y) die einzige PostGIS Funktion, die verwendet wurde. ST_Within überprüft, ob Geometrieobjekt vollständig innerhalb eines anderen Geometrieobjekts liegt. Sprich, ist X innerhalb von Y?

Wenn ja, wird der Boolean-Wert true zurückgegeben, wenn nicht wird der Boolean-Wert false zurückgegeben (PostGIS, o. J., S. 1).

3.3 Text-to-SQL

Text-to-SQL ist die Art bei dem durch normale Spracheingaben eine SQL-Abfrage erstellt wird. Die Idee, SQL-Abfragen dynamisch oder automatisch zu generieren, gibt es schon vor KI-Zeiten. Die Systeme waren rudimentär und regelbasiert. Mit LLMs besteht jetzt die Möglichkeit SQL-Abfragen zu generieren. Probleme wird es geben, wenn mit der Datenbankerweiterung PostGIS gearbeitet wird aufgrund der zunehmenden Komplexität von PostGIS Abfragen.

3.3.1 Text-to-SQL bevor KI

SQL ist eine wichtige Sprache für SQL-basierte Datenbanken. Für nicht technisch versierte Benutzer stellt die SQL-Sprache eine grosse Hürde dar (Fürst et al., 2025, S. 2). Text-to-SQL reduziert diese Hürde durch die Verwendung natürlicher Spracheingaben, die anhand eines LLMs automatisch in SQL-Abfragen umgewandelt werden. (Jiang & Yang, 2024, S. 5). Datengetriebene Entscheidungsprozesse gewinnen zunehmend an Bedeutung aber viele Arbeitskräfte haben keine SQL-Kenntnisse, um komplexe SQL-Abfragen durchzuführen (Han et al., 2024, S. 3). Alte Text-to-SQL waren regelbasiert und hatten nur die Fähigkeiten einfache SQL-Statements auf einzelnen Tabellen zu generieren (Li et al., 2023, S. 7). Frühere Text-to-SQL Ansätze wie WikiSQL und ATIS basieren auf regelbasierte Methoden, das heisst, nur einfache SQL Abfragen können durch diese Ansätze generiert werden (Han et al., 2024, S. 4). Diese SQL-Ansätze haben grosse Probleme mit komplexen SQL-Strukturen insbesondere wenn mehrere Tabellen, Joins und Aggregationen vorliegen (Li et al., 2023, S. 5).

3.3.2 Text-to-SQL nach KI

Moderne LLMs wie GPT-4 ermöglichen genauere SQL-Abfragen zu generieren, indem LLMs semantische Zusammenhänge in Anfragen besser verstehen (Zhang et al., 2024, S. 7). Auch mit weiteren Fortschritten gibt es weiterhin Herausforderungen in der Korrektheit und Effizienz von komplexen SQL-Abfragen, speziell bei PostGIS-spezifischen Funktionen (Jiang & Yang, 2024, S. 7). Mit Transformer-Modellen wie GPT4 wurden die SQL-Abfragen, die generiert werden, erheblich verbessert. Grund dafür ist, dass sie auf grossen Mengen echter SQL-Abfragen trainiert wurden (Jiang & Yang, 2024, S. 7). Der BIRD-Benchmark wurde entwickelt, um verschiedene Text-to-SQL Modelle möglichst realitätsnahe zu evaluieren (Jiang & Yang, 2024, S. 9). Durch Resultate vom BIRD Benchmark ist es deutlich, dass LLM-basierte

Text-to-SQL Modelle besser und genauer sind als alte regelbasierte Systeme (Zhang et al., 2024, S. 6). Auch mit diesen Fortschritten bestehen weiterhin gröbere Herausforderungen bei der Generierung von PostGIS-Abfragen, weil viele der LLMs nicht spezifisch auf räumliche SQL-Abfragen trainiert wurden (Zhang et al., 2024, S. 7). Ein weiterer Punkt, mit dem LLMs oft Probleme haben sind Halluzinationen, sprich ein LLM erfindet eine Antwort ohne Rückschlüsse woher diese Informationen kommen (Farquhar et al., 2024, S. 1). Damit LLMs gut abschneiden benötigen sie im Input Angaben, wie die Datenbank aussieht, welche Tabellen die Datenbank beinhaltet und die Kardinalitäten zwischen den Tabellen. Diese Informationen können anhand von Goldtabellen mitgegeben werden, um valide und korrekte SQL-Abfragen zu erstellen (Han et al., 2024, S. 3). Komplexe SQL-Abfragen erhöhen die Wahrscheinlichkeit auf Syntaxfehlern und semantischen Ungenauigkeiten für LLMs (Jiang & Yang, 2024, S. 5).

3.3.3 Herausforderungen bei Text-to-SQL mit PostGIS

LLMs wie zum Beispiel GPT-4 wurden primär für allgemeine SQL-Abfragen trainiert. Unter diesen Trainingsdaten befinden sich kaum spezialisierte geospatiale PostGIS spezifische SQL-Abfragen, was zu Fehlern und Ungenauigkeiten bei der Generierung von PostGIS SQL-Abfragen führen kann (Zhang et al., 2024, S. 3). Während Trainingsdaten ein Problem sind, haben LLMs den Vorteil, dass sie grosse Zusammenhänge verstehen, was besonders in räumlichen Datenbanken hilfreich sein könnte (Nascimento et al., 2025, S. 1). Produktionsdatenbanken sind komplex aufgebaut mit vielen verschiedenen Tabellen und Kardinalitäten. Dies erschwert den Datenbank Kontext mit Golden Tabellen, Prompts und eine Liste von Beispielen einem LLM mitzugeben, ohne dass ein LLM den Überblick verliert (Nascimento et al., 2025, S. 3). PostGIS beinhaltet räumliche Integritätsregeln, mit denen normale SQL-Abfragen nicht immer kompatibel sind. Das kann zum Problem werden, weil viele LLMs keine speziellen Mechanismen zur Validierung der geometrischen Konsistenz beinhalten (Lizardo & Davis, 2017, S. 2). Ferner haben LLMs Probleme mit der Generierung von Joins und Subqueries. Für simple SQL-Abfragen ist dies ausreichend, aber komplexe PostGIS Abfragen können ineffiziente Ergebnisse liefern (Li et al., 2023, S. 5). Eine Studie von GS-SQL hat die Performance von PostGIS SQL-Abfragen speziell mit ST_Contains, ST_Distance und ST_Intersects durchgeführt. Die Studie deutet darauf hin, dass in 50 – 65% der Fälle eine korrekte SQL-Abfrage generiert wurde verwendet wurde Verwendet wurde dabei der GeoSpatialSpider-Datensatz (Zhang et al., 2024, S. 6).

3.3.4 Benchmark

Um die Leistung eines Text-to-SQL Modells zu messen, werden Benchmarks verwendet. Benchmarks beinhalten natürliche Spracheingaben und die äquivalenten SQL-Abfragen, die als der Gold-Standard der Evaluierung dienen (Pourreza & Rafiei, 2023, S. 1). Die Aufgaben variieren sich stark von der Komplexität und in verschiedenen Domänen. Die Ergebnisse der einzelnen Modelle können am Ende miteinander verglichen werden. Die Evaluierung der Benchmarks erfolgt über zwei Hauptmetriken.

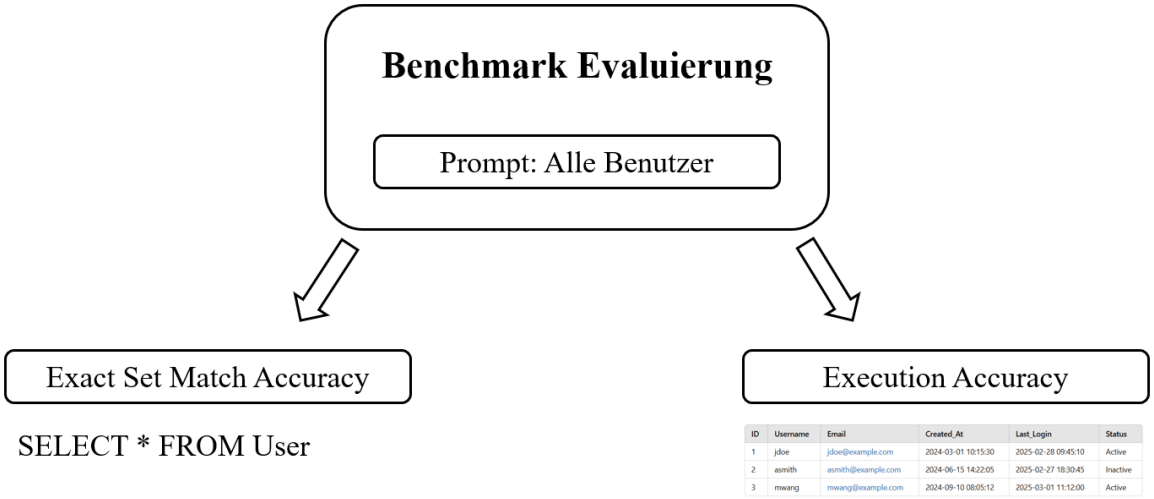

Abbildung 1 Evaluierung Benchmarks

In Abbildung 1 wird die Evaluierung zwischen Exact Set Match Accuracy und Execution Accuracy unterteilt. Ein Exact Set Match Accuracy prüft, ob die Antwort des LLMs die der Referenzlösung übereinstimmt. Das Problem ist, dass zwei verschiedene SQL-Abfragen das gleiche Ergebnis liefern, auch wenn die SQL-Abfrage sich leicht voneinander unterscheiden (Pourreza & Rafiei, 2023, S. 6). Um die Lücke in der Überprüfung zu schliessen, kann anhand der Execution Accuracy die Ausgabe der SQL-Abfrage überprüft werden. Eine Abfrage gilt als korrekt, wenn aus der Testdatenbank die gleichen Daten abgefragt oder manipuliert werden, wie bei der Referenzabfrage (Pourreza & Rafiei, 2023, S. 7). Im Text-to-SQL Bereich existieren bereits einige Benchmarks mit unterschiedlichen Schwerpunkten.

Tabelle 4 Text-to-SQL Benchmarks

Benchmark	Erklärung
WikiSQL	Enthält einfache SQL-Abfragen, die meistens nur eine Tabelle betreffen. Die Tabellen sind mit Daten von Wikipedia befüllt, Nachteil es kann keine komplexen Anfragen mit Joins oder Aggregationen abdecken (Pourreza & Rafiei, 2023, S. 3).
Spider	Enthält 200 verschiedene rationaler Datenbankenschemas mit Fokus auf komplexen Abfragen. Fokussiert auf rein SQL-Abfragen mit Joins, Aggregationen und geschachtelte Unterabfragen. In den meisten Fällen ist dies der de-facto Benchmark für Text-to-SQL Modelle (Pourreza & Rafiei, 2023, S. 4).
BIRD	Enthält mehr als 95 verschiedene Datenbanken in mehr als 35 verschiedenen Domänen. Fokussiert auf eine sehr breite Sammlung von SQL-Abfragen und inkludiert Text und SQL-Abfragen mit der PostGIS Erweiterung (Li et al., 2023, S. 5).

In der Tabelle 4 sind drei verschiedene Text-to-SQL-Benchmarks aufgelistet. Der Spider und der BIRD-Benchmark werden viel in Literatur verwendet, um die Leistung von Text-to-SQL Modellen zu identifizieren. Beide Benchmarks sind legitim und es wird mindestens eine kleine Developer Version zum Testen bereitgestellt. Das Problem ist, dass die gelisteten Benchmarks entweder nur einen Teil PostGIS Abfragen beinhalten oder gar keine PostGIS Abfragen beinhalten. Aus diesem Grund wird für diese Arbeit ein eigener Benchmark erstellt, welcher nur Aufgaben beinhaltet, die nur mit PostGIS gelöst werden können.

4 Methode und Vorgehensweise

Um die Forschungsfrage dieser Arbeit zu beantworten, wird ein mehrstufiges methodisches Vorgehen angewandt. Der erste Teil der Arbeit legt die Basisanwendung voraus, auf der später ein Benutzer und ein Benchmark das Backend verwenden kann. Als erstes werden die Daten vorbereitet, dabei werden Goldtabellen selektiert, die Datenbank mit OpenStreetMap Daten befüllt und Beispiel Einträge für die Feedback-Loops geschrieben. Anschliessend selektieren wir die Modelle und Tools selektiert. Dabei werden die verwendete SQL-Datenbank sowie die LLMs selektiert. Alles wird in einer benutzerfreundlichen Schnittstelle in Form einer Webapplikation bereitgestellt, um natürliche Spracheingaben in SQL-Abfragen umzuwandeln. Anhand Feedback-Loops sollte die Performance der Anwendung verbessert werden. Der zweite Teil der Arbeit ist einen eigenen Benchmark zu erstellen, auszuwerten und die Resultate aus dem Benchmark im Frontend anzeigen.

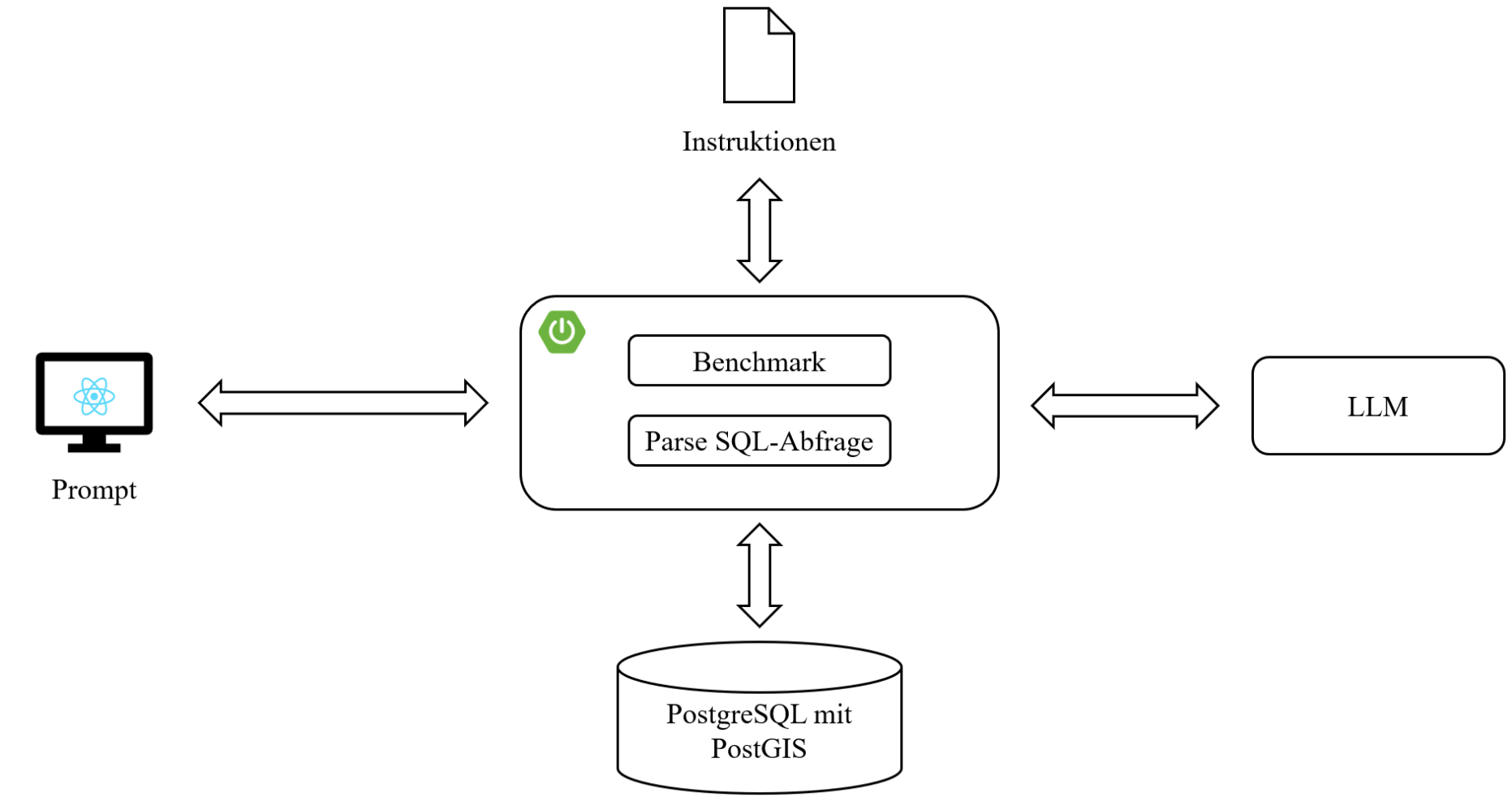

Abbildung 2 Beschreibung der Elemente in der Infrastruktur

In der Abbildung 2 wird die grundlegende Infrastruktur der Applikation aufgezeigt.

Tabelle 5 Elemente der Infrastruktur

Element	Beschreibung
Prompt	Ein Prompt ist eine Texteingabe, die nach Manipulationen und Instruktionen an ein LLM weitergeleitet wird. Ein Prompt beschreibt was beabsichtigt ist und wie das Resultat aussehen soll. Ein Prompt wird in einer Webapplikation durch einen Benutzer eingegeben. Die Webapplikation basiert auf dem React Frontend Framework.
LLM	Ein LLM ist ein Large Language Modell. Es nimmt Prompts in Text-Form entgegen und generiert basierend auf den Bedingungen und Regeln eine Antwort. Für diese Applikation ist das LLM wie das Gehirn und hat die Aufgabe genaue SQL-Abfragen zu generieren.
Benchmark	Der Benchmark hat Zugriff auf Parse SQL-Abfragen und wertet die Leistung der Anwendung, sowie die Leistung der LLMs aus.
Instruktionen	Damit die Antwort immer im gleichen Format generiert wird für weitere Verarbeitung, erstellen wir die Regeln in einem Template Prompt. Neben den Regeln werden die Feedback-Daten als sinnvolle Beispielpaare hinzugefügt zusammen mit dem eigentlichen Prompt des Benutzers.
Parse SQL-Abfrage	Die Antwort vom LLM wird danach ausgeführt. Anhand des Templates wissen wir, in welchem Format die Antwort aussieht. Die SQL-Abfrage kann vom LLM extrahiert werden und ausgeführt werden. Die Antwort der Datenbank wird im Frontend angezeigt.
PostgreSQL mit PostGIS	PostgreSQL ist das Datenbankmodell mit der PostGIS Erweiterung. Primär werden PostGIS, sprich räumliche Daten abgespeichert. In dieser Datenbank werden auch die Favoriten abgespeichert, sowie die Benchmark-Resultate und die Benchmark-Fälle.

Tabelle 5 erklärt alle Elemente, die innerhalb der Abbildung 2 abgezeichnet sind. Ein Prompt wird durch den Benutzer im Frontend in ein Inputfeld eingegeben. Der Benutzer kann weitere Parameter definieren, zum Beispiel welche Feedback-Loops zu verwenden sind. Anschliessend wird im Backend ein Prompt mit allen relevanten Artefakten zusammengestellt, wie zum Beispiel mit den Instruktionen. Die Abfrage wird den LLMs übergeben und wir warten auf eine Antwort. Sobald die Anwendung eine Antwort erhalte, wird den SQL-Code aus der Antwort des LLM extrahiert. Anschliessend wird die SQL-Abfrage ausgeführt und das Ergebnis aus der Datenbankabfrage wird im Frontend angezeigt. Denselben Durchlauf führt auch der Benchmark aus. Während im regulären Betrieb ein Benutzer den Prompt eingibt, wird beim Benchmarking stattdessen der Endpoint der Applikation direkt aufgerufen.

4.1 Daten Aufarbeiten

Die PostgreSQL-Datenbank mit der PostGIS Datenbankerweiterung stellt die technische Grundlage der Arbeit dar. Die OpenStreetMap Daten sind auch in der Schweiz sehr umfangreich und es kann beim Importprozess zu Problemen kommen. Gefahr besteht, dass beim Import etwas falsch geht und die Fehler unbemerkt bleiben. Zu Beginn wird eine leere Datenbank auf PostreSQL erstellt. Danach wird der Importprozess gestartet für die Datenbank mit OpenStreetMap Daten. Das ist in einer Cloud-Umgebung ist nicht trivial aufgrund der Datenmenge. Zuerst wird Schritt für Schritt erklärt, wie die Schweizer OpenStreetMap Daten in eine Supabase PostgreSQL Datenbank mit der Datenbankerweiterung PostGIS importiert werden. Bei LLMs kommt die Qualität des Outputs sehr auf die Qualität des Inputs an. Zuerst müssen ein Paar Favoriten abgespeichert werden, damit LLMs dies als Referenzpunkt verwenden können. Abgespeichert wird immer der Prompt mit der dazugehörigen SQL-Antwort.

4.1.1 Erstellung einer PostGIS Datenbank

Das Erstellen der Datenbank in einer Cloud-Umgebung viele Probleme zubereitet. Im folgenden Paragrafen werden die Schritte diskutiert, die notwendig sind um OpenStreetMap Daten in eine Supabase PostgreSQL Datenbank zu Importieren. In Supabase sollte man zuerst ein neues Projekt anlegen und anschliessend eine neue Datenbank anlegen.

CREATE EXTENSION postgis;

Als nächstes muss die PostGIS Erweiterung aktivieren werden, wenn dies nicht bereits über die GUI erfolgt ist.

CREATE SCHEMA IF NOT EXISTS extensions;

CREATE EXTENSION IF NOT EXISTS hstore WITH SCHEMA extensions;

Als nächstes erstellen wir das Schema Extentions und aktivieren die Erweiterung hstore. Hstore wird für die Speicherung von Key-Value Paaren in PostgreSQL Datenbanken verwendet.

ALTER ROLE postgres SET statement_timeout = '3600000';

Danach erhöhen wir das Timeout auf der Datenbank für die Rolle postgres auf eine Stunde. Wir machen das, denn der Datenimport wird den Standardtimeout überschreiten und der Importprozess wird in der Mitte abgebrochen, der schnellste Weg danach ist eine neue

Datenbank anzulegen. Als nächstes müssen wir die Leistung der Datenbank temporär erhöhen, um OSM-Daten importieren zu können.

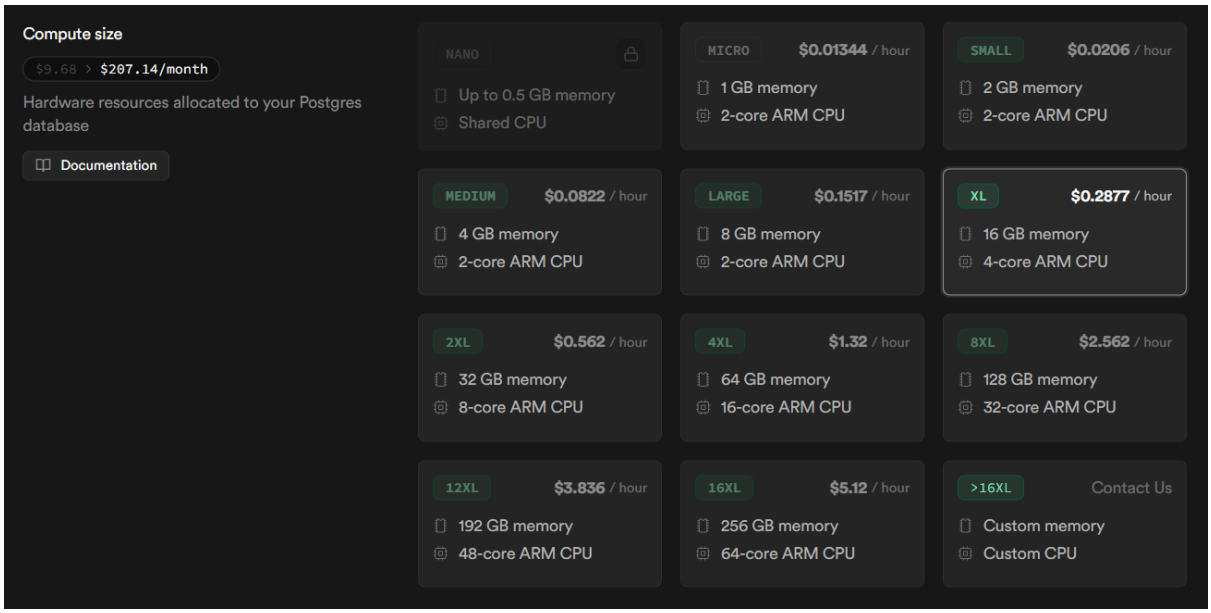

Abbildung 3 CPU der Datenbank festlegen

Unter Settings -> Compute and Disk wird die Compute Size auf mindestens ein XL (16GB, 4-core ARM CPU) erhöht. Abbildung 3 zeigt das Supabase Benutzeroberfläche, welche zu bearbeiten ist. Dies wird die Geschwindigkeit des Importprozess erhöhen und wird vermeiden, dass der Free Tier aufgebraucht wird und während dem Inputprozess ausgebremst werden.

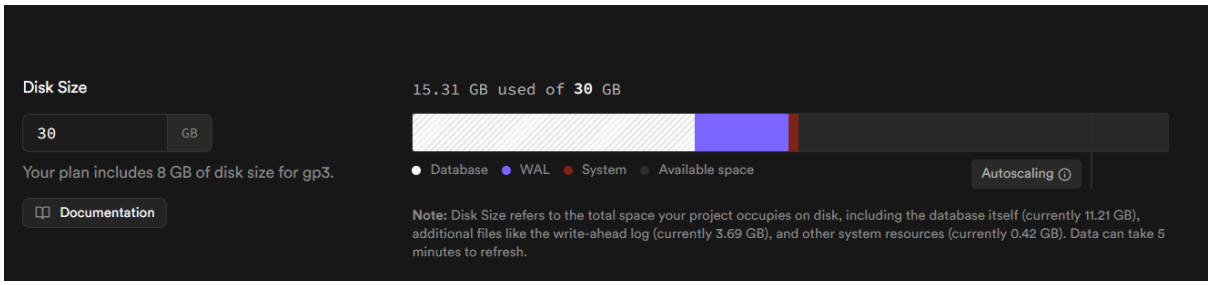

Abbildung 4 Speicher der Datenbank festlegen

Abbildung 4 zeigt die Supabase Benutzeroberfläche. Im Bild ist zu erkennen, dass die Datenbank ungefähr 16 GB gross wird. Die OSM-Daten allein werden auf ungefähr 10 GB Daten umfassen. Der Import wird intensiv WAL schreiben. Also bevor eine Veränderung auf der Datenbank eintrifft, werden sie zuerst in Logdateien hinterlegt, welche auf der Disk gespeichert wird. Das wird dazu führen, dass während dem Import die Disk Size grösser sein wird als die eigentlichen Daten in der Datenbank. Unter *Settings -> Compute and Disk* wird der Disk Size auf 30 GB erhöht, um nicht an Disk Size Grenzen zu stossen während dem Import.

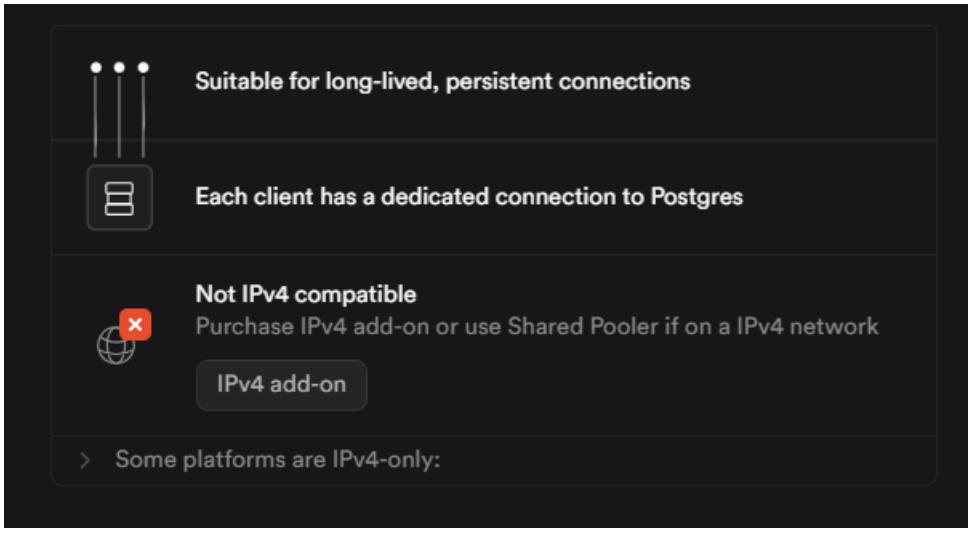

Abbildung 5 IPv4-Zugriff der Datenbank aktivieren

Als Letztes wird eine IPv4-Adresse für den Import eingekauft. Dafür klicke auf Verbinden oben links und unter Direkt Verbinden rechts gibt es die Möglichkeit eine IPv4-Adresse zu kaufen. Die Supabase Benutzeroberfläche wird gemäss der Abbildung 5 aussehen. Wir machen dies für den Import, um eine längere, ununterbrochene Verbindung mit der Datenbank aufzubauen.

Wir benötigen jetzt noch die OSM-Daten, um sie danach in die Datenbank hochzuladen.

Tabelle 6 Herkunft der OSM-Daten

Ressource	Beschreibung
OSM-Daten Schweiz (OSM, 2025)	Open Street Map Daten der Schweiz
Default Style der OSM-Daten (OSM, 2025b)	Konfigurationsdatei für OSM-Tags -> Definiert Key-Value Paare die Importiert werden
osm2pgsql (Burgess et al., 2025)	Programm welches für den Import von OSM-Daten in PostgreSQL mit PostGIS Erweiterung verwendet wird.

Tabelle 6 gibt eine Auflistung der Programme und Dateien, die heruntergeladen werden müssen, um OSM-Daten in eine Datenbank zu importieren. Unter den Dateien befindet sich die OSM-Daten der Schweiz, die Konfigurationsdatei für OSM-Tags und das Programm, welches den Import von OSM-Daten in eine Datenbank ermöglicht.

"C:\Program Files (x86)\osm2pgsql-latest-x64\osm2pgsql-bin\osm2pgsql.exe" --create --slim --
database=postgresql://postgres:DEINPASSWORT@db.gnmlckuqxwcfsxpuftar.supabase.co:5432/postgres --hstore --style=default.style --verbose switzerland-exact.osm.pbf

Im Command Prompt kann jetzt der Import gestartet werden mit diesem Befehl. Osm2pgsql kann auch in den System Variablen hinterlegt werden, dann kann anhand osm2pgsql aufgerufen werden, statt den Pfad zur Executable zu hinterlegen. Navigiere in den Ordner im Command Prompt, wo die default.style und die OSM-Daten befinden. Nach Start des Befehls wird jetzt die OSM-Daten in der PostgreSQL-Datenbank gespeichert. Dieser Prozess wird ungefähr 25 Minuten dauern, sofern alle Schritte bis anhin verfolgt wurden. Nach erfolgreichem Import der OSM-Daten können wieder die kostenpflichtigen Einstellungen zurückgesetzt werden. Der Compute kann wieder auf den Free Tier gesetzt werden (nicht empfohlen, wenn komplexe PostGIS-Abfragen generiert werden). IPv4 kann nach Import auch ausgeschalten werden. Und der Disk Size kann auf 20 GB reduziert werden.

4.1.2 Favoriten einem LLM hinzufügen

Die Applikation lernt mit jeder Anfrage, die als ein Favorit markiert wird. Dabei speichert man die Texteingabe des Benutzers und den generierten SQL. Organisch können somit nur Favoriten abgespeichert werden, die von einem LLM erzeugt worden sind. Das limitiert die Kreativität der LLMs, um dies zu beheben, habe ich zusätzlich weitere Text Input in einem Style geschrieben, wie ein Benutzer es eingeben würde mit den dementsprechenden SQL-Abfragen. Diese Werte in der Favoritentabelle werden jedem Prompt mitgegeben, sofern dem Programm mitgeteilt wird, dass der User-Feedback-Loop verwendet werden soll.

4.2 Modell und Tool-Auswahl

Es werden verschiedene der AI-Modelle ausprobiert und ein Modell, das funktioniert, wird dann ausgewählt. Dabei besteht das Ziel nicht zu sagen, welches AI-Modell besser ist als das andere, was unter den gegebenen Bedingungen nicht fair wäre, sondern die Frage zu beantworten, ob LLMs in der Lage sind, PostGIS-Abfragen zu schreiben. Die Datenbank wird eine PostgreSQL-Datenbank mit der PostGIS Erweiterung sein.

4.2.1 Auswahl der LLMs für SQL-Generierung

Es werden verschiedene AI-Modelle ausprobiert und ein Modell, welches die Anzahl Tokens als Input entgegennehmen kann, wird dann ausgewählt. Die Anfragen, welche wir schicken, beinhalten viele Tokens als Input und nicht jede LLM-Version kann 25'000 Tokens oder mehr entgegennehmen. Die Arbeit verwendet die LLMs die stand Mai 2025 in Coding Benchmarks vertreten sind. Namentlich werden Gemini, Grok, Deepseek und ChatGPT miteinander verglichen. Dabei besteht das Ziel nicht zu sagen, welches AI-Modell besser ist als das andere, was aufgrund der Input-Bedingungen unfair ist, sondern die Frage zu beantworten, ob LLMs in der Lage sind PostGIS-Abfragen zu schreiben. Die Datenbank wird eine PostgreSQL-Datenbank mit der PostGIS Erweiterung sein.

Tabelle 7 Verwendete Modelle der LLMs

LLM	Modell
ChatGPT	GPT_4O_MINI
Claude	CLAUDE_3_5_HAIKU_LATEST
Deepseek	DEEPSEEK-CHAT
Gemini	GEMINI-2.0-FLASH
Grok	GROK-2-LATEST

Tabelle 7 listet alle LLMs, die in der weiteren Arbeit eingebaut werden und deren Modelle. ChatGPT wird mit der GPT_4O_MINI gearbeitet, für Claude wird CLAUDE_3_5_HAIKU_LATEST verwendet. Für Deepseek wurde DEEPSEEK-CHAT selektiert. Für Gemini wurde GEMINI-2.0-FLASH ausgewählt und für Grok wurde GROK-2-LATEST ausgewählt. Die verwendeten Modelle sind nicht immer die neuesten oder die besten, es sind aber Modelle, die eine Input-Menge von mindestens 25'000 Tokens verarbeiten können und von den Anbietern/Herstellern gepflegt oder gewartet werden.

4.2.2 Auswahl der Testdatenbank (PostGIS, PostgreSQL)

Die Datenbank stellt die Grundlage für jede Text-To-SQL-Anwendung dar. Ziel ist es, die APP möglichst realitätsnah zu gestalten. Wie es auch in der Privatwirtschaft üblich ist, wird die Datenbank in der Cloud gehostet. Genauer gesagt befindet sich die Datenbank bei Supabase in der Region eu-central-2 (Zürich). Gemäss Supabase läuft die Datenbank auf einer EC2-Instanz in Zürich. Supabase ist ein neuer Anbieter im Bereich cloudbasierter Datenbanken und stellt sich als Open-Source-Alternative zu Firebase dar. OpenStreetMap-Daten sind gross und stellen eine Herausforderung für den Import dar. Für die Bachelorarbeit verwenden wir ausschliesslich die OSM-Daten aus der Schweiz. Die OSM-Daten kann direkt in eine SQL-Datenbank gespeichert werden, die über die Datenbankerweiterung PostGIS verfügt. PostGIS erlaubt es, komplexe Abfragen für geobasierte Daten in einer SQL-Datenbank zu schreiben.

4.3 Webapplikation-Entwicklung

Die Webapplikation dient als benutzerfreundliche Schnittstelle zur Text-to-SQL-Anwendung und ermöglicht es Nutzern, natürliche Spracheingaben einzugeben und direkt die generierten SQL-Abfragen sowie die Abfrageergebnisse anzuzeigen. Das Frontend wird als React-Anwendung umgesetzt und nutzt Tailwind CSS sowie Shadcn, um ein modernes und responsives Design sicherzustellen. Die Benutzeroberfläche umfasst ein Eingabefeld für natürliche Sprache, eine Darstellung des generierten SQL-Codes und eine Visualisierung der Abfrageergebnisse. Folgend werden die Hauptfunktionalitäten wie der Prompt Buildern, Syntax-Feedback-Loop und Benchmark in der technischen Umsetzung beschrieben.

4.3.1 Query Execution

Alle Anfragen werden über die Query Execution verarbeitet. Die Query Execution beinhaltet alle Einstellungen der Anfrage. Ein wesentlicher Punkt für die Query Execution ist, dass dieser Endpoint einerseits im Frontend verwendet wird und andererseits durch den Benchmark aufgerufen wird.

Tabelle 8 Parameter für die Ausführung der Anwendung

Parameter	Beschreibung
Prompt	Als String wird hier die Eingabe des Benutzers übermittelt.
UserFeedbackLoop	Ist ein boolean, also ein true oder false, ob User-Feedback-Loop verwendet werden sollte oder nicht
SyntaxFeedBackLoop	Ist ein boolean, also ein true oder false, ob Syntax-Feedback-Loop verwendet werden sollte oder nicht
AllowEmptyResponse	Ist ein boolean, also ein true oder false, ob eine leere Datenbankantwort in Ordnung ist.
Model	Ist ein String mit dem Modell, welches verwendet werden sollte.

Die Tabelle 8 beinhaltet alle Parameter, die in einem Request vorhanden sind. Ein Request enthält den Prompt. Ein Prompt ist ein String, der dem LLM als Eingabe übergeben wird. Zum einen handelt es sich um den UserFeedbackLoop, zum anderen um den SyntaxFeedbackLoop. Wenn die booleschen Parameter auf ‚true' gesetzt sind, wird die dementsprechende Feedback-Loop für die Abfrage eingeschaltet. Zudem kann eine leere Antwort zugelassen werden. Dies ist machbar, weil OpenStreetMap Daten der Schweiz Lücken in den Daten enthalten. Manchmal gibt es keine Daten, die für eine spezifische Region auffindbar sind. Zuletzt wird im Request über den Model-Parameter mitgeteilt, welches LLM diese Anfrage bekommt.

4.3.2 Prompt Builder

Um immer denselben Aufbau der Antwort der LLMs zu erhalten, werden die Prompts mit vielen relevanten Elementen jeweils zusammengebaut. Gewisse Elemente sind optional und können vom Benutzer gesteuert werden, andere Elemente hingegen werden mit jeder Anfrage geschickt.

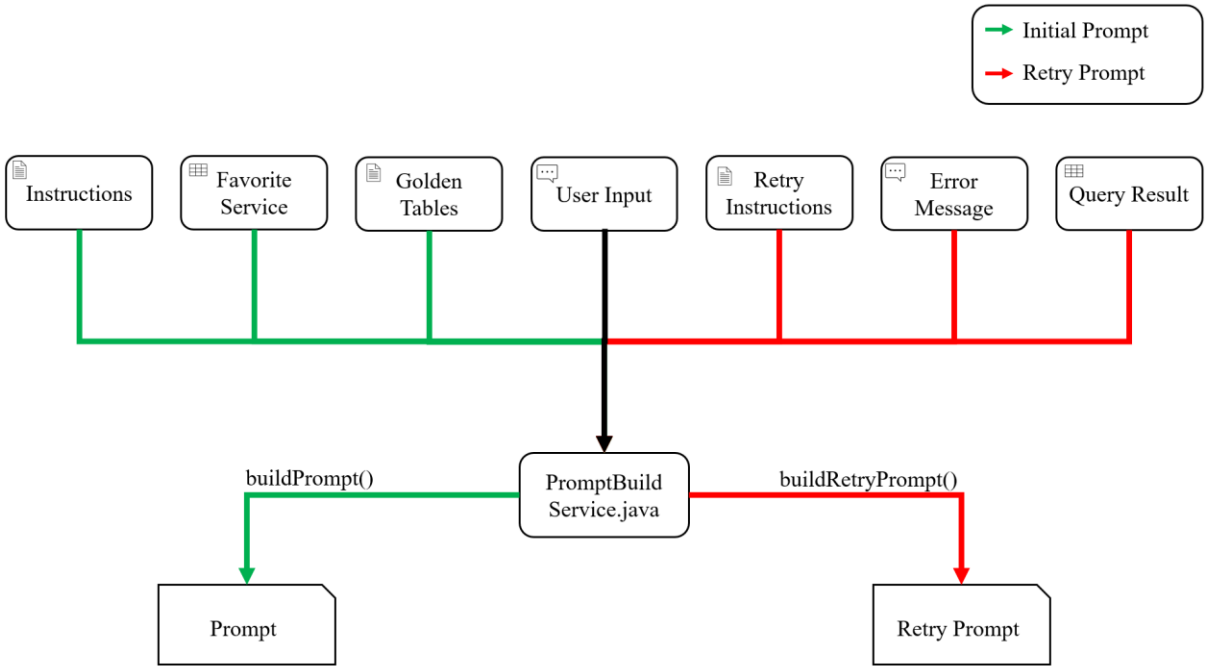

Abbildung 6 Aufbau der PrompBuildService Klasse

Abbildung 6 zeigt alle Elemente, die von der PromptBuildService Klasse verwendet werden. Die Instruktionen beinhalten die Beschreibung der Rolle und Aufgabe. Die Instruktion wird in Form einer Textdatei im Ressource-Order abgespeichert. Es gibt eine separate Instruktion für den Fall, dass es ein Retry Versuch ist. Der Favorite Service ruft die Favoriten aus der zugehörigen Tabelle in der Datenbank aus. Die Goldtabellen sind das Grundgerüst der Datenbank. Die Goldtabellen definieren die Tabellen, Felder und deren Datentypen. Dies ist das Datenbankschema der OpenStreetMap-Datenbank im JSON-Format und ist im Ressource-Order abgespeichert. Der Prompt des Benutzers ist der User Input. Error Message ist die Antwort der Datenbank im Falle eines Fehlers und die Query Result sind die Resultate, welche von der Datenbank geschickt werden. Die in Rot markierten Pfade sind relevant für den Retryprompt, die in grün markierten Elementen sind relevant für den Standardprompt.

In grün sind alle Ressourcen markiert, die für den initialen Prompt verwendet werden. Der Standardprompt beinhaltet Goldtabellen, was als Referenzpunkt für den LLM dient, um die

Richtigkeit der generierten SQL-Abfragen selber zu evaluieren (Han et al., 2024, S. 2). Falls der User-Feedback-Loop aktiviert wurde, werden die Favoriten am Prompt zugefügt. Der initiale Prompt verwendet die Instruktionen mit folgendem Wortlaut:

> *"You are an Postgres expert with deep understanding of writing postgis sql queries. On the top you receive good examples of queries you already wrote. Then you receive the golden tables for the database, this gives you the layout for the entire database. Your task is it to write and respond ONLY with a postgis SQL query. Do not write any introductions or reasoning why you wrote the SQL query as you did. When selecting the region of queries, please follow the given table to correctly specify the data you need to select.*
>
> *| admin_level | Typical Entity |*
>
> *| ---------- | ---------- |*
>
> *| 2 | Country |*
>
> *| 4 | Canton (state/region) |*
>
> *| 6 | District or Region (optional) |*
>
> *| 8 | Municipality / City / Commune |*
>
> *| 10 | Suburb / Quarter / Local Division |"*

Die Instruktionen geben dem LLM an, dass er ein PostgreSQL Experte ist und viel Verständnis von PostGIS-Abfragen hat. Dem LLM wird mitgeteilt, welche weitere Elemente im Prompt stehen. Danach wird erklärt, wie die Antwort aussehen soll, nämlich ohne Text, der die Abfrage begründet und erklärt, dass nur die essenziellen Tabellen auszuwählen sind.

In Rot sind alle Ressourcen markiert, die für den Retry Prompt verwendet werden. Der Retry Prompt beinhaltet eigene Instruktionen, Instruktionen mit folgendem Wortlaut:

> *„The last SQL query did not return any data or had an error.*
>
> *Try a different approach. Maybe rephrase the condition, loosen filters, or use an alternative spatial method.*
>
> *Respond ONLY with valid SQL. No code blocks, no extra text."*

Die Instruktion besagt, dass ein Fehler passiert ist. Ziel ist es nochmals eine valide SQL-Abfrage zu erstellen aber durch eine andere Art und Weise. Auch hier wird mitgeteilt, dass nur mit der SQL-Abfrage geantwortet werden soll. Zudem wird der Fehler, der durch die Datenbank generiert wird, dem Prompt der von der Datenbank generierte Fehler mitgegeben. Auch hier wird die Aufgabe des Benutzers nochmals mitgeschickt.

4.3.3 Favorites und userFeedBackLoop

Die Favoriten können in einer separaten Tabelle in der PostgreSQL-Datenbank abgespeichert werden. Alle Datensätze in der Tabelle werden mit der ersten Query mitgegeben, wenn userFeedbackLoop als true mitgegeben wird. Die Tabelle in der Datenbank beinhaltet die Spalte ID, Prompt und SQL. Durch Favorites kann das Programm nach jeder Query mitlernen, sofern dies vom Benutzer erwünscht wird. Aufgrund der Auswirkung den die Favoriten haben, können Favoriten nicht nur gespeichert und gelesen werden, sondern wieder gelöscht werden über die ID. Die Favoriten werden, wenn userFeedbackLoop als true mitgegeben wird, in Promptbuilder-Klasse mitgegeben.

4.3.4 SyntaxFeedbookLoop

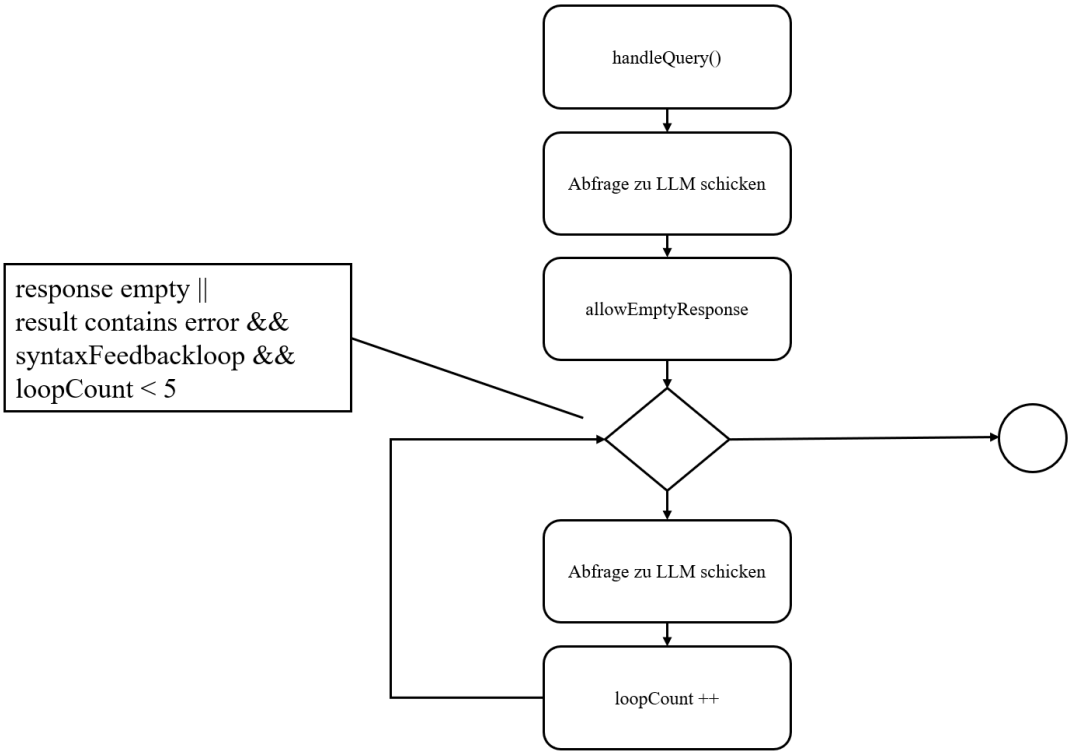

Abbildung 7 Logik syntaxFeedbackLoop

Abbildung 7 zeigt die Logik des Syntax-Feedback-Loops. Zuerst wird eine Anfrage an die Query Execution geschickt, wobei die Funktion handleQuery ausgeführt wird. Danach wird die Anfrage an das LLM geschickt. Wenn eine leere Antwort erlaubt ist und die Antwort leer bleibt, wird eine entsprechende Meldung in der Datenbankantwort gespeichert. Der Syntax-Feedback-Loop begrenzt sich auf zwei Aspekte. Darf die Antwort leer sein und ob die Antwort einen Fehler von der Datenbank enthält. Es gibt viele Abfragen, die von einem Benutzer gestellt werden, die keine OSM-Daten haben. Dies ist optimal für Benutzer, die wissen, dass die Informationen zur Anfrage vorhanden sind und gut für Benutzer, die nicht 5-mal auf eine Korrespondenz warten mit dem LLM, bis ein Ergebnis angezeigt wird. können sie allowEmptyResponse auf falsch setzen und eine nicht-leere Antwort der Datenbank zu bekommen. Wenn allowEmptyResponse auf false oder feedbackLoop auf true ist, wird in eine Schleife eingegangen. Diese Schleife hat drei Bedingungen, die erfüllt sein müssen.

- Die Antwort ist leer oder enthält einen Fehler
- syntaxFeedbackloop muss true sein
- loopCount muss kleiner als 5 sein

Mit jedem Durchgang in der Schleife wird der Fehler mit dem originellen Prompt an den originellen LLMs zurückgegeben. Das LLM hat keinen Nachrichtenverlauf, wird jedes Mal in einem neuen Chat gestartet. Der loopCounter wird mit jedem Durchgang um 1 inkrementiert. Die gleichen 4 Bedingungen werden überprüft, falls ein neuer Fehler auftritt, wird dieser erneut übergeben. Das heisst, die LLMs werden nicht informiert, über die Historie der generierten SQL-Abfragen. Nach dem fünften Durchlauf wird die letzte Antwort dem Benutzer angezeigt. Es kann sein, dass es eine leere Antwort ist, dass ein Fehler besteht oder dass die Antwort eine korrekte Abfrage zurückliefert.

4.3.5 Benchmark

Benchmark ist die Testmethode der Anwendung. Es gibt eine Benchmark_cases Tabelle, die mit einem Prompt, einer case_id und dem zugehörigen SQL befüllt ist. Dabei lassen sich drei Einstellungen definieren:

- userFeedbackLoop
- syntaxFeedbackLoop
- runNumber

Dies ermöglicht es die Effektivität von Syntax-Feedback-Loop und User-Feedback-Loop zu vergleichen. Manuell muss eine neue Runnumber für jede individuelle Benchmark-Einstellungen gewählt werden. Im Rahmen des Benchmarks werden zahlreiche Informationen generiert. Ein Teil der Daten wird automatisch generiert, ein anderer muss manuell ergänzt werden. Die Daten werden nicht überschrieben, damit im Vergleich auf die alten Resultate zurückgegriffen werden kann. Diese Informationen werden in der Tabelle benchmark_results abgespeichert:

Tabelle 9 SQL-Tabelle für benchmark_results

Spalte	Beschreibung
ID	Unique ID
is_correct	Vergleich des Inhaltes der Tabellen zwischen der Benchmark-Vorgabe und der vom LLM generierten SQL-Abfrage.
query	Die generierte SQL-Abfrage des LLM-Modells
response_time	Zeit, die es braucht, vom Start Punkt des Benchmarks, bis eine Antwort von der Datenbank zurückgegeben wird.
run_number	Ist eine Nummer, die vom Benutzer gesetzt wird, sodass verschiedene Runs in der gleichen Tabelle abgespeichert werden mit verschiedenen Feedback-Loop-Konfigurationen.
retry_number	Anzahl loops innerhalb des syntax_feedback_loops
llm	Welches LLM wurde ausgewählt
user_feedback_loop	Einstellung: ist user_feedback_loop eingestellt?
syntax_feedback_loop	Einstellung: ist syntax_feedback_loop eingestellt?
created_at	Timestamp wann der Datensatz der Datenbank geschickt wird.
human_correction	Wird mit is_correct initialisiert. Dieses Feld kann durch den Benutzer eine manuelle Korrektur durchgeführt werden kann. Is_correct unterliegt bereits einer sehr strengen Kontrolle, daher wird generell nur von false auf true umgewechselt
issue_type	Damit jeder Fehler analysiert wird, kann der Fehler kategorisiert werden. Dies wird auch manuell durch den Benutzer durchgeführt.

Benchmark_results Tabelle beinhaltet viele Spalten. Tabelle 9 hat alle Spalten in benchmark_results gelistet mit einer Erklärung. Die Spalte is_correct ist ein boolescher Wert und ist true, wenn der Inhalt der Datenbankantwort übereinstimmt mit der Datenbankantwort der Beispiel SQL-Abfrage. Die query ist die generierte SQL-Abfrage. Die response_time besagt wie viel Zeit seit dem Start des Prozesses bis zur ausgeführten Datenbankantwort vergangen ist. Der run_number gibt an zu welchem Benchmark-Durchlauf die Frage gehört. Die retry_number besagt, wie oft die Frage durch den Syntax-Feedback-Loop gegangen ist. Die Einstellungen des Benchmarks werden in user_feedback_loop und syntax_feedback_loop gespeichert. Die Spalte human_correction wird mit is_correct initialisiert, jedoch manuell

angepasst, falls die Antwort befriedigt war. Die Spalte issue_type erklärt, was das Problem der SQL-Abfrage war. Issue_type wird auch manuell geführt.

Der Benchmark wird Limiten erreichen und wird zwischen den einzelnen Abfragen 1 Minute pausiert, um keine Rate Limiten zu erhalten. Weil der Benchmark mehrmals Probleme mit den LLMs erhalten wird, gibt es eine Restart Logik. Die Restart Logik überprüft, ob eine Antwort bereits existiert für den ausgewählten Durchlauf für den LLM und dem Benchmark Fall, wenn eine Antwort bereits existiert, wird dieser Fall übersprungen.

4.3.6 Frontend

Das Frontend zeigt ausschliesslich Daten an, es gibt sehr wenig Logik im Frontend, das Model-View-Controller-Framework wird daher angewendet. Das Frontend beinhaltet vier verschiedene Pages.

Homepage: Text-to-Postgis Abfrage, es besteht ein Eingabefeld, wo ein Benutzer seinen Prompt eingeben kann. Nach dem Absenden wird eine Ladeanimation angezeigt. Wenn eine Antwort eingekommen ist, wird der SQL angezeigt. In diesem SQL kann rechts mit einem Herz versehen werden, dies führt dazu, dass die SQL-Abfrage als Favorite abgespeichert wird. Eine Karte wird angezeigt, falls sich Punkte oder Polygone im Ergebnis der Datenbank befinden. Alle Antworten der Datenbank werden in einer Tabelle angezeigt.

Favorite: Unter der Favorite-Seite können die Favoriten verwaltet werden. Es wird eine Tabelle erstellt mit allen Favoriten Einträgen. In der rechten Spalte befindet sich ein Delete-Button. Nach klicken des Delete-Buttons wird der Datensatz gelöscht und im Frontend wird ein Refresh der Seite getätigt.

Benchmark: Auf der Benchmark Seite werden alle Informationen der Resultate angezeigt. Die Resultate werden im Backend per SQL-Abfragen geladen. Keine der angezeigten Daten sind fest kodiert ausser die Kostentabelle. Die Kostentabelle wurde von Hand und selbst geführt und eingetragen. Eine Tabelle zeigt alle Abfragen eines Durchlaufs an, wenn man auf einen Run klickt, wird exakt dieser Run und alle generierten SQL abfragen für alle LLMs angezeigt sowie der Prompt, sprich der Fall und die Beispiellösung. Auch die Fehler werden auf dieser Seite für jeden Benchmark-Test angezeigt.

4.4 Evaluierung und Feedback

Es gibt zwei verschiedene Varianten wie Feedback-Loops umgesetzt werden. Durch die Webapplikation kann ein Benutzer, nachdem eine SQL-Abfrage erstellt wurde mit einem Herzsymbol markieren, dass die Abfrage korrekt war und die Abfrage wird als Beispiel-Datensatz unter der Favoriten-Tabelle abgespeichert. Die zweite Variante ist durch korrekte Beantwortung einer Text-to-SQL-Frage. Wenn die Frage richtig beantwortet wird, geht die Frage in keinen Syntax-Feedback-Loop. Das heisst das Modell wird während des Benchmarks dazulernen, sobald eine Abfrage richtig beantwortet wurde. Zur Evaluation gehört auch der Benchmark, denn der Benchmark enthält eigene Aufgaben, die beantwortet werden.

4.4.1 Automatische Bewertung der SQL-Generierung

Weil es keine öffentlich zugängliche Benchmark gibt, welche auf PostGIS-Abfragen limitiert sind, wurde ein eigenen Benchmark erstellt. In einer Tabelle namens Benchmark_cases befinden sich 25 Einträge mit Aufgaben, die von einfachen bis zu schwierigen Aufgaben reichen. Die einfachen Aufgaben befinden sich am Anfang des Benchmarks, sprich Fälle 1-6, mittelschwere Aufgaben von 7 bis 13, schwierige Aufgaben für die Fälle 14-17, sehr schwierige Aufgaben 18-21 und spezielle Aufgaben 22-25. Alle Aufgaben enthalten PostGIS-Anforderungen. Alle Benchmark-Fälle sind in der folgenden Tabelle gespeichert:

Tabelle 10 SQL-Tabelle für benchmark_cases

Spalte	Beschreibung
id	Unique ID
expected_sql	Die eigentliche Aufgabe als Prompt
query	Eine Beispiel SQL-Lösung
difficulty	Definiert die Schwierigkeit
region	Ort, wo die Aufgabe sich befindet.
Tags	Tags für die Einteilung

Tabelle 10 enthält alle Spalten, die in benchmark_cases vorkommen. Die Spalte Difficulty, Region und Tags werden nicht in den Resultaten angesprochen, weil diese drei Spalten für das Erstellen der Benchmark wichtig waren. Diese dienen dazu, die Verteilung von Aufgaben nach Schwierigkeitsgrad und Typ besser überprüfen zu können. Die Aufgabenstellung befindet sich in query, die Musterlösung in expected_sql. Der Benchmark ist nur über ein Rest interface aufrufbar. Im Frontend werden lediglich die Ergebnisse angezeigt. Die folgenden Parameter werden für den Benchmark angefordert: SyntaxFeedbackLoop, UserFeedbackLoop und

runNumber. Die App prüft nicht, ob ein Run stets mit denselben Einstellungen durchgeführt wird. Ein Run kann mehrmals abbrechen, für dies gibt es eine Skip. Auf Datenbankebene wird iterativ über alle LLMs und Runs gegangen, um bereits verarbeitete Kombinationen zu überspringen. Für eine einfachere Fehlerbehandlung werden die LLMs nacheinander abgefragt. Die SQL-Generierung wird automatisch durch den benchmark überprüft. Hierfür wird eine exakte 1:1-Überprüfung der zurückgegebenen Daten durchgeführt. Die Tabellenüberschriften werden ignoriert, es wird nur verglichen, was innerhalb der Antworttabelle existiert. Damit dies funktioniert, sind alle Prompts der Benchmark Fälle so deklariert, dass eine Sortierung gemacht werden muss. Fehlt die Sortierung in der Antwort, wird dies ebenfalls als falsch gewertet.

Nachdem der Benchmark durchlaufen wurde bei aktivierter Option syntaxFeedbackLoop und userFeedbackLoop, habe ich alle fehlerhaften Benchmarks manuell überprüft, kategorisiert, und die Korrektheit der generierten SQL-Abfrage manuell im Feld human_correction mit true oder false markiert.

4.4.2 Implementierung eines Feedback-Loops zur Modellverbesserung

Die Applikation generiert einen Mehrwert durch zwei Feedback-Loops. Das Ziel von Feedback-Loops ist die Antwort der LLMs zu verbessern, indem die Prompts gezielt mit relevanten Informationen angereichert werden, um zum einen Fehler zu beheben und zum anderen Beispiele zu liefern, sodass sich die Applikation mit der Nutzung kontinuierlich verbessert.

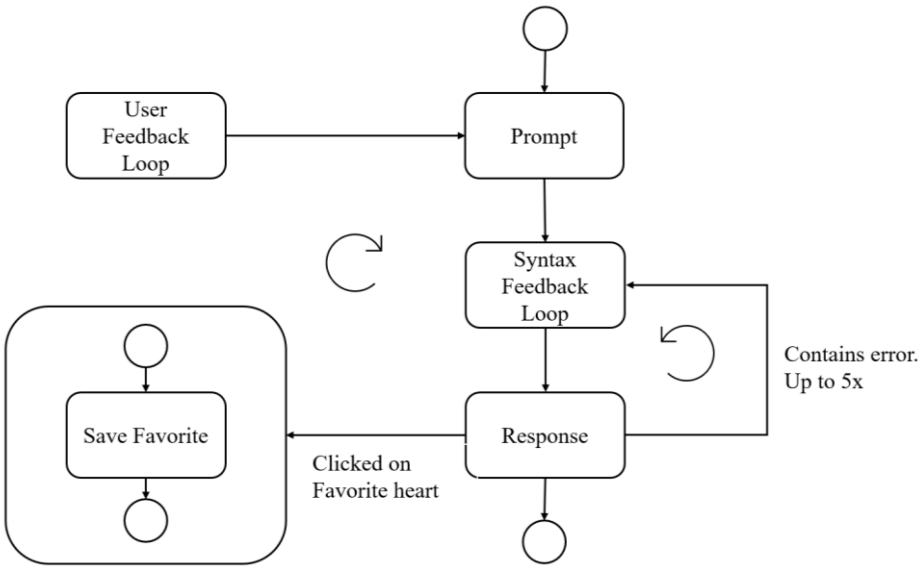

Abbildung 8 Umgesetzte Feedback-Loops

Abbildung 8 illustriert beide Feedback-Loops. Zum einen der Syntax-Feedback-Loop mit einem Restart Pfeil auf der rechten Seite markiert, überprüft, ob die Antwort der Datenbank leer ist oder einen Fehler enthält. Dies wird bis zu 5-mal überprüft, nach dem fünften Versuch wird diese dem Benutzer angezeigt, egal ob es weiterhin einen Fehler beinhaltet. Auf der linken Seite gibt es den User-Feedback-Loop. Der User-Feedback-Loop ist ein teilautomatisierter Prozess. Benutzer können jederzeit SQL-Abfragen zu ihren Favoriten hinzufügen. Die Favoriten werden dann mit jedem neuen Request mitgeschickt.

4.5 Systemvoraussetzungen

Bevor die Applikation gestartet werden kann, wird mindestens eine Java Developement Kit Version von 17 benötigt. Maven muss auch auf dem System installiert sein. NodeJS muss auch auf dem System installiert sein, wenn das Frontend gestartet werden soll.

Das System benötigt eine PostgreSQL Datenbank mit der PostGIS-Erweiterung. Die Anmeldedaten sind in der application.properties Datei zu hinterlegen. Ferner benötigt die Anwendung API Keys für ChatGPT, Claude, Deepseek, Gemini und Grok. Die API Keys werden auch in der application.properties hinterlegt.

spring.application.name=text-to-sql

spring.datasource.url=jdbc:postgresql://

spring.datasource.username=postgres.

spring.datasource.password=XXXX

spring.jpa.database-platform=org.hibernate.spatial.dialect.postgis.PostgisPG95Dialect

spring.jpa.hibernate.ddl-auto=none

openai.api.key=sk-proj-

gemini.api.key=AIz

claude.api.key=sk-

deepseek.api.key=sk-

grok.api.key=xai-

Die application.properties Datei sollte etwa wie das Beispiel oberhalb aussehen.

Unter Maven erstelle ein clean install, um fehlende Abhängigkeiten in deine lokale Abhängigkeiten Bibliothek herunterzuladen.

Danach kann das Backend gestartet werden. Klicke dafür auf eine Java Klasse und selektiere innerhalb des Code-Editor deiner Wahl auf run. Die Backend Applikation wird auf localhost:8080 laufen. Anhand Postman können folgende Abfragen dem Backend mitgeteilt werden: https://documenter.getpostman.com/view/26856010/2sB2j1hCkg

Das Frontend kann mit dem Befehl npm run start gestartet werden und ist über localhost:3000 aufrufbar.

Alternativ kann die Webseite unter https://www.text-to-postgis.com aufgerufen werden.

5 Ergebnisse

Die Arbeit liefert zwei zentrale Ergebnisse, die im folgenden Kapitel präsentiert werden. Zum einen die funktionale Anwendung selbst. Anhand von Bildschirmaufnahmen wird erläutert, wie jede Seite funktioniert und welche Funktionen sie bereitstellt. Zum anderen werden die Ergebnisse des Benchmarks vorgestellt. Die Benchmark-Ergebnisse sind in mehrere Kategorien unterteilt. Der Abschnitt Benchmark-Ergebnisse geht auf die einzelnen Durchläufe des Benchmarks ein. Dabei wird gezeigt, wie sich die Ergebnisse im Laufe der Durchläufe verbessert haben, insbesondere in Abhängigkeit der aktivierten Feedback-Mechanismen. Anschliessend erfolgt eine Fehleranalyse sowie eine Auswertung der Benchmark-Leistung.

5.1 Funktionale Ergebnisse der Webanwendung

Das Frontend wurde mit Tailwind CSS und Shadcn-Komponenten umgesetzt. Frontend und Backend wurden mit Hilfe von GitHub Copilot in Visual Studio Code entwickelt. Die Anwendung ist unter *text-to-postgis.com* öffentlich zugänglich. Die Webanwendung besteht aus drei verschiedenen Komponenten: Home, Favorites und Benchmark. Die Benchmark-Seite enthält zudem eine Detailansicht der Benchmark-Ergebnisse pro Fall.

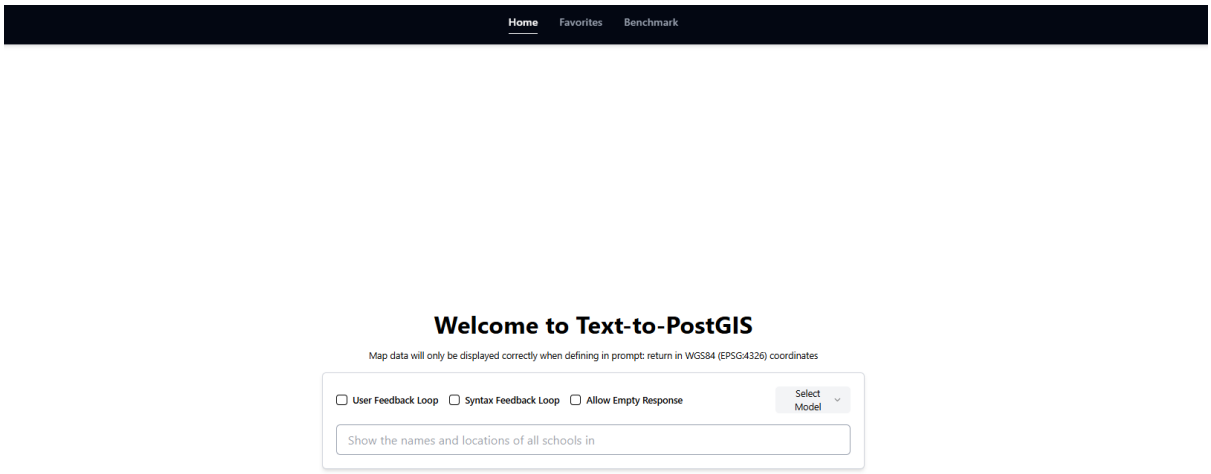

Abbildung 9 Webanwendung Homepage

Abbildung 9 zeigt eine Bildschirmaufnahme der Startseite und enthält ein Eingabeformular, welches die Konfiguration und Eingabe des Prompts ermöglicht. Oben kann definiert werden, welche Feedback-Loops verwendet werden sollen. Zudem kann eine leere Antwort auch

erlaubt werden, um die Kosten gering zu halten und die Antwortzeit zu verkürzen. Dies ist relevant, weil OpenStreetMap-Daten stark in ihrer Genauigkeit und Struktur variieren, kann es vorkommen, dass vorhandene Daten nicht gefunden werden. Rechts neben den Checkboxen befindet sich ein Auswahlmenü zur Auswahl des gewünschten LLM-Anbieters. Hier kann zwischen ChatGPT, Claude, Deepseek, Gemini und Grok entschieden werden. Darunter befindet sich ein Eingabefeld, in dem durch einen Typewriter-Effekt gespeicherte Favoriten als Platzhalter eingeblendet werden. Im Eingabefeld werden die gewünschten Daten als Texteingabe hinterlegt. Damit die Daten später richtig angezeigt werden, muss in der Texteingabe stehen, dass das WGS84 Koordinaten System für Antwort formatiert werden sollte. Durch die Entertaste wird der Prompt abgeschickt und eine Ladeanimation wird angezeigt. Die Wartezeit kann bis zu mehreren Minuten sich strecken. Die Antwort wird unterhalb des Eingabefeldes angezeigt. Zuerst wird die generierte SQL-Abfrage formatiert in einem Codeblock angezeigt Innerhalb des Codeblocks befindet sich rechts ein Herzsymbol. Wenn daraufgeklickt wird, werden der Prompt und die SQL-Antwort als Favorit gespeichert. Falls sich Koordinaten in der Antwort befinden, wird unterhalb des Codeblocks eine Karte mit den Ergebnissen angezeigt. Unterhalb der Karte befindet sich eine Tabelle, die alle von der Datenbank zurückgegebenen Daten der SQL-Abfrage enthält.

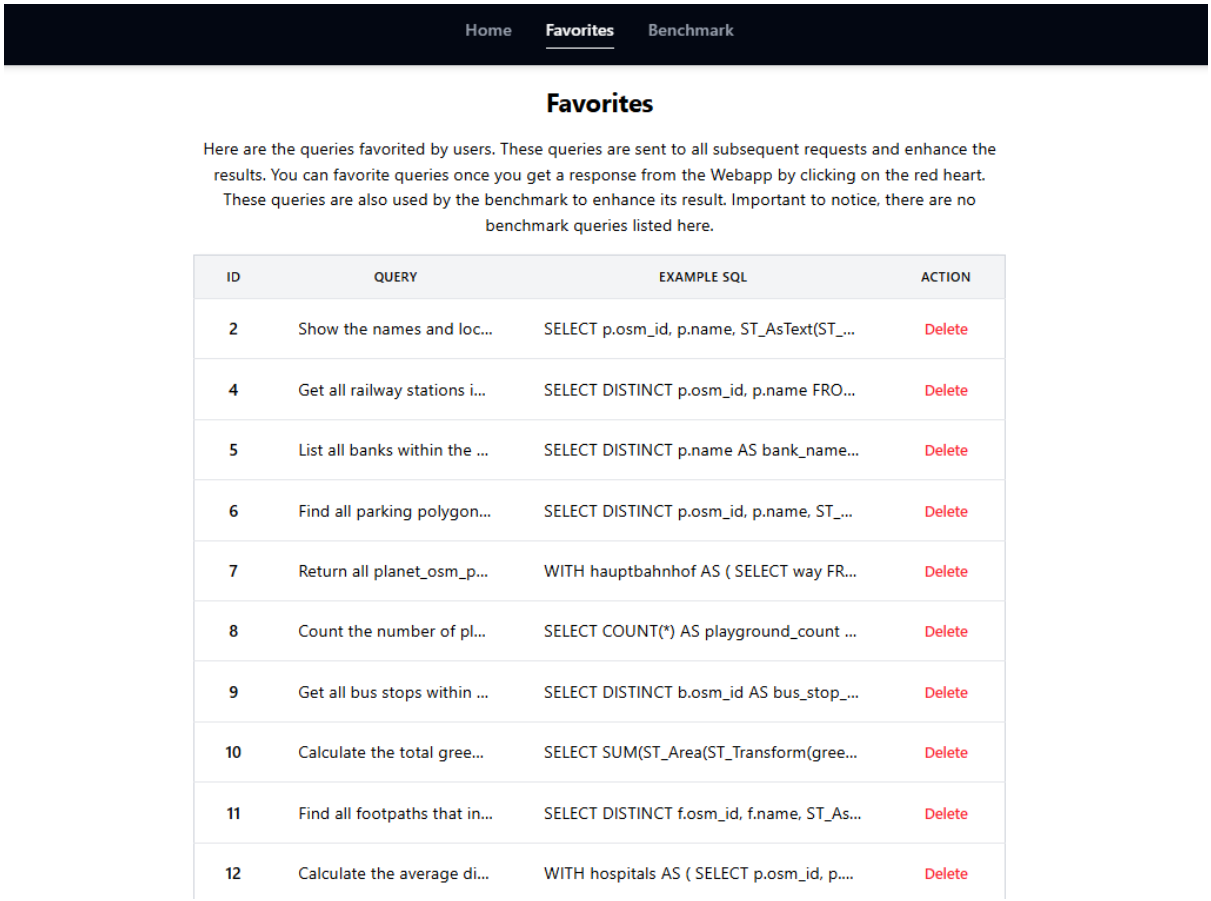

Abbildung 10 Webanwendung Favoriten

Abbildung 10 zeigt die Favoritenseite der Anwendung. Auf der Favoritenseite sind alle vom Benutzer gespeicherten Favoriten aufgeführt. Zu jedem Favoriten sind die ID, der Prompt und die zugehörige Beispiel-SQL-Abfrage sichtbar. Viele der ersten Einträge habe ich manuell erstellt, während ich am Aufbau des Benchmarks gearbeitet habe. Von den rund 43 erstellten Benchmark-Fällen habe ich 25 für den eigentlichen Benchmark reserviert und die restlichen unter Favoriten abgespeichert, um der Applikation eine solide Ausgangsbasis zu geben. Unter Favoriten befinden sich keine aktiven Benchmark-Fälle, der Benchmark greift jedoch auf die Favoritenliste zurück, um bessere SQL-Abfragen zu generieren. Alle Favoriten können gelöscht werden etwa, wenn auf der Startseite versehentlich ein Prompt als Favorit gespeichert wurde oder sich ein Benchmark-Fall irrtümlich in die Favoritenliste eingeschlichen hat.

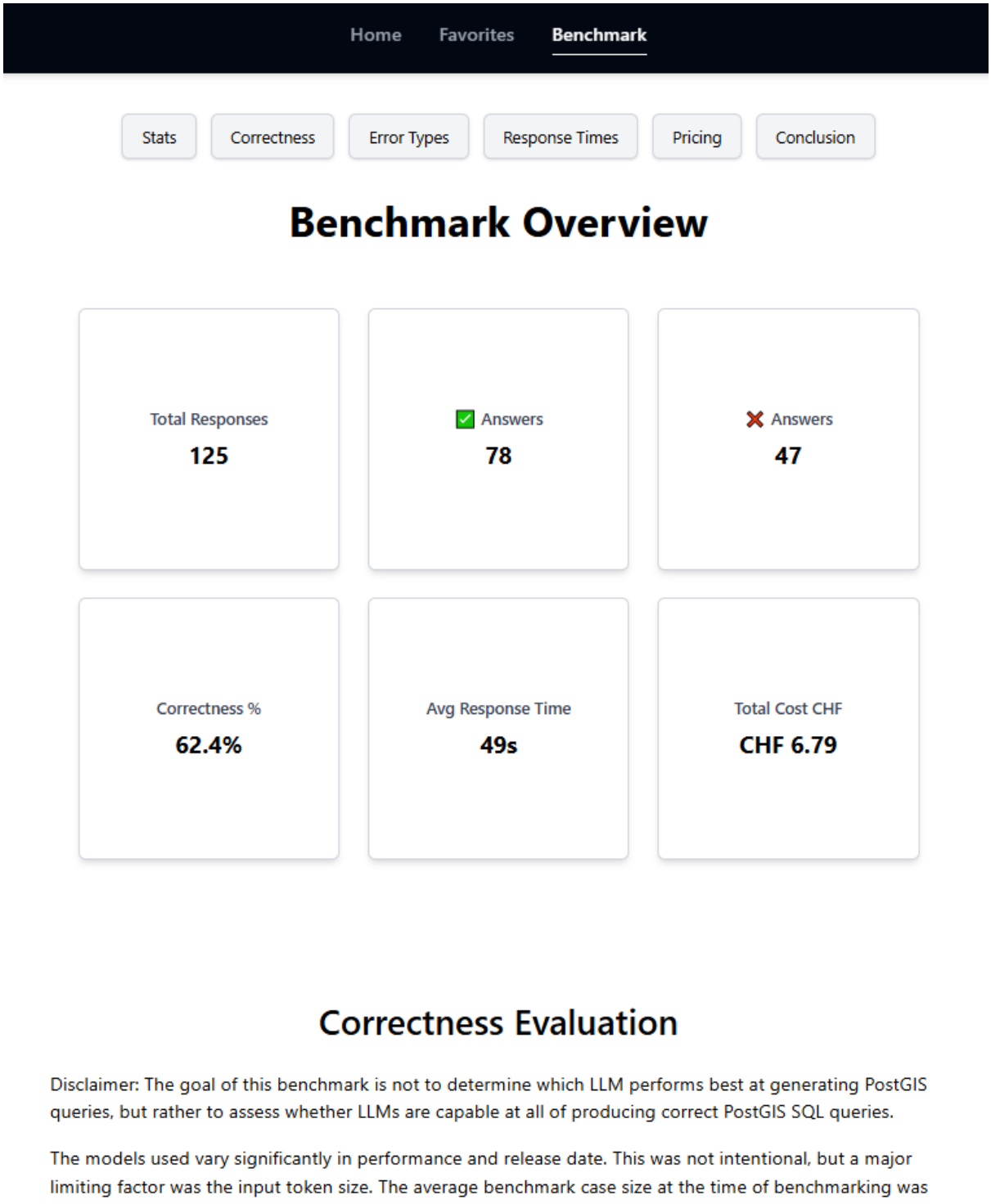

Abbildung 11 Webanwendung Benchmark

Die Benchmark-Seite, dargestellt in Abbildung 11, zeigt sämtliche Ergebnisse des Benchmarks. Dabei werden die Ergebnisse zusammen mit einer Interpretation der Daten dargestellt. Die Interpretation kann Unterschiede zur Diskussion haben, weil die Diskussion meine Ergebnisse mit anderen Ergebnissen vergleicht. Alle Resultate werden anhand von SQL-Abfragen von der Datenbank gezogen direkt gezogen ausser die Pricing Tabelle, welche manuell geführt wurde.

Auf der Ergebnisseite werden zunächst die wichtigsten Informationen zum besten Benchmark angezeigt (mit beiden aktivierten Feedback-Loops und manueller Korrektur). Folgend werden diese Resultate präsentiert:

Tabelle 11 Aufbau der Resultate

Thema	Beschreibung
Stats	Allgemeine Resultate über den besten Benchmark (beide Feedback-Loops aktiv und nach Korrektur von mir)
Correctness	Grosse Tabelle mit allen LLM-Antworten. Jeder Benchmark-Fall wird durchlaufen und mit einem Haken (richtig) oder einem Kreuz (falsch) markiert. Weiter unten wird die Gesamtzahl der Antworten jeweils für jedes LLM angezeigt und deren Korrektheit prozentual dargestellt. Auf jedem Benchmark kann geklickt werden, um weitere Informationen über den Benchmark Fall zu sehen. Wie die Frage des Benchmarks, die Beispiellösung, die Fehler der LLM und falls ein Fehler im besten Benchmark angezeigt werden, sieht man auch die Zuteilung der Fehler. Die gleiche Tabelle existiert für drei Benchmark-Durchläufe mit unterschiedlichen Einstellungen: • Beide Feedback-Loops deaktiviert • Nur UserFeedback-Loop aktiviert • Beide Feedback-Loops aktiviert • Beide Feedback-Loops aktiviert mit Korrektur
Error Types	Eine Übersicht der gemachten Fehler.
Response Times	Antwort Zeit für die LLMs
Pricing	Preise die durch den Benchmark entstanden sind
Conclusion	Kurze Zusammenfassung der Ergebnisse mit einer eigenen Interpretation.

Tabelle 11 zeigt alle Abschnitte der Benchmark-Seite im Überblick. Die Resultate aus den Themen werden in den nächsten Kapiteln behandelt. Die Stats zeigen eine allgemeine Statistik mit den wichtigsten Werten aus dem Benchmark. Die Correctness beinhaltet alle Benchmark Durchläufe und die Korrektheit in Nummern und Prozenten ausgedrückt. Die Fehler werden unter Error Types analysiert. Die Antwortzeiten und Preisübersicht werden im Kapitel

Performanz aufgegriffen und am Ende der Seite besteht eine Zusammenfassung.

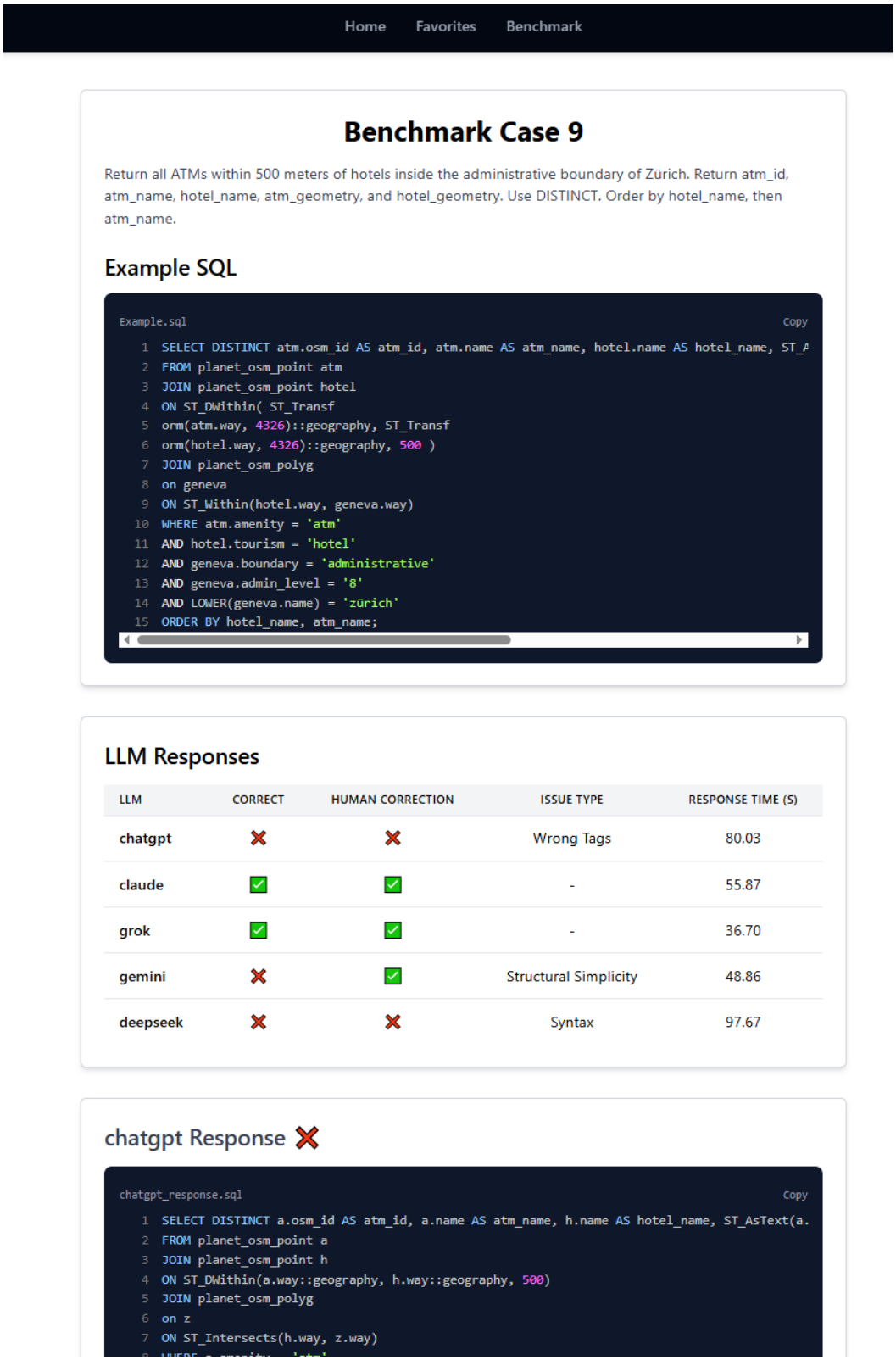

Abbildung 12 Webanwendung Benchmark Fall

Abbildung 12 ist eine Bildschirmaufnahme eines Benchmark Falls. Innerhalb der Correctness der Ergebnisse kann auf jedem einzelnen Benchmark Fall pro Durchlauf geklickt werden. Dies

führt zu einer separaten Detailansicht, die dynamisch mit den relevanten Informationen befüllt wird. Es handelt sich um eine generische Vorlage, die bei jedem Aufruf mit den Informationen zum gewählten Durchlauf sowie dem selektierten Benchmark-Fall befüllt wird. Zuerst steht ein Titel mit der Benchmark Case Nummer. Danach ist die Aufgabe des Benchmark Falls angezeigt mit der Beispiel SQL Abfrage. Oben rechts im Codeblock befindet sich ein Kopiersymbol. Wenn daraufgeklickt wird, wird die SQL-Abfrage in die Zwischenablage kopiert. Danach werden der zugewiesene Fehlertyp und die benötigte Antwortzeit des LLMs angezeigt. Zuerst wird angezeigt, ob es automatisch als korrekt erkannt wurde, danach wie es durch einen Menschen korrigiert wurde. Danach folgt der zugewiesene Fehlertyp sowie die Zeit, die benötigt wurde, um eine Antwort vom LLM zu erhalten. Weiter unten werden die Antworten jedes LLMs jeweils in einem eigenen Codeblock dargestellt. Auch hier befindet sich oben rechts im Codeblock ein Kopierknopf, der beim Anklicken die SQL-Abfrage des jeweiligen LLMs in die Zwischenablage kopiert.

5.2 Benchmark-Ergebnisse

Tabelle 12 Übersicht der Benchmark Resultate

Totale Antworten	Korrekte Antworten	Falsche Antworten	Korrektheit	Durchschnittliche Antwortzeit	Totale Kosten
125	78	47	62.4%	49s	CHF 6.79

Alle Werte in der Live-Applikation werden direkt von der Datenbank gezogen. Im Backend ist eine Service Klasse mit SQL-Abfragen vorhanden, welche alle Daten für die nächsten Kapitel aufbereitet. Tabelle 12 ist eine Übersicht des Durchlaufs mit der besten Benchmark-Konfiguration. Dies ist der Fall mit Syntax-Feedback-Loop und User-Feedback-Loop aktiviert und manuell überprüft. Die Werte sind in der Applikation anhand von Cards dargestellt, die Werte sind in der Tabelle oben übernommen. Jeder Benchmark-Durchlauf beinhaltet 125 Antworten (5 LLMS, 25 Benchmark Cases). Der Durchlauf führte zu 78 richtigen und 47 falschen Antworten, was eine Korrektheit von 62.4% entspricht. Im Durchschnitt betrug die Antwortzeit 49 Sekunden, dazu zählen auch die Wiederholungen im Syntaxfeedbackloop. Für 125 Abfragen über 5 LLMs verteilt mit verschiedenen Modellen betrugen die Kosten 6.79 CHF. Diese Übersicht gibt einen ersten Einblick in die Resultate, folgend wird in die Resultate genauer eingegangen. In den Benchmark Ergebnissen werden die verschiedenen Benchmark-Durchläufe ausgewertet und verglichen.

5.2.1 Korrektheit der generierten SQL-Abfragen

Im Folgenden wird die Korrektheit des Benchmarks ausgewertet. Die Benchmark-Fragen sind nach Schwierigkeit geordnet: Fall 1 ist sehr einfach, Fall 25 sehr schwierig. Dabei ist es weniger wichtig, die verschiedenen LLMs anzuschauen, sondern die Leistung aller LLMs zusammen zu interpretieren. Denn nicht alle Modelle erlauben einen durchschnittlichen Input von 22’400 Tokens. Somit musste bei einigen LLMs auf leistungsschwächere Modellvarianten zurückgegriffen werden, um sich den Umständen der Anwendung anzupassen. Einzelne Extremwerte werden diskutiert und mit Namen des LLMs erläutert, jedoch für die Beantwortung der Forschungsfrage ist der Kern herauszufinden, ob LLMs überhaupt SQL-Abfragen in einer räumlichen Datenbank erstellen können. Der Benchmark wurde in drei Durchläufen mit unterschiedlichen Einstellungen durchgeführt. Die beste Variante wurde anschliessend manuell von mir korrigiert und separat im gleichen Format ausgewertet. Folgende Übersicht gibt es vier verschiedene Einstellungen, die ausgewertet werden:

1. Keine Feedback-Loops
2. Nur User-Feeback-Loop
3. User-Feedback-Loop und Syntax-Feedback-Loop
4. User-Feedback-Loop und Syntax-Feedback-Loop mit Korrektur

Tabelle 13 Benchmark-Durchlauf ohne Feedback-Loops

Benchmark ID	ChatGPT	Claude	Deepseek	Gemini	Grok
1	✗	✓	✗	✓	✓
2	✓	✓	✓	✓	✓
3	✗	✓	✓	✗	✗
4	✗	✗	✗	✗	✗
5	✗	✗	✓	✓	✓
6	✗	✗	✓	✓	✓
7	✗	✗	✗	✗	✗
8	✗	✗	✗	✗	✗
9	✗	✗	✗	✗	✗
10	✗	✗	✓	✓	✓
11	✗	✗	✗	✗	✗
12	✗	✗	✗	✓	✓
13	✗	✗	✗	✗	✗
14	✗	✗	✗	✗	✗
15	✗	✗	✗	✗	✗
16	✗	✗	✗	✗	✗
17	✗	✗	✗	✗	✗
18	✗	✗	✗	✗	✗
19	✗	✗	✗	✗	✗
20	✗	✗	✓	✗	✗
21	✗	✗	✗	✗	✗
22	✗	✗	✗	✗	✗
23	✗	✗	✗	✗	✗
24	✗	✗	✗	✗	✗
25	✗	✓	✓	✗	✓
Total korrekt	1/25	4/25	7/25	6/25	7/25
Korrektheit	4%	16%	28%	24%	28%

Der Benchmark-Durchlauf ohne Feedback-Loops wird in der Tabelle 13 dargestellt. Einfache Fragen konnten vereinzelt richtig beantwortet werden, die mittleren und schwierigen Aufgaben wurden schlecht beantwortet. ChatGPT erzielte eine sehr schwache Leistung mit nur einer einzigen korrekten Antwort, das entspricht einer Korrektheit von 4 %. Claude hat 4 Antworten von 25 richtig beantwortet, dies entspricht einer Korrektheit von 16%. Deepseek beantwortete 7 von 25 Fragen korrekt, also 28 %. Gemini hat 6 Fragen von 25 richtig beantwortet, umgerechnet entspricht das 24% Korrektheit und Grok hat 7 Fragen von 25 richtig beantwortet, umgerechnet ist das eine Korrektheit von 28%. Unter den LLMs kann im optimalen Fall eine Erfolgsquote von 30% der Antworten für Text-to-SQL für PostGIS Aufgaben richtig erreicht werden. Das ist der Stand der aktuellen LLMs ohne die Hilfe von Feedback-Loops.

Tabelle 14 Benchmark-Durchlauf mit User-Feedback-Loop

Benchmark ID	ChatGPT	Claude	Deepseek	Gemini	Grok
1	✓	✓	✓	✓	✓
2	✓	✓	✓	✗	✓
3	✓	✓	✓	✓	✓
4	✗	✗	✗	✗	✗
5	✓	✗	✓	✓	✓
6	✓	✓	✓	✓	✓
7	✓	✓	✓	✗	✓
8	✗	✗	✓	✗	✓
9	✗	✗	✓	✗	✗
10	✓	✗	✓	✓	✓
11	✓	✗	✗	✓	✗
12	✓	✓	✓	✓	✓
13	✗	✗	✓	✓	✓
14	✗	✗	✗	✓	✗
15	✗	✓	✗	✗	✓
16	✗	✗	✗	✗	✗
17	✗	✗	✗	✗	✗
18	✗	✗	✗	✗	✗
19	✗	✗	✗	✗	✗
20	✗	✗	✓	✗	✓
21	✗	✗	✗	✗	✗
22	✗	✗	✗	✗	✗
23	✗	✗	✗	✗	✗
24	✗	✗	✗	✗	✗
25	✗	✓	✓	✗	✓
Total korrekt	9/25	8/25	13/25	9/25	13/25
Korrektheit	36%	32%	52%	36%	52%

Der Benchmark-Durchlauf mit User-Feedback-Loop aktiviert ist in der Tabelle 14 abgebildet. Dieser Durchlauf beinhaltet den User Feedbackloop. Das bedeutet, die LLMs haben für jede Aufgabe alle Favoriten erhalten. Die einfacheren Aufgaben (1–10) wurden von allen LLMs mehrheitlich korrekt beantwortet, mittlere und schwierige Aufgaben hingegen deutlich seltener. ChatGPT hat 9 von 25 Fragen richtig beantwortet, umgerechnet 36% Korrektheit. Claude hat 8 von 25 Fragen richtig beantwortet, umgerechnet 32% Korrektheit. Deepseek hat 13 von 25 Fragen richtig beantwortet, umgerechnet 52% Korrektheit. Gemini hat 9 von 25 Fragen richtig beantwortet, umgerechnet 36% Korrektheit. Grok hat 13 von 25 Fragen richtig beantwortet, umgerechnet 52% Korrektheit.

Tabelle 15 Benchmark-Durchlauf mit User-Feedback-Loop und Syntax-Feedback-Loop

Benchmark ID	ChatGPT	Claude	Deepseek	Gemini	Grok
1	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓
4	✗	✗	✗	✗	✗
5	✓	✗	✓	✓	✓
6	✓	✓	✓	✓	✓
7	✓	✓	✗	✓	✓
8	✓	✓	✗	✓	✓
9	✗	✓	✗	✗	✓
10	✓	✗	✓	✓	✓
11	✓	✓	✗	✓	✗
12	✓	✓	✓	✓	✓
13	✗	✓	✓	✗	✓
14	✗	✗	✗	✗	✗
15	✓	✓	✗	✗	✓
16	✗	✗	✗	✗	✗
17	✗	✗	✗	✗	✗
18	✗	✗	✗	✗	✗
19	✗	✗	✗	✗	✗
20	✗	✗	✓	✗	✓
21	✗	✗	✗	✗	✗
22	✗	✓	✗	✗	✗
23	✓	✗	✗	✗	✗
24	✗	✓	✗	✗	✗
25	✗	✗	✓	✗	✓
Total korrekt	12/25	13/25	10/25	10/25	14/25
Korrektheit	48%	52%	40%	40%	56%

Der Benchmark-Durchlauf mit User-Feedback-Loop und Syntax-Feedback-Loop aktiviert ist in der Tabelle 15 abgebildet. Der dritte Durchlauf beinhaltet den User-Feedback-Loop sowie den Syntax-Feedback-Loop. Die einfachen Aufgaben wurden gut gelöst, jedoch bereiteten die mittleren und schwierigen Aufgaben weiterhin Schwierigkeiten. ChatGPT hat 12 von 25 Fragen richtig beantwortet, umgerechnet 48% Korrektheit. Claude hat 13 von 25 Fragen richtig beantwortet, umgerechnet 52% Korrektheit. Deepseek hat 10 von 25 Fragen richtig beantwortet, umgerechnet 40% Korrektheit. Gemini hat 10 von 25 Fragen richtig beantwortet, umgerechnet 40% Korrektheit. Grok hat 14 von 25 Fragen richtig beantwortet, umgerechnet 56% Korrektheit.

Tabelle 16 Benchmark-Durchlauf mit User-Feedback-Loop und Syntax-Feedback-Loop nach Korrektur

Benchmark ID	ChatGPT	Claude	Deepseek	Gemini	Grok
1	✓	✓	✓	✓	✓
2	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓
4	✗	✗	✗	✗	✗
5	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓
7	✓	✓	✗	✓	✓
8	✓	✓	✗	✓	✓
9	✗	✓	✗	✓	✓
10	✓	✗	✓	✓	✓
11	✓	✓	✗	✓	✗
12	✓	✓	✓	✓	✓
13	✗	✓	✓	✗	✓
14	✗	✓	✓	✗	✓
15	✓	✓	✗	✗	✓
16	✗	✗	✗	✗	✓
17	✓	✓	✓	✓	✓
18	✓	✗	✓	✓	✗
19	✗	✗	✗	✗	✗
20	✗	✗	✓	✓	✓
21	✗	✗	✗	✗	✓
22	✗	✓	✗	✗	✗
23	✓	✓	✗	✗	✓
24	✗	✓	✓	✗	✗
25	✗	✗	✓	✗	✓
Total korrekt	14/25	17/25	14/25	14/25	19/25
Korrektheit	56%	68%	56%	56%	76%

Der Benchmark-Durchlauf mit User-Feedback-Loop und Syntax-Feedback-Loop aktiviert mit manueller Korrektur ist in der Tabelle 16 abgebildet. Von den drei vorherigen Durchgängen wurde der beste Durchgang selektiert. Alle als falsch markierte Aufgaben wurden durch mich ausgewertet, korrigiert und kategorisiert. Die Resultate haben sich stark verbessert in allen Schwierigkeiten durch die menschliche Überprüfung. Dabei haben alle LLMs Antworten als falsch markiert bekommen vom Benchmark System, welche von mir als richtig markiert wurden. Besonders Grok AI konnte schwierige Aufgaben gut lösen, jedoch auf eine komplexe Weise, die nicht mehr dem strikten 1:1-Vergleich des Benchmarks entsprach. ChatGPT hat 14 von 25 Fragen richtig beantwortet, umgerechnet 56% Korrektheit. Claude hat 17 von 25 Fragen

richtig beantwortet, umgerechnet 68% Korrektheit. Deepseek hat 14 von 25 Fragen richtig beantwortet, umgerechnet 56% Korrektheit. Gemini hat 14 von 25 Fragen richtig beantwortet, umgerechnet 56% Korrektheit. Grok hat 19 von 25 Fragen richtig beantwortet, umgerechnet 76% Korrektheit.

5.2.2 Einfluss von Feedback-Loops

Tabelle 17 Vergleich der Benchmark Durchläufe

	Feedback-Loops	**Human Feedback-Loop**	**Syntax-Feedback-Loop**	**Korrektheit**
Keine Feeback-Loops	✗	✗	✗	20%
Nur User-Feeback-Loop	✓	✓	✗	41.60%
Mit Feedback-Loops	✓	✓	✓	47.20%
Mit Manueller Evaluation	✓	✓	✓	62.40%

Alle Benchmark-Durchläufe sind in der Tabelle 17 abgebildet. Auf der X-Achse befinden sich die Feedback-Loops und am Ende die Korrektheit in Prozent ausgedrückt. Auf der Y-Achse befinden sich die Benchmark-Durchläufe. Ohne Feedback-Loops liegt die Korrektheit bei 20% im Schnitt. Ohne Feedback-Loops bedeutet es wird nur ein Prompt dem LLM geschickt und keine Beispiele werden ihm geschickt und der LLM wird auch nicht informiert, wenn ein Syntaxfehler besteht. Dies entspricht der reinen, unbeeinflussten Leistung des LLMs im Benchmark. Mit aktivierter User-Feedback-Loop erhält der LLM-Zugriff auf die gespeicherten Favoriten. Dies führte zu einer absoluten Verbesserung der Leistung von 21.6%. Die Leistung verdoppelte sich nahezu, sobald dem LLM ähnliche Beispielabfragen wie im Benchmark mitgegeben wurden. Durch Aktivierung des Syntax-Feedback-Loops lag die durchschnittliche Korrektheit bei 47,2 % – eine weitere absolute Verbesserung um 5,6 %. Total bedeutet das, dass durch die Applikation und den programmierten Feedbackloops eine absolute Verbesserung von total 27.2% gegeben hat. Wobei die 27.2% auch höher hätte sein können, wenn bessere und mehr korrekte Favoriten mitgegeben werden. Die Benchmark-Leistung konnte sich somit mehr

als verdoppeln mit der Hilfe des Syntax-Feedback-Loops, kombiniert mit dem Human Feedback-Loop.

5.2.3 Manuelle Korrekturen

Manuell wurden alle falsch markierten Benchmark-Resultate aus dem Syntax-Feedback-Loop und User-Feedback-Loop Durchlauf nochmals bewertet. Der letzte Benchmark-Durchlauf aus der Tabelle 16 zeigt die Resultate mit manueller Korrektur.

Tabelle 18 Vorgenommenen Korrekturen

Korrektur	Korrekt geblieben	Korrigiert	Falsch geblieben
Anzahl	59	19	47

In Tabelle 18 werden die Anzahl korrigierter Benchmark-Resultate für den Durchlauf Syntax-Feedback-Loop und User-Feedback-Loop aktiviert dargestellt. In Zahlen ausgedrückt gab es 59 Abfragen, die direkt richtig beantwortet wurden und somit keine Korrektur durchgegangen sind. Die als korrekt markierten Aufgaben werden nicht kontrolliert, weil der Vergleich von der Antwort des LLM und der Musterlösung sehr streng gemessen wird. Eine Aufgabe wird nur als korrekt markiert, wenn die Antwort der SQL-Abfrage gleich ist mit der Antwort der Muster SQL-Abfrage. Die Tabellen-Header werden beim Vergleich nicht miteinbezogen. 19 Fragen sind durch mich manuell verbessert worden und als korrekt eingestuft worden. Bei 47 Fällen, war auch nach der Korrektur weiterhin als falsch bewertet. Die manuelle Korrektur durch mich hat die durchschnittliche Korrektheit von 47.2% auf 62.4% erhöht. Das heisst in 15.2% der Fällen gab es einen false negative (falsch bewertet, aber korrekter SQL). Die false negative Fehler sind meistens Simplizitätsfehler, die vom Beispiel innerhalb des Benchmarks stammen. Sprich die Lösung des LLM überschreitet dies was erwartet wird und wird darum als falsch bewertet. Die Bestleistung im Schnitt durch den Benchmark beträgt somit 62.4%.

5.3 Fehleranalyse

Nur der Benchmark-Durchlauf mit beiden Feedback-Loops aktiviert wurde nachkorrigiert. Die Fehler wurden alle kategorisiert. Es gibt Fehler, aber es gibt auch Fehler, die eigentlich keine Fehler entsprechen, sprich ein false negative. Die Fehler sind in zwei Kategorien unterteilt, die falschen Antworten, true negatives und die durch den Menschen korrigierten Antworten die als richtig erkennt werden, sprich false negatives. Jede Antwort hat dabei genau einen Fehler-Typen zugewiesen. Wenn zwei Fehler auftauchen, wird der Fehler zugewiesen, der gravierender ist. Dies macht die Übersicht der Fehler einfacher und verschafft ein sauberes Bild der Fehler.

5.3.1 Fehlertypen: False negatives

Tabelle 19 False negatives

Fehlertyp	**Structural Simplicity**	**Performance Issue**	**Encoding Artifact**
Anzahl	15	3	1

In der Tabelle 19 dargestellt gibt es drei verschiedene Fehlertypen, die für korrigierte Aufgaben als richtig markiert wurden, also false negatives. Der häufigste Fehlertyp ist Structural Simplicity, der insgesamt 15-mal auftrat. Structural Simplicity bedeutet, dass die Antwort korrekt war und zusätzlich Details oder weitere Tags hinzugefügt, als beim Benchmark Beispiel hinterlegt wurde. Dreimal kam ein Performance Issue auf. Unter Performance Issue ist die SQL-Abfrage korrekt, jedoch gab es ein Timeout auf der Datenbank. Bei Supabase, wo die Datenbank sich befindet, gibt es automatisch einen Timeout nach 60 Sekunden. Dies wird nicht als echter Fehler gewertet, weil ein solcher Performance-Engpass in einer praxisnahen Umgebung durch leistungsfähigere Hardware vermieden werden könnte. Einmal trat einen Encoding Artifact auf, die Lösung des LLMs wurde falsch formatiert der Datenbank überwiesen. Der Fehler ist auf das LLM selbst zurückzuführen. Die SQL-Antwort war jedoch inhaltlich korrekt.

5.3.2 Fehlertypen: True negatives

Tabelle 20 True negatives

Fehlertyp	Spatial Context Misuse	Tag Loss	Syntax	Wrong Tags
Anzahl	22	10	8	7

In der Tabelle 20 dargestellt gibt es vier verschiedene Fehlertypen, die für korrigierte Aufgaben als falsch markiert wurden, also true negatives. Fehler, die auch nach manueller Korrektur weiterhin als falsch gelten – sogenannte True Negatives – wurden in vier Kategorien unterteilt. Spatial Context Misuse ist 22-mal vorgekommen und damit ist eine falsche geographische Interpretation gemeint, sprich ein Problem mit der Verwendung von PostGIS Erweiterung im SQL. Tag Loss ist zehnmal vorgekommen. Unter Tag Loss fehlen wichtige Filterkonditionen, um die richtigen Daten zu erhalten. Syntaxfehler sind achtmal vorgekommen, unter Syntax wird ein Problem mit der SQL-Abfrage verstanden, welche es nicht ermöglicht die SQL-Abfrage gegen die Datenbank laufen zu lassen. Zuletzt gibt es Wrong Tags, was siebenmal vorkam, wurden die falschen Tags verwendet für das Filtrieren von Aufgaben.

5.3.3 Beispiele für typische Fehler

Im Folgenden werden drei typische Fehlertypen vorgestellt. Der eine wird ein false negative Fall sein, den wir danach als korrekt bezeichnen werden. Die anderen beiden werden true negative Fälle sein. Zuerst wird der Benchmark Fall genannt, danach wird die Aufgabe des Benchmark-Falls aufgeschrieben und danach wird in einer Tabelle die SQL-Antwort des LLM mit der SQL-Antwort der Beispiellösung verglichen.

False negative, Deepseek, Benchmark Fall: 14. Find all roads (NOT NULL) that intersect with rivers (NOT NULL) inside the administrative boundary of Lucerne. Use geometry from planet_osm_line. Return osm_id, name. Use DISTINCT. Order by name.

Tabelle 21 Beispiel 1: False negative

LLM-Antwort	**Beispiellösung**
SELECT DISTINCT r.osm_id, r.name *FROM planet_osm_line r* *JOIN planet_osm_line w ON* *ST_Intersects(r.way, w.way)* *JOIN planet_osm_polygon a ON* *ST_Within(r.way, a.way)* *WHERE r.highway IS NOT NULL* ***AND w.waterway IS NOT NULL*** *AND a.admin_level = '8'* *AND (LOWER(a.name) = 'luzern' OR* *LOWER(a.name) = 'lucerne')* *ORDER BY r.name;*	*SELECT DISTINCT r1.osm_id, r1.name* *FROM planet_osm_line r1* *JOIN planet_osm_line r2 ON* *ST_Intersects(r1.way, r2.way)* *JOIN planet_osm_polygon l ON* *ST_Within(r1.way, l.way)* *WHERE r1.highway IS NOT NULL* ***AND r2.waterway = 'river'*** *AND l.boundary = 'administrative'* *AND l.admin_level = '8'* *AND LOWER(l.name) = 'luzern'* *ORDER BY r1.name;*

In Tabelle 21 hat das LLM die Bedingung gesetzt, dass der Waterway-Wert nicht null sein darf, während sich die Musterlösung auf den Wert „river“ beschränkt. Dadurch enthält die LLM-Antwort mehr Informationen als die Referenzlösung und wurde vom Benchmark automatisch als falsch eingestuft. Weil jedoch die richtige Abfrage gestellt wurde und relevante Informationen korrekt berücksichtigt wurden – insbesondere vor dem Hintergrund unvollständiger OSM-Daten –, wurde dieser Fall nachträglich als korrekt bewertet. Er erhielt den Tag „Structural Simplicity“, und das Feld „human_correction“ wurde auf true gesetzt.

True negative, Claude, Benchmark Fall: 10. Find all distinct kindergartens within 500 meters of primary schools inside the administrative boundary of Bern. Use geometries from planet_osm_point and proximity via ST_DWithin in SRID 4326 with ::geography. Return kindergarten_id, kindergarten_name. Use DISTINCT. Order by kindergarten_name.

Tabelle 22 Beispiel 2: True negative

LLM-Antwort	**Beispiellösung**
SELECT DISTINCT b.osm_id, b.name, *ST_AsText(ST_Transform(b.way, 4326)) AS geometry* *FROM planet_osm_polygon b* *JOIN planet_osm_polygon city ON ST_Within(b.way, city.way)* *WHERE b.building = 'public'* *AND city.boundary = 'administrative'* *AND city.admin_level = '8'* *AND LOWER(city.name) = 'bern'* *ORDER BY b.name;*	*SELECT DISTINCT* *k.osm_id AS kindergarten_id,* *k.name AS kindergarten_name* *FROM planet_osm_point k* *JOIN planet_osm_point s* *ON ST_DWithin(* *ST_Transform(k.way, 4326)::geography,* *ST_Transform(s.way, 4326)::geography,* *500* *)* *JOIN planet_osm_polygon a* *ON ST_Within(k.way, a.way)* *WHERE* *k.amenity = 'kindergarten'* *AND s.amenity = 'school'* *AND a.boundary = 'administrative'* *AND a.admin_level = '8'* *AND LOWER(a.name) = 'bern'* *ORDER BY kindergarten_name;*

In der Tabelle 22 traten mehrere Fehler auf. Zum einen wurde das Polygon anstatt eine Punkte-Tabelle ausgewählt, um die Daten zu bekommen. Die Filtrierung war auf public limitiert, jedoch eigentlich wurde Kindergarten erwarten. Zudem fehlt die Prüfung der Nähe einer Primarschule. Hier gab es viele Fehler und wurde mit dem Tag Spatial Context Misuse versehen.

True negative, Grok, Benchmark Fall: 18 Find all building polygons (NOT NULL) that share a border with any park. Return osm_id and geometry (ST_AsText). Order by osm_id.

Tabelle 23 Beispiel 3: True negative

LLM-Antwort	**Beispiellösung**
SELECT DISTINCT b.osm_id, *ST_AsText(b.way) AS building_geom* *FROM planet_osm_polygon b* *JOIN planet_osm_polygon p ON* *ST_Touches(b.way, p.way)* *JOIN planet_osm_polygon z ON* *ST_Within(b.way, z.way)* *WHERE b.building IS NOT NULL* *AND p.leisure = 'park'* *AND z.boundary = 'administrative'* *AND z.admin_level = '8'* *AND LOWER(z.name) = 'zürich'* *ORDER BY b.osm_id;*	*WITH buildings AS (* *SELECT osm_id, way* *FROM planet_osm_polygon* *WHERE building IS NOT NULL* *),* *parks AS (* *SELECT way* *FROM planet_osm_polygon* *WHERE leisure = 'park'* *)* *SELECT b.** *FROM buildings b* *JOIN parks p ON ST_Touches(b.way, p.way)* *ORDER BY b.osm_id;*

Die Tabelle 23 zeigt die LLM-Antwort und der Request inhaltlich deutlich verfehlt. Die Zürich Filtrierung war nicht in der originalen Fragestellung erwähnt. Zudem wurde ein separater Join eingebaut, der für die Beantwortung der Frage nicht notwendig war. Die Antwort wurde dementsprechend als falsch bewertet und mit dem Tag Spatial Context Misuse versehen.

5.4 Performanz-Messungen

Der Benchmark erfasst neben der Korrektheit auch die Performanz. So wurden die Antwortzeiten für jede Abfrage auch in den Resultaten abgespeichert und manuell wurden die Kosten für einen Benchmark-Durchlauf dokumentiert. Anhand dieser Daten können weitere Rückschlüsse entnommen werden, ob ein Einsatz von Text-to-PostGIS-Applikationen in der Praxis realistisch und wirtschaftlich sinnvoll ist.

5.4.1 Antwortzeiten der LLMs

Tabelle 24 Antwortzeiten

LLM	ChatGPT	Claude	Deepseek	Gemini	Grok
Durchschnittliche Zeit in Sekunden	33.63	57.66	67.05	47.78	40.84

Die Tabelle 24 zeigt, dass die Antwortzeit der LLMs sich stark variieren. Im Schnitt liegen die LLMs zwischen 34 und 67 Sekunden. ChatGPT ist im Durchschnitt das schnellste Modell, während Deepseek im Durchschnitt das langsamste Modell ist. In der Mitte befinden sich Grok mit einer durchschnittlichen Zeit von 41 Sekunden, Gemini mit einer durchschnittlichen Zeit von 48 Sekunden und Claude mit einer durchschnittlichen Zeit von 58 Sekunden. Der gemessene Benchmark hat in diesem Fall 102.9 Minuten gebraucht, um durch alle Benchmark Cases für jeden LLM zu gehen.

Tabelle 25 Extremwerte bei Antwortzeiten

Typ	LLM	Antwortzeit (s)
Schnellste Antwort	Gemini	4.14
Langsamste Antwort	Deepseek	221.52

Unter den einzelnen Benchmark-Ergebnissen zeigen sich zwei Extremwerte. Die Tabelle 25 hat die schnellste sowie die langsamste Antwort der LLMs festgehalten. Die schnellste Antwort war von Gemini und betrug 4.14 Sekunden. Die langsamste Antwortzeit war von Deepseek und betrug 221.52 Sekunden, umgerechnet mehr als 3.5 Minuten.

Tabelle 26 Durchschnittliche Antwortzeit basierend auf Korrektheit

Antwort Typ	Falsche Antworten	Richtige Antworten
Durchschnittliche Antwortzeit (s)	53.92	44.33

Wie aus der Tabelle 26 zu entnehmen ist, kamen falsche Antworten im Durchschnitt langsamer zurück als korrekte Antworten. Falsche Antworten benötigten im Durchschnitt gerundet 54 Sekunden. Korrekte Antworten kamen im Durchschnitt nach abgerundet 44 Sekunden zurück. Das ergibt ein Unterschied von 9.59 Sekunden.

Tabelle 27 Faktoren für Antwortzeiten

Thema	Beschreibung
Geographisch	Wie weit ist der Standort vom Server von Zürich entfernt. Die Supabase-Datenbank ist in Zürich in einer EC2-Instanz gehosted. Gewisse LLMs wie zum Beispiel Deepseek haben keine Instanz in Europa, somit muss jeder Request nach China und zurück. Der Benchmark wurde immer lokal von Uitikon Waldegg durchgeführt.
Syntax-Feedback-Loop	Die Anzahl Wiederholungen, die durch den Feedback-Mechanismus ausgelöst wurden (bis zu fünfmal).
Antwortzeit Datenbank	Wie lange die Datenbank benötigt, um eine SQL-Abfrage auszuführen. Mit einem stärkeren CPU kann die Ausführung der SQL-Abfrage schneller gehen.
Länge der Antwort	Eine längere Antwort vom LLM benötigt mehr Zeit generiert zu werden.
LLM	Durch einen Fehler des LLMs wurde keine Antwort retourniert, ein langsames Modell wurde selektiert oder die Antwort vom LLM kann verspätet kommen.

In dieser Arbeit wurde nicht gemessen, welche Faktoren wie gross für einen Zeitunterschied ausmachen. Jedoch ist es wichtig diese fünf Faktoren aus der Tabelle 27 zu berücksichtigen, wenn über die Antwortzeit gesprochen wird. Der geografische Standort der Benutzer und der Server kann die Latenz geringfügig beeinflussen. Der Syntax-Feedback-Loop kann eine Anfrage bis zu fünfmal durchführen, was die Antwortdauer einer Anfrage verlängert. Die Antwortzeiten der Datenbank hängen unter anderem von der Stärke der CPU ab. Auch die Länge der generierten Antwort hat einen leichten Einfluss auf die Gesamtdauer. Wo die Server eines LLM befinden, hat durchaus eine grosse Auswirkung, vor allem weil Antworten von Fremdsystemen manchmal verspätet kommen.

5.4.2 Kostenübersicht

Tabelle 28 Kostenübersicht

LLM	ChatGPT	Claude	Deepseek	Gemini	Grok
Kredit Start	$7.65	$8.57	¥19.94	0	$4.90
Kredit Ende	$7.44	$6.79	¥19.27	CHF 1.73	$0.72
Preis Total	$0.21	$1.78	¥0.67	CHF 1.73	$4.18
Preis Total in CHF	CHF 0.17	CHF 1.44	CHF 0.07	CHF 1.73	CHF 3.38
Preis pro Fall in CHF	CHF 0.0068	CHF 0.0576	CHF 0.0028	CHF 0.0692	CHF 0.1352

Die Kostentabelle 28 wurde manuell mitgeführt. Die Kosten bei den jeweiligen LLMs variieren stark. Dies ist aber auch verständlich, weil nicht immer das meist performante Modell, welches gerade von den LLMs verfügbar ist, verwendet werden konnte für diese Arbeit. Wichtig war, dass die LLMs die Input Datenmengen verarbeiten konnten. Auf Deepseek konnte verfolgt werden, wie viele Token für einen Benchmark-Durchlauf benötigt werden. Gerundet sind es 560'000 Tokens oder im Schnitt 22'400 Token für jeden Fall. Die Modelle müssen in der Lage sein 30'000 Tokens als Input zu erhalten, ansonsten wird lediglich eine Fehlermeldung zurückgegeben. Gewisse Modelle, wie zum Beispiel Gemini, benötigen keine Vorauszahlung, sondern werden nach dem Pay-as-you-go-Modell abgerechnet. Die anderen LLMs erlauben den Kauf von Guthabenpakete, die nach Subsequenten Abfragen verwendet werden, um Antworten zu generieren. Anhand des Guthabens am Anfang verglichen mit dem Guthaben am Ende wird ermittelt, wie viel ein Benchmark-Durchlauf gekostet hat. Alle Kosten sind in Schweizer Franken umgerechnet worden, der Stand des Wechselkurses vom 22. April 2025 wurde für die Berechnung verwendet. Am günstigsten war Deepseek mit Kosten pro Benchmark Fall auf 0.0028 CHF, oder total 0.07 CHF. Am teuersten war Grok mit einem durchschnittlichen Preis pro Benchmark-Fall von 0.1352 CHF. Total hat Grok 3.38 CHF gekostet. Die anderen LLMs befinden sich in der Mitte. ChatGPT hat durchschnittliche Kosten pro Benchmark Fall von 0.0068 CHF oder total 0.17 CHF gekostet. Claude hat durchschnittliche Kosten pro Benchmark Fall von 0.0576 CHF oder total 1.44 CHF und Gemini hat durchschnittliche Kosten pro Benchmark Fall von 0.0692 CHF oder total 1.73 CHF. Ein Benchmark-Durchlauf mit allen 5 LLMs zusammengerechnet hat total 6.79 CHF gekostet. Dabei werden 125 Antworten pro Benchmark-Durchlauf generiert und die Antworten in einer Datenbank festgehalten.

6 Diskussion

In diesem Kapitel werden die Ergebnisse der Arbeit diskutiert und verglichen mit Ergebnissen von anderen relevanten Arbeiten. Die Kapitelstruktur orientiert sich am Aufbau des Ergebnisteils, jedoch ohne das Kapitel Funktionale Ergebnisse der Webanwendung. Die Webapplikation ist unter text-to-postgis.com zugänglich und trägt wenig zu den eigentlichen wissenschaftlichen Erkenntnissen der Arbeit wesentlich dabei. Eine kürzere Zusammenfassung mit den wichtigsten Punkten ist auf der Webseite unter Benchmark zu finden, wo die Resultate präsentiert und diskutiert werden.

6.1 Benchmark-Ergebnisse

Die Leistungsfähigkeit verschiedener LLMs bei der Generierung von SQL-Abfragen wird diskutiert und mit anderen relevanten Arbeiten aus dem Text-to-SQL-Arbeiten verglichen. Es wird weniger in die individuelle Leistung der LLMs eingegangen, sondern wie gut die LLMs die SQL-Abfragen beantworten. Aus den Ergebnissen wurden 125 natürlichsprachige Anfragen über mehrere LLMs verteilt und mithilfe von definierten Kriterien auf Korrektheit überprüft und schlussendlich von mir manuell überprüft. Die Diskussion wird in folgende Unterkapitel hinsichtlich Korrektheit, Verbesserung durch Feedback-Loops und manuellen Korrekturen diskutiert.

6.1.1 Korrektheit der generierten SQL-Abfragen

Frühere Arbeiten haben gezeigt, dass LLMs bei der Generierung von SQL-Abfragen nur moderate Genauigkeit erreichen. GPT-4 erhielt vom BIRD-Testset eine Ausführungsgenauigkeit von 54.89%, während Claude-2 und ChatGPT 3.5 lediglich 49.02% und 39.3% erreichten (Li et al., 2023, S. 7). Anhand von strategischen Prompts und Werkzeugen wie CHASE-SQL wurde in späteren Studien höhere Werte von bis zu 73% erzielt (Pourreza et al., 2024, S. 11). Die Ausführungsgenauigkeiten der LLMs konnten somit drastisch gesteigert werden. Dennoch bleibt eine zentrale Herausforderung bestehen, wie zum Beispiel LLMs haben bei zunehmender Komplexität der Text-to-SQL Aufgaben einen grösseren Leistungsabfall. Dies wird nach der Auswertung anhand der Kategorisierung von den Aufgaben in einfach, mittleren und schwierigen Abfragen im BIRD-Benchmark klar ersichtlich (Li et al., 2023, S. 21).

Ein Vergleich mit meinen Ergebnissen bestätigt diese Tendenz. Einfache Aufgaben konnten gut gelöst werden, schwierige Aufgaben hingegen weniger. Auch Spezialaufgaben, für die sich in

den Favoriten kaum passende Beispielabfragen fanden, wurden nur selten korrekt gelöst. Kein Modell konnte vom Anfangszustand mehr als 30% Korrektheit erzielen in meinem Benchmark, nur anhand von Feedback-Loops und Korrektur durch mich konnte die Leistung bis auf 76% Korrektheit wachsen. Meine Werte übertreffen die Werte vom BIRD-Benchmark, jedoch das CHASE-SQL Beispiel deutet hin, dass eine fein abgestimmte Umgebung mit gut geschriebenen Favoriten oder allgemein SQL-Strukturen, die Leistung erheblich steigern kann (Pourreza et al., 2024, S. 11). Die Ergebnisse zeigen, dass aktuelle LLMs grundsätzlich in der Lage sind, korrekte SQL-Abfragen für räumliche Datenbanken zu generieren. Dabei gilt: Je gezielter ein LLM angeleitet wird und je besser der Input ist, desto höher fällt die Qualität der generierten Abfragen aus. LLMs werden auch in Zukunft sehr abhängig von der konkreten Struktur und Beispielen der erwarteten SQL-Abfragen sein.

6.1.2 Einfluss von Feedback-Loops

Die Einbindung von Feedback-Loops hatte einen signifikanten positiven Einfluss auf die Korrektheit der Abfragen. Dies hat zu einer absoluten Verbesserungsrate von 21.6% geführt. Ähnliche Ergebnisse zeigen Resultate in Verbundenheit mit dem Spider Benchmark, welche einen Leistungsverbesserung zwischen 14.5% bis 39.7% entnommen haben, durch Large Language Modell Enhanced Embedding im Vergleich zum Basis Modell (Zhan et al., 2025, S. 6). Dies suggeriert, dass der User-Feedback-Loop vergleichbare Effektivität hat, wie datengetriebene Optimierung durch Embeddings. Die Wirksamkeit des User-Feedback-Loops lässt sich weiter steigern, je umfangreicher und qualitativ hochwertiger die Favoritenliste ist. Insbesondere im produktiven Einsatz, etwa in Assistenzsystemen, wird der Nutzen von Feedback-Loops besonders deutlich. Denn die Applikation passt sich kontinuierlich an das Benutzerverhalten und Rückmeldungen an. Ein gesondertes Modelltraining ist nicht zwingend erforderlich – gute Beispiele allein reichen aus, um die Leistung zu steigern (Zhai et al., 2025, S. 8). In unserem Fall ist Feedback-Loop besonders skalierbar, der keine Anpassung an den LLM-Modellen benötigt. Die Auswahl und Organisation von Beispielen allein kann ein signifikante Leistungssteigerungen ermöglichen, ohne dass Fine-Tuning an einem Modell erforderlich ist (Gao et al., 2023, S. 11). Dies belegt auch eine weitere Studie, welche anhand von Off-Policy Training, sprich Training mit Paare von korrekten und falschen Beispielen und Ausführungsfeedback die Leistung um 10.4% gegenüber dem BIRD Benchmark steigern konnte (Zhai et al., 2025, S. 8). Zudem konnte eine kleine Leistungssteigerung anhand des Syntax-Feedback-Loops von 5.6% festgestellt werden. Wie meine Resultate und die Resultate von anderen Studien zeigen, kann das Mitschicken von Beispielen einen positiven Effekt haben

auf die Erzeugung von SQL-Abfragen. Grundlegend für eine erfolgreiche Umsetzung ist, dass die Feedback-Loops hoch qualitative Daten beinhalten. Das können Beispiele sein, wie korrekt umgesetzt wird, wie falsch umgesetzt wird oder wie die Struktur der Datenbank aussieht. Studien zeigen auch, dass eine längere und umfangreichere Feedback-Loop zu abnehmenden Effektivität führt (Zhai et al., 2025, S. 8). LLMs profitieren deutlich, wenn sie mit einer Datenbank arbeiten, mit der sie bereits vertraut sind, wie in dieser Arbeit mit OpenStreetMap-Daten. Je mehr Vorwissen die LLMs haben, desto besser und kreativer werden SQL-Abfragen generiert.

6.1.3 Manuelle Korrekturen

Ein wesentlicher Bestandteil der Ergebnisanalyse war das Erkennen sogenannter false negatives, also Aufgaben, die vom System fälschlicherweise als falsch bewertet wurden. Die Benchmark-Bewertung von mir war zwingend, weil der automatisierte Vergleich zu restriktiv ist. Der restriktive Vergleich ermöglicht mir dafür alle vom System her anerkannte Aufgaben welche korrekt sind als korrekt anzuschauen, ohne tief nachzuforschen. Die manuelle Korrektur des Benchmarks ist das zentrale Element in der Interpretation der Modelleistung. Von 125 Benchmark-Fälle wurden 78 (59 korrekt und 19 manuell korrigiert) als richtig bewertet, was einer Korrektheit von 62.4% entspricht. In Prozent ausgedrückt wurden 15.2% der totalen Fragen von falsch auf richtig markiert. Nicht alle Benchmarks sind komplett zugänglich für die Öffentlichkeit und das Thema von false negatives wird dementsprechend wenig bis gar nicht in anderen Arbeiten besprochen (Ascoli et al., 2025, S. 2). Der Spider-Benchmark wurde auf false negatives getestet mit verschiedenen Modellen die folgenden Zahlen sind über die ganze Anzahl von Fragen, nicht nur falsch beantwortete Fragen. Der Spider-Benchmark hat eine Rate von 3.2% bis 17.5% welche als false negatives bewertet wurden (Ascoli et al., 2025, S. 7). Der BIRD-Benchmark wurde mit den gleichen Modellen auch getestet und die Resultate von false negatives liegen zwischen 20.7% und 28.9% (Ascoli et al., 2025, S. 7). Diese Zahlen verdeutlichen, dass etwa jede fünfte Benchmark-Frage fälschlich als falsch eingestuft wird, obwohl ein typischer Benutzer die Antwort vermutlich als korrekt ansehen würde.

6.2 Fehleranalyse

Neben der Korrektheit ist es ebenso entscheidend, die Fehler der LLMs zu identifizieren und zu analysieren. Folgend werden die Fehlertypen systematisch analysiert, um besser einzuschätzen, welche die Herausforderungen sind beim Einsatz von LLMs im Text-to-SQL Kontext. Der Fokus wird auf false negatives gelegt, die auf Limitierungen des Benchmarks dieser Arbeit und anderen Benchmarks zurückweisen. Zudem wird der falsche Umgang mit Datenbankfunktionen und Datenbankverständnis analysiert und mit anderen Studien verglichen.

6.2.1 Fehlertypen

Das Kategorisieren der Fehler in false negatives und true negatives zeigt, dass es oft in meinem Benchmark zu Fehlern gekommen ist, welche eigentlich gar keine Fehler waren. Dies war aufgrund des strengen Vergleichs der Resultate zurückzuführen. Vorteil bei dieser Implementierung ist, dass die vom System korrekt markierten Antworten überreinstimmen mit der Antwort der Muster-SQL Abfrage. Besonders oft ist Structural Simplicity aufgetaucht, dies ist, wenn ein LLM die SQL-Abfrage nicht nur korrekt beantwortet, sondern sie auch erweitert und vom System aufgrund der Erweiterung als falsch markiert wurde. Je kreativer das LLM ist im Erstellen von SQL-Abfragen, desto eher besteht die Wahrscheinlichkeit, dass die generierte SQL-Abfrage weitere Punkte beinhaltet, die in der Beispiel SQL-Abfrage nicht beinhaltet sind. Dies weist auf eine technische Limitierung des gemachten Benchmarks hin. Andere Studien legten ihren Fokus stärker auf die Analyse der true negatives. Ein Grossteil der true negatives sind Schema-Linking Fehler mit einer 41,6 % Wahrscheinlichkeit. Zudem Missverständnisse bezüglich des Datenbankinhalts 40.8% sind ein grosser Faktor (Li et al., 2023, S. 10). In dieser Arbeit trat am häufigsten der Fehler Spatial Context Misuse auf, was auf eine fehlerhafte Nutzung der räumlichen Datenbankerweiterung PostGIS hinweist. Tag Loss und Wrong Tags lassen sich hingegen zu Fehlern im Datenbank Inhalt widerspiegeln. In absoluten Zahlen ausgedrückt, entspricht das 17 von 47 falsch beantworten Fragen, oder 36.2% ähnlich, was den Ergebnissen von Li entspricht. Ein weiteres Problem ist Overfitting von Prompt-Formate, wobei das Einbinden des User-Feedback-Loops mit den Favoriten effektiv zum Verhängnis wird, weil weitere Beispiele von verschiedener Qualität die Antworten des LLMs auch verschlechtern können (Gao et al., 2023, S. 11).

6.3 Performanz-Messungen

Neben der Korrektheit der Benchmarks ist ein weiterer Punkt die Performanz. Aspekte wie Antwortzeiten und Betriebskosten der LLMs sind entscheidend dafür, ob ein System für den produktiven Einsatz geeignet ist. Besonders in realen Anwendungen können, wo viele Anfragen zeitkritisch und günstig verarbeitet werden müssen, können diese Faktoren über die Skalierbarkeit und praktischen Nutzen entscheidend sein. In diesem Kapitel werden die LLMs hinsichtlich der Antwortzeiten untersucht, sowie die Kosten pro Anfrage einer einzelnen Anfrage im OpenStreetMap-Kontext untersucht.

6.3.1 Antwortzeiten der LLMs

Antwortzeiten haben keinen Einfluss auf die Korrektheit der Antworten der LLMs. Im Fall dieser Arbeit können LLMs in Syntax-Feedback-Loops hineinfallen, dann wird die Antwortzeit sich verlängern. Jedoch je schwieriger die Aufgabe wird, desto weniger gut wurden die Fragen beantwortet. Das Argument kann auch gemacht werden, dass je schwieriger die Aufgabe war, desto länger war die Antwortzeit. Ein Blick auf die durchschnittlichen Antwortzeiten in Abhängigkeit vom Schwierigkeitsgrad der Aufgaben bestätigt diesen Zusammenhang: Komplexere Anfragen führten tendenziell zu längeren Antwortzeiten (Yang et al., 2024, S. 20). Ein weiterer Punkt spezifisch auf diese Arbeit, die Korrektheit der Antwort hat die Antwortzeit beeinflusst, weil es in einen Syntax-Feedback-Loop manchmal gegangen ist. Die Schwierigkeit ist ein Teilfaktor, der zusammen mit dem Syntax-Feedback-Loop sich multipliziert. Der spürbarste Faktor dieser Arbeit war die Antwortzeit der Datenbank. In mehreren Fällen überschritt die Datenbankantwort die 60-Sekunden-Grenze, woraufhin eine Fehlermeldung ausgegeben wurde. Konsequent ist eine Fehlermeldung angezeigt worden. Insbesondere bei komplexeren Anfragen stösst PostGIS an seine Leistungsgrenzen. Die Zeitkomplexität erhöht sich exponentiell mit einem steigenden Schwierigkeitsgrad (Agarwal & Rajan, 2016, S. 675). Der Syntax-Feedback-Loop hat auch dazu getragen, dass die Antwortzeiten sich drastisch verlängert haben. Die langsamste Antwort von Deepseek welche 3.5 Minuten betrug und den Syntax-Feedback-Loop aktiviert hatte, war mehrmals durch den Loop gegangen. Geographisch kann auch bei Deepseek erkannt werden, dass die Server sich in China befinden und daher die durchschnittliche Antwortzeit sich über einer Minute befand. Die Länge der Antwort wird in anderen Applikationen wesentlich essenzieller sein, weil der LLM die Instruktion erhalten hat, nur mit dem SQL zu antworten, sollte dies keinen grossen Unterschied in der Antwortzeit widerspiegeln. Die Varianz des LLM, wie schnell gerade eine Antwort zurückgeschickt werden kann, ist schwierig zu messen. Auf jeden Fall kann hingegen gesagt werden, dass immer eine

Antwort von den LLMs zurückgekommen ist und es nie zu einem Timeout von seitens des LLMs gekommen ist.

6.3.2 Kostenübersicht

Die Kosten unterscheiden sich stark. Ein vollständiger Benchmark-Durchlauf mit 125 Fragen, mit jeweils 25 Fragen pro LLM, verursacht Kosten von 6.79 CHF. Am günstigsten war Deepseek mit 0.0028 CHF pro Fall und am teuersten war Grok mit 0.1352 CHF pro Fall. Dies entspricht einem Kostenfaktor von 48 gegenüber dem günstigsten Modell. Ein entscheidender Faktor der Kosten ist die Anzahl von Input und Output Tokens (Chen et al., 2023, S. 3). Einfachere Datenbanken verursachen hier deutlich geringere Kosten. In dieser Arbeit haben sich die Input Tokens im Schnitt auf 22’400 belaufen. Die grösste Mehrheit war die Goldtabellen mitzuschicken, und mit einer immer grösseren Favoritenliste durch den User-Feedback-Loop immer mehr unübersichtlich und führt zu einem Kostentreiber, wenn die Applikation nicht moderiert wird. Zudem stellt sich die Frage nach dem konkreten Anwendungsfall der App. Praktisch gesehen sind die durchschnittliche Antwortzeiten von 44 – 54 Sekunden für einen Benutzer sehr lange. Falls die Abfragen automatisch generiert werden oder nicht den End-Kunden betreffen, könnte sich eine App in dieser Form Sinn machen. Somit kann auch genauer eine Kostenüberprüfung gemacht werden, wie viele SQL-Abfragen erwartet werden und ob es sich lohnt, die SQL-Abfragen zu generieren lassen. Wenn die Komplexität der Aufgaben eher auf der schwierigen Seite liegt, müssen Experte die SQL-Abfragen entweder reparieren oder selbst schreiben. Entsprechend, ob ein solches System es sich lohnt einzusetzen, hängt vom konkreten Anwendungsfall und dem Budget ab.

7 Schlussfolgerungen

Diese Arbeit hat eine umfassende Evaluation anhand verschiedener LLMs im Kontext zu Text-to-SQL für räumliche Datenbanken mit PostGIS durchgeführt. Um messbare Werte zu erzielen, wurde ein Benchmark mit 25 Fragen erstellt mit diversen Schwierigkeiten. Fünf verschiedene LLMs haben den Benchmark mehrmals mit verschiedenen Einstellungen durchgeführt, um die Effektivität von Feedback-Loops zu evaluieren. Der Benchmark-Durchgang ohne Hilfe von Feedback-Loops hat 20% der Aufgaben korrekt beantwortet. Mit aktiviertem User-Feedback-Loop (Beispiel-Prompts mit zugehöriger SQL-Antwort) konnten 41,6 % der Aufgaben korrekt gelöst werden. Der Durchgang mit User-Feedback-Loop und Syntax-Feedback-Loop (bei leerer Datenbankantwort oder Syntaxfehlern) wurde mit 47.20% korrekt beantwortet. Im letzten Durchlauf mit beiden Feedback-Loops wurden alle Fehler manuell korrigiert, wodurch die Korrektheit auf 62,4 % anstieg. Eine Kritik an dieser Arbeit ist der strenge Bewertungsansatz, der eine 1:1 Übereinstimmung des Inhalts der Benchmark benötigt, um als korrekt bewertet zu werden. Es bedarf eine manuelle Korrektur, die durchaus fehleranfällig sein kann, um die wahre Leistung des LLMs zu entdecken. Ein weiterer Kritikpunkt ist die Umsetzung des User-Feedback-Loops, sprich die Favoritenliste, die mit jeder subsequenten Abfrage mitgeschickt wird. Schlechte Beispiele in der Favoriten-Liste können zu einer negativen Verzerrung der Resultate führen. Zudem wurde die Arbeit anhand eines eigenen Benchmarks bewertet mit nur 25 Fragen. Die Leistung des Benchmarks hätte sich stark verändern können, wenn schwierige Aufgaben durch einfache Aufgaben ersetzt wären. Ferner sind die OpenStreetMap-Daten und Datenbank Schema bereits den LLMs bekannt. Für weitere Arbeiten wäre es interessant zu sehen, wie die Leistung sich gegenüber einer unbekannten Datenbank aussieht. Für eine weitere Arbeit wäre es auch interessant zu sehen, wenn die SQL-Antwort zuerst mit dem LLM zurückgeschickt wird. Wenn das LLM zufrieden ist mit der Antwort der Datenbank, wird es automatisch der Favoritenliste hinzugefügt. Folgende Handlungsempfehlungen werden aus dieser Arbeit entnommen. Die Einbindung eines User-Feedback-Loops kann die Leistung signifikant steigern anhand qualitativ hochwertiger Beispiele. Der Syntax-Feedback-Loop ist sinnvoll in Text-to-SQL Abfragen, welche auch Datenbank Erweiterungen, wie zum Beispiel PostGIS verwenden. Die Leistung der Feedback-Loops hängt ab von der Qualität der gespeicherten Prompts und subsequenten SQL-Abfragen. Hierbei empfiehlt sich entweder eine Einschränkung der Schreibrechte für Favoriten oder eine aktive Moderation der Favoritenliste. Der Standardprompt benötigt ein solides Fundament mit Aufgabenstellung, Favoriten und Goldtabellen.

Literaturverzeichnis

Agarwal, S., & Rajan, K. S. (2016). Performance analysis of MongoDB versus PostGIS/PostGreSQL databases for line intersection and point containment spatial queries. *Spatial Information Research*, *24*(6), 671–677. https://doi.org/10.1007/s41324-016-0059-1

Ascoli, B. G., Kandikonda, Y. S. R., & Choi, J. D. (2025). *ETM: Modern Insights into Perspective on Text-to-SQL Evaluation in the Age of Large Language Models* (No. arXiv:2407.07313). arXiv. https://doi.org/10.48550/arXiv.2407.07313

Burgess, J., Hoffmann, S., & Topf, J. (2025). *Osm2pgsql* [Software]. osm2pgsql. https://osm2pgsql.org/doc/install.html

Chen, L., Zaharia, M., & Zou, J. (2023). *FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance* (No. arXiv:2305.05176). arXiv. https://doi.org/10.48550/arXiv.2305.05176

Corbellini, A., Mateos, C., Zunino, A., Godoy, D., & Schiaffino, S. (2017). Persisting big-data: The NoSQL landscape. *Information Systems*, *63*, 1–23. https://doi.org/10.1016/j.is.2016.07.009

Farquhar, S., Kossen, J., Kuhn, L., & Gal, Y. (2024). Detecting hallucinations in large language models using semantic entropy. *Nature*, *630*(8017), 625–630. https://doi.org/10.1038/s41586-024-07421-0

Fürst, J., Kosten, C., Nooralahzadeh, F., Zhang, Y., & Stockinger, K. (2025). *Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries* (Version 1) [Dataset]. OpenProceedings.org. https://doi.org/10.48786/EDBT.2025.13

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2023). *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation* (No. arXiv:2308.15363). arXiv. https://doi.org/10.48550/arXiv.2308.15363

Han, M., Park, S., Kim, S., & Kim, H. (2024). Bridging the gap between text-to-SQL research and real-world applications: A unified all-in-one framework for text-to-SQL. *Knowledge-Based Systems*, *306*, 112697. https://doi.org/10.1016/j.knosys.2024.112697

Head, C. B., Jasper, P., McConnachie, M., Raftree, L., & Higdon, G. (2023). Large language model applications for evaluation: Opportunities and ethical implications. *New Directions for Evaluation*, *2023*(178–179), 33–46. https://doi.org/10.1002/ev.20556

Hsu, L. S., & Obe, R. O. (2021). *PostGIS in Action, Third Edition*. *3*. https://www.manning.com/books/postgis-in-action-third-edition

Ilba, M. (2021). Parallel algorithm for improving the performance of spatial queries in SQL: The use cases of SQLite/SpatiaLite and PostgreSQL/PostGIS databases. *Computers & Geosciences*, *155*, 104840. https://doi.org/10.1016/j.cageo.2021.104840

Jiang, Y., & Yang, C. (2024). Is ChatGPT a Good Geospatial Data Analyst? Exploring the Integration of Natural Language into Structured Query Language within a Spatial Database. *ISPRS International Journal of Geo-Information*, *13*(1), 26. https://doi.org/10.3390/ijgi13010026

Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Cao, R., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C. C., Huang, F., Cheng, R., & Li, Y. (2023). *Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs* (No. arXiv:2305.03111). arXiv. https://doi.org/10.48550/arXiv.2305.03111

Lizardo, L. E. O., & Davis, C. A. (2017). A PostGIS extension to support advanced spatial data types and integrity constraints. *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 1–10. https://doi.org/10.1145/3139958.3140020

Martins, D. M. L. (2019). Reverse engineering database queries from examples: State-of-the-art, challenges, and research opportunities. *Information Systems*, *83*, 89–100. https://doi.org/10.1016/j.is.2019.03.002

Nascimento, E. R., García, G., Izquierdo, Y. T., Feijó, L., Coelho, G. M. C., De Oliveira, A. R., Lemos, M., Garcia, R. L. S., Leme, L. A. P. P., & Casanova, M. A. (2025). LLM-Based Text-to-SQL for Real-World Databases. *SN Computer Science*, *6*(2), 130. https://doi.org/10.1007/s42979-025-03662-6

OSM. (2025a). *OpenStreetMap Daten Schweiz* (Version Latest) [Dataset]. https://planet.osm.ch/switzerland-exact.osm.pbf

OSM. (2025b). *Osm2pgsql .style Datei* [Dataset]. https://raw.githubusercontent.com/openstreetmap/osm2pgsql/master/default.style.

PostGIS. (o. J.). *Documentation ST_Withiin*. https://postgis.net/docs/ST_Within.html

Pourreza, M., Li, H., Sun, R., Chung, Y., Talaei, S., Kakkar, G. T., Gan, Y., Saberi, A., Ozcan, F., & Arik, S. O. (2024). *CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL* (No. arXiv:2410.01943). arXiv. https://doi.org/10.48550/arXiv.2410.01943

Pourreza, M., & Rafiei, D. (2023). *Evaluating Cross-Domain Text-to-SQL Models and Benchmarks* (No. arXiv:2310.18538). arXiv. https://doi.org/10.48550/arXiv.2310.18538

Scalegrid. (2019, April 3). *2019 Database Trends – SQL™ vs. NoSQL, Top Databases, Single vs. Multiple Database Use* [Saas]. https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use

Silva, Y. N., Almeida, I., & Queiroz, M. (2016). SQL: From Traditional Databases to Big Data. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 413–418. https://doi.org/10.1145/2839509.2844560

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Zhong, S., Yin, B., & Hu, X. (2024). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. *ACM Transactions on Knowledge Discovery from Data*, *18*(6), 1–32. https://doi.org/10.1145/3649506

Zhai, B., Xu, C., He, Y., & Yao, Z. (2025). *ExCoT: Optimizing Reasoning for Text-to-SQL with Execution Feedback* (No. arXiv:2503.19988). arXiv. https://doi.org/10.48550/arXiv.2503.19988

Zhan, Z., Haihong, E., & Song, M. (2025). Leveraging Large Language Model for Enhanced Text-to-SQL Parsing. *IEEE Access*, *13*, 30497–30504. https://doi.org/10.1109/ACCESS.2025.3540072

Zhang, X., Xiao, F., Yan, L., & Zhang, Z. (2024). GS-SQL: Modeling Spatial Semantics in Spatial Text-to-SQL. *2024 International Joint Conference on Neural Networks (IJCNN)*, 1–7. https://doi.org/10.1109/IJCNN60899.2024.10651125

Anhang

Im Anhang befindet sich der Benchmark mit allen Aufgaben und die dazugehörige Musterlösung.

1. Find all restaurants within the administrative boundary of Zurich. Use geometry from planet_osm_point. Return osm_id, name. Use DISTINCT. Order the results by `name` in ascending order.

```
SELECT DISTINCT p.osm_id, p.name
FROM planet_osm_point p
JOIN planet_osm_polygon z
  ON ST_Within(p.way, z.way)
WHERE p.amenity = 'restaurant'
  AND z.boundary = 'administrative'
  AND z.admin_level = '8'
  AND LOWER(z.name) = 'zürich'
ORDER BY p.name ASC;
```

2. Find all hotels (`tourism = 'hotel'`) located within the administrative boundary of the canton Luzern. Use only geometries from `planet_osm_point` and polygon boundaries from `planet_osm_polygon`. Return `osm_id`, `name`. Use `ST_Within`. Order the results by `name` in ascending order.

```
SELECT
    p.osm_id,
    p.name
FROM
    planet_osm_point p
JOIN
```

```
    planet_osm_polygon w
    ON ST_Within(p.way, w.way)
WHERE
    p.tourism = 'hotel'
    AND w.boundary = 'administrative'
    AND w.admin_level = '4'
    AND LOWER(w.name) = 'luzern'
ORDER BY p.name ASC;
```

3. Find all distinct parks located within the administrative boundary of Bern. Use geometries from planet_osm_point transformation to SRID 4326 and boundary from planet_osm_polygon. Return `osm_id`, `name`, and geometry as WKT. Use `DISTINCT`. Order by `osm_id`.

```
SELECT DISTINCT
  p.osm_id,
  p.name,
  ST_AsText(ST_Transform(p.way, 4326)) AS geom
FROM planet_osm_point p
JOIN planet_osm_polygon a
  ON ST_Within(p.way, a.way)
WHERE
  p.leisure = 'park'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'bern'
```

```
ORDER BY p.osm_id;
```

4. Find all distinct bus stops in Bern. Only include points that fall within the administrative boundary of Bern. Use ST_Within and DISTINCT. Return osm_id, name. Order by osm_id.

```
SELECT DISTINCT
  p.osm_id,
  p.name
FROM planet_osm_point p
JOIN planet_osm_polygon a ON ST_Within(p.way, a.way)
WHERE
  (
    p.public_transport = 'platform' OR
    p.public_transport = 'bus_stop' OR
    p.highway = 'bus_stop'
  )
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'bern'
ORDER BY p.osm_id;
```

5. Show all distinct hospitals located within the administrative boundary of Baden. Use planet_osm_point for the hospital points and planet_osm_polygon for the administrative area. Return name, and geometry as WKT. Use DISTINCT. Order by name.

```
SELECT DISTINCT
  p.name,
```

```
  ST_AsText(p.way)
FROM planet_osm_point p
JOIN planet_osm_polygon a ON ST_Within(p.way, a.way)
WHERE
  p.amenity = 'hospital'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'baden'
ORDER BY p.name;
```

6. Show all distinct cinemas within the administrative boundary of Spreitenbach. Use planet_osm_point and planet_osm_polygon. Return osm_id, name. Use DISTINCT. Order by osm_id.

```
SELECT DISTINCT
  p.osm_id,
  p.name
FROM planet_osm_point p
JOIN planet_osm_polygon a ON ST_Within(p.way, a.way)
WHERE
  p.amenity = 'cinema'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'spreitenbach'
ORDER BY p.osm_id;
```

7. Show all distinct cafes located within 300 meters of ETH Zurich (coordinates 8.548, 47.376) using SRID 4326 for distance calculation. Return name as cafe_name and geometry as WKT. Use DISTINCT. Order by cafe_name.

```
SELECT DISTINCT
  p.name AS cafe_name,
  ST_AsText(p.way) AS location
FROM planet_osm_point p
WHERE
  p.amenity = 'cafe'
  AND ST_DWithin(
    ST_Transform(p.way, 4326)::geography,
    ST_SetSRID(ST_MakePoint(8.548, 47.376), 4326)::geography,
    300
  )
ORDER BY cafe_name;
```

8. List all distinct bridges that intersect with the river Limmat tagged as waterway = 'river'. Return name and geometry as WKT. Use DISTINCT. Order the results by name.

```
SELECT DISTINCT b.name, ST_AsText(b.way) AS geometry
FROM planet_osm_line b
JOIN planet_osm_line r
  ON ST_Intersects(b.way, r.way)
WHERE
  b.bridge IS NOT NULL
  AND r.waterway = 'river'
```

```
  AND r.name ILIKE 'Limmat'
ORDER BY b.name;
```

9. Return all ATMs within 500 meters of hotels inside the administrative boundary of Zürich. Return atm_id, atm_name, hotel_name, atm_geometry, and hotel_geometry. Use DISTINCT. Order by hotel_name, then atm_name.

```
SELECT DISTINCT
  atm.osm_id AS atm_id,
  atm.name AS atm_name,
  hotel.name AS hotel_name,
  ST_AsText(atm.way) AS atm_geometry,
  ST_AsText(hotel.way) AS hotel_geometry
FROM planet_osm_point atm
JOIN planet_osm_point hotel
  ON ST_DWithin(
    ST_Transform(atm.way, 4326)::geography,
    ST_Transform(hotel.way, 4326)::geography,
    500
  )
JOIN planet_osm_polygon geneva
  ON ST_Within(hotel.way, geneva.way)
WHERE
  atm.amenity = 'atm'
  AND hotel.tourism = 'hotel'
  AND geneva.boundary = 'administrative'
```

```
  AND geneva.admin_level = '8'
  AND LOWER(geneva.name) = 'zürich'
ORDER BY hotel_name, atm_name;
```

10. Find all distinct kindergartens within 500 meters of primary schools inside the administrative boundary of Bern. Use geometries from planet_osm_point and proximity via ST_DWithin in SRID 4326 with ::geography. Return kindergarten_id, kindergarten_name. Use DISTINCT. Order by kindergarten_name.

```
SELECT DISTINCT
  k.osm_id AS kindergarten_id,
  k.name AS kindergarten_name
FROM planet_osm_point k
JOIN planet_osm_point s
  ON ST_DWithin(
    ST_Transform(k.way, 4326)::geography,
    ST_Transform(s.way, 4326)::geography,
    500
  )
JOIN planet_osm_polygon a
  ON ST_Within(k.way, a.way)
WHERE
  k.amenity = 'kindergarten'
  AND s.amenity = 'school'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
```

```
  AND LOWER(a.name) = 'bern'
 ORDER BY kindergarten_name;
```

11. Show all distinct buildings tagged as building = 'public' located inside the administrative boundary of Bern. Return osm_id, name, and geometry as WKT. Use DISTINCT. Order by name.

```
SELECT DISTINCT
  b.osm_id,
  b.name,
  ST_AsText(b.way) AS geometry
FROM planet_osm_polygon b
JOIN planet_osm_polygon a
  ON ST_Within(b.way, a.way)
WHERE
  b.building = 'public'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'bern'
ORDER BY b.name;
```

12. Show all museums located inside the administrative boundary of Winterthur. Use SRID 4326 for WKT transformation. Return osm_id, name, tourism, and geometry as WKT. Order by name.

```
SELECT
  p.osm_id,
```

```
  p.name,
  p.tourism,
  ST_AsText(ST_Transform(p.way, 4326)) AS geometry
FROM planet_osm_point p
JOIN planet_osm_polygon a
  ON ST_Within(p.way, a.way)
WHERE
  p.tourism = 'museum'
  AND a.boundary = 'administrative'
  AND a.admin_level = '8'
  AND LOWER(a.name) = 'winterthur'
 ORDER BY p.name;
```

13. Find all bars within 300 meters of hotels. Only include bars and hotels where both geometries are within 300m in SRID 4326 using ST_DWithin and ::geography. Return hotel_id, hotel_name, bar_id, bar_name, and distance_meters. Order by hotel_name, then distance_meters.

```
SELECT
  h.osm_id AS hotel_id,
  h.name AS hotel_name,
  b.osm_id AS bar_id,
  b.name AS bar_name,
  ST_Distance(
    ST_Transform(h.way, 4326)::geography,
    ST_Transform(b.way, 4326)::geography
```

```
  ) AS distance_meters
FROM planet_osm_point h
JOIN planet_osm_point b
  ON ST_DWithin(
    ST_Transform(h.way, 4326)::geography,
    ST_Transform(b.way, 4326)::geography,
    300
  )
WHERE
  h.tourism = 'hotel'
  AND b.amenity = 'bar'
ORDER BY h.name, distance_meters;
```

14. Find all roads (NOT NULL) that intersect with rivers (NOT NULL) inside the administrative boundary of Lucerne. Use geometry from planet_osm_line. Return osm_id, name. Use DISTINCT. Order by name.

```
SELECT DISTINCT r1.osm_id, r1.name
FROM planet_osm_line r1
JOIN planet_osm_line r2 ON ST_Intersects(r1.way, r2.way)
JOIN planet_osm_polygon l ON ST_Within(r1.way, l.way)
WHERE r1.highway IS NOT NULL
  AND r2.waterway = 'river'
  AND l.boundary = 'administrative'
  AND l.admin_level = '8'
```

```
  AND LOWER(l.name) = 'luzern'
ORDER BY r1.name;
```

15. Return all restaurants within 100 meters of tourist attractions in Bern. Distance must be calculated with ST_DWithin using ::geograph in SRID 4326. Return restaurant_id, restaurant_name, attraction_id, attraction_name, and distance_meters. Order by restaurant_name.

```
SELECT DISTINCT r.osm_id AS restaurant_id, r.name AS restaurant_name,
        a.osm_id AS attraction_id, a.name AS attraction_name,
        ST_Distance(ST_Transform(r.way, 4326)::geography, ST_Transform(a.way, 4326)::geography) AS distance_meters
FROM planet_osm_point r
JOIN planet_osm_point a
  ON ST_DWithin(ST_Transform(r.way, 4326)::geography, ST_Transform(a.way, 4326)::geography, 100)
JOIN planet_osm_polygon g
  ON ST_Within(a.way, g.way)
WHERE r.amenity = 'restaurant'
  AND a.tourism = 'attraction'
  AND g.boundary = 'administrative'
  AND g.admin_level = '8'
  AND LOWER(g.name) = 'bern'
ORDER BY restaurant_name;
```

16. Find all overlapping landuse polygons inside Winterthur. Return landuse_a_id, landuse_b_id, landuse_a_type, landuse_b_type, and the overlapping geometry using ST_AsText(ST_Intersection(...)). Limit to 100 results. Order by landuse_a_id.

```
SELECT
  a.osm_id AS landuse_a_id,
  b.osm_id AS landuse_b_id,
  a.landuse AS landuse_a_type,
  b.landuse AS landuse_b_type,
  ST_AsText(ST_Intersection(a.way, b.way)) AS overlap_geom
FROM planet_osm_polygon a
JOIN planet_osm_polygon b
  ON a.osm_id < b.osm_id
  AND ST_Intersects(a.way, b.way)
  AND a.landuse IS NOT NULL
  AND b.landuse IS NOT NULL
JOIN planet_osm_polygon city
  ON ST_Contains(city.way, a.way)
WHERE city.name ILIKE 'winterthur'
  AND city.admin_level = '8'
  AND city.boundary = 'administrative'
ORDER BY landuse_a_id
LIMIT 100;
```

17. List all buildings (NOT NULL) that intersect with green areas (landuse IN ...). Filter to Zurich. Return building_id, building_name, green_area_id, green_type, building_geom, green_geom as WKT. Order by building_id.

```
SELECT
  b.osm_id AS building_id,
  b.name AS building_name,
  g.osm_id AS green_area_id,
  g.landuse AS green_type,
  ST_AsText(b.way) AS building_geom,
  ST_AsText(g.way) AS green_geom
FROM planet_osm_polygon b
JOIN planet_osm_polygon g
  ON ST_Intersects(b.way, g.way)
JOIN planet_osm_polygon z
  ON ST_Within(b.way, z.way)
WHERE
  b.building IS NOT NULL
  AND g.landuse IN ('grass', 'recreation_ground', 'forest', 'village_green', 'meadow')
  AND z.boundary = 'administrative'
  AND z.admin_level = '8'
  AND LOWER(z.name) = 'zürich'
 ORDER BY building_id;
```

18. Find all building polygons (NOT NULL) that share a border with any park. Return osm_id and geometry (ST_AsText). Order by osm_id.

```
WITH buildings AS (
  SELECT osm_id, way
  FROM planet_osm_polygon
  WHERE building IS NOT NULL
),
parks AS (
  SELECT way
  FROM planet_osm_polygon
  WHERE leisure = 'park'
)
SELECT b.*
FROM buildings b
JOIN parks p ON ST_Touches(b.way, p.way)
ORDER BY b.osm_id;
```

19. Calculate the average distance between supermarkets within the administrative boundary of Winterthur. Distance should be calculated using ST_Distance with ::geography using SRID 4326. Return the result as avg_distance_between_supermarkets_m with 2 decimal precision.

```
WITH supermarkets AS (
  SELECT osm_id, way
  FROM planet_osm_point
  WHERE shop = 'supermarket'
    AND ST_Within(
      way,
      (SELECT way FROM planet_osm_polygon
```

```
      WHERE LOWER(name) = 'winterthur'
        AND boundary = 'administrative'
        AND admin_level = '8'
      LIMIT 1)
    )
),
distances AS (
  SELECT s1.osm_id,
        MIN(ST_Distance(
          ST_Transform(s1.way, 4326)::geography,
          ST_Transform(s2.way, 4326)::geography
        )) AS nearest_distance_m
  FROM supermarkets s1
  JOIN supermarkets s2 ON s1.osm_id != s2.osm_id
  GROUP BY s1.osm_id
)
SELECT ROUND(AVG(nearest_distance_m)::numeric, 2) AS
avg_distance_between_supermarkets_m
FROM distances;
```

20. Find all hotels with more than 3 bars within a 200-meter radius. Transform to SRID 3857. Return osm_id, name, and bar_count. Only return hotels where bar_count > 3. Order by bar_count DESC.

```
WITH hotels AS (
  SELECT osm_id, name, ST_Transform(way, 3857) AS geom
```

```
  FROM planet_osm_point
  WHERE tourism = 'hotel'
),
bars AS (
  SELECT ST_Transform(way, 3857) AS geom
  FROM planet_osm_point
  WHERE amenity = 'bar'
),
bars_near_hotels AS (
  SELECT h.osm_id, h.name, COUNT(b.*) AS bar_count
  FROM hotels h
  JOIN bars b
    ON ST_DWithin(h.geom, b.geom, 200)
  GROUP BY h.osm_id, h.name
)
SELECT * FROM bars_near_hotels WHERE bar_count > 3 ORDER BY bar_count DESC;
```

21. Find all industrial zones within the administrative boundary of Winterthur that are within 200 meters of a residential area. Use SRID 2056 for all transformations. Return osm_id, name. Order by osm_id.

```
WITH winterthur_boundary AS (
  SELECT ST_Transform(way, 2056) AS geom
  FROM planet_osm_polygon
  WHERE boundary = 'administrative'
    AND admin_level = '8'
```

```
    AND LOWER(name) = 'winterthur'
  LIMIT 1
),
residential_areas AS (
  SELECT ST_Transform(way, 2056) AS geom
  FROM planet_osm_polygon, winterthur_boundary
  WHERE landuse = 'residential'
    AND ST_Intersects(ST_Transform(way, 2056), winterthur_boundary.geom)
),
industrial_areas AS (
  SELECT osm_id, name, ST_Transform(way, 2056) AS geom
  FROM planet_osm_polygon, winterthur_boundary
  WHERE landuse = 'industrial'
    AND ST_Intersects(ST_Transform(way, 2056), winterthur_boundary.geom)
)
SELECT i.osm_id, i.name
FROM industrial_areas i
JOIN residential_areas r ON ST_DWithin(i.geom, r.geom, 200)
ORDER BY i.osm_id;
```

22. Identify all roundabouts located within the administrative boundary of the canton Zug. Return osm_id, name, and geom as WKT. Order by osm_id.

```
SELECT l.osm_id, l.name, ST_AsText(l.way) AS geom
FROM planet_osm_line l
JOIN planet_osm_polygon p ON ST_Within(l.way, p.way)
```

```
WHERE l.junction = 'roundabout'
  AND p.boundary = 'administrative'
  AND p.admin_level = '4'
  AND LOWER(p.name) = 'zug'
ORDER BY l.osm_id;
```

23. List all tunnels (NOT NULL) in the administrative boundary of St. Gallen name ILIKE '%st. gallen%' that intersect a river (NOT NULL). Return osm_id, name. Order by osm_id.

```
SELECT t.osm_id, t.name
FROM planet_osm_line t
JOIN planet_osm_line r ON ST_Intersects(t.way, r.way)
JOIN planet_osm_polygon a ON ST_Within(t.way, a.way)
WHERE t.tunnel IS NOT NULL
  AND r.waterway IS NOT NULL
  AND a.admin_level = '8' AND LOWER(a.name) LIKE '%st. gallen%'
ORDER BY t.osm_id;
```

24. Find all churches that are at least 10 years old based on the start_date tag. Return osm_id, name, start_date from tags, and geometry (way). Order by osm_id.

```
SELECT
  osm_id,
  name,
  tags -> 'start_date' AS start_date,
  way
FROM planet_osm_point
WHERE amenity = 'place_of_worship'
```

```
  AND tags ? 'start_date'
  AND (tags -> 'start_date') ~ '^\d{4}$'
  AND CAST(tags -> 'start_date' AS INTEGER) < (EXTRACT(YEAR FROM
CURRENT_DATE) - 10)
ORDER BY osm_id;
```

25. List all post offices located within shopping centers. Return post_office (name of point), and shopping_center (name of polygon). Order by post_office.

```
SELECT p.name AS post_office, s.name AS shopping_center
FROM planet_osm_point p
JOIN planet_osm_polygon s ON ST_Within(p.way, s.way)
WHERE p.amenity = 'post_office'
  AND s.shop = 'mall'
ORDER BY p.name;
```