Herausforderungen bei der Generierung von SQL-Statements mithilfe von LLMs

Patrik Graf, merlin.zwo InfoDesign

Die Generierung von SQL-Statements durch Large Language Models (LLMs) wie GPT-4 von OpenAI bietet viele Möglichkeiten, birgt jedoch auch Herausforderungen. Während LLMs durch ihr umfangreiches Wissen prinzipiell in der Lage sind, komplexe SQL-Abfragen zu erstellen, gibt es typische Probleme wie Halluzinationen, syntaktische Fehler oder die Nutzung nichtexistierender Tabellen und Spalten. In diesem Artikel teile ich meine Erfahrungen, beleuchte die häufigsten Herausforderungen und zeige Lösungen sowie Best Practices auf, die sich in der Praxis bewährt haben.

Typische Herausforderungen

1. Halluzinationen und fehlerhafte Ergebnisse

LLMs neigen dazu, Informationen zu erfinden, da sie auf statistischen Mustern in der natürlichen Sprache basieren und keine direkte Anbindung an eine spezifische Datenbank haben. Dadurch erstellen sie oft scheinbar plausible, aber falsche SQL-Statements zum Beispiel mit nichtexistierenden Tabellen oder Spalten *(siehe Listing 1)*. Das Modell versucht, Lücken zu füllen, indem es basierend auf Wahrscheinlichkeiten Begriffe ergänzt, die zwar in ähnlichen Kontexten vorkommen, aber nicht zwangsläufig in der realen Datenbank existieren. Ein Beispiel ist die Verwendung von Spaltennamen, die plausibel klingen, aber nicht in der zugrunde liegenden Datenbank existieren.

2. Inkonsistente oder syntaktisch fehlerhafte SQL-Abfragen

Modelle wie GPT-3.5-Turbo haben häufig Probleme mit der korrekten SQL-Syntax. Dies liegt vor allem daran, dass LLMs nicht über ein explizites Verständnis von Grammatikregeln verfügen. Dadurch entstehen oft Fehler in der Struktur der

```
SELECT customer_name,
       order_total
  FROM sales_data
 WHERE order_date > '01.01.2024';
```

Listing 1: Die Tabelle sales_data existiert nicht, oder die Spalte customer_name ist falsch

```
SELECT product_name,
       SUM(sales_amount
  FROM orders
 WHERE order_date BETWEEN '01.01.2023' AND '31.12.2023';
```

Listing 2: Fehlende schließende Klammer in der SUM()-Funktion

```
{
    "answer": "Hier ist Ihre SQL-Abfrage.",
    "sql": "SELECT SUM(sales_amount) FROM orders WHERE
order_date > '01.01.2024';"
}
```

Listing 3: Beispiel für eine strukturierte Ausgabe im JSON-Format

SQL-Abfragen. Die erzeugten Abfragen enthalten manchmal falsche Klammern oder unvollständige JOIN-Bedingungen *(siehe Listing 2)*. Ein weiterer Grund für diese Fehler ist, dass SQL eine sehr strukturierte und formale Sprache ist, während LLMs auf natürlichen Sprachmustern basieren. Ohne explizite Regelwerke für SQL-Syntax können sie Syntaxfehler wie fehlende oder falsch platzierte Zeichen generieren. Zudem haben kleinere Modelle wie GPT-3.5-Turbo ein eingeschränktes Kontextfenster, wodurch sie die Gesamtstruktur längerer SQL-Abfragen nicht immer korrekt erfassen, und dadurch unvollständige oder fehlerhafte Statements produzieren.

3. Fehlende Kontextsensitivität

LLMs haben in der Regel keine Kenntnis über die spezifische Datenbankstruktur eines Unternehmens. Zudem fehlt ihnen die direkte Anbindung an diese Datenbanken. Sie können nur durch den bereitgestellten Kontext mit den nötigen Informationen zur Datenbankstruktur versorgt werden, denn ohne detaillierte Tabelleninformationen kann das Modell keine maßgeschneiderten Abfragen generieren. Hierbei spielen DDL-Statements eine entscheidende Rolle. Durch das Bereitstellen von CREATE TABLE-Anweisungen oder DESCRIBE TABLE-Ergebnissen erhält das Modell eine präzisere Vorstellung der vorhandenen Strukturen. Diese zusätzlichen Informationen ermöglichen es, realistische und syntaktisch korrekte SQL-Abfragen zu generieren, die auf den tatsächlich vorhandenen Strukturen basieren. Ebenso wichtig sind Tabellen- und Spaltenkommentare, da sie wertvolle Metadaten über die Bedeutung der einzelnen Tabellen und Felder liefern. Sie helfen dem Modell, den semantischen Kontext besser zu verstehen. Wenn diese Informationen bereitgestellt werden, kann das LLM gezieltere und fachlich korrektere Abfragen generieren, da es die geschäftliche Bedeutung der Datenbankstrukturen berücksichtigt.

Strukturierte Ausgabeformate zur Weiterverarbeitung

Um ein LLM besser in bestehende Workflows zu integrieren, bietet sich die Ergebnisrückgabe im JSON-Format an, da es eine einfache Weiterverarbeitung ermöglicht *(siehe Listing 3)*. Natürlich sind auch andere strukturierte Formate als Rückgabe möglich, wie beispielsweise XML, jedoch bietet hier JSON entscheidende Vorteile:

- Einfach lesbar
- Geringer struktureller Overhead
- Typenunterstützung (z. B. bool, string, etc.)
- Sehr gute Integration in moderne Web-Technologien

Ein Beispiel für einen effektiven Prompt

Ein klar definierter Prompt verbessert die Qualität der generierten SQL-Statements erheblich. Das folgende Beispiel *(siehe Listing 4)* gibt dem LLM detaillierte Anweisungen, um fehlerfreie und strukturierte SQL-Abfragen zu erstellen:

Ein solcher detaillierter Prompt hilft, die generierten SQL-Abfragen zu verbessern und häufige Fehler zu vermeiden. Im Folgenden gehe ich nochmal auf den Zweck der wichtigsten Punkte im Prompt ein, um zu verdeutlichen, was damit im Detail bezweckt wird:

- Strukturierte JSON-Antwort
  Dies ermöglicht die strukturierte Weiterverarbeitung durch ein nachgelagertes System.
- Erstellung eines praxisnahen SQL-Statements
  Das LLM soll vorhandene Spalten und Tabellen nutzen und keine Elemente einfügen. Dadurch wird sichergestellt, dass das generierte SQL hinsichtlich der Datenobjekte tatsächlich ausführbar ist.
- Mandanteneinschränkung
  Falls Tabellen eine Mandantenspalte enthalten, wird mit dem Wert des APEX-Feldes „GLOBAL_MANDANT“ gefiltert. Dies stellt sicher, dass nur die Daten des aktuellen Mandanten abgefragt werden, was für Multi-Tenant-Systeme wichtig ist.
- Benutzerspezifische Abfragen
  Falls sich ein Benutzer auf sich selbst bezieht, wird seine ID durch die APEX-Felder „GLOBAL_M_NR“ oder „APP_USER“ ermittelt, wodurch personalisierte Abfragen möglich werden – ohne direkte Eingabe von Benutzerinformationen.
- Sortierregel: NULLS LAST
  Stellt sicher, dass NULL-Werte am Ende der Sortierung stehen, was bei unseren Anwendungsfällen eine bessere Benutzererfahrung bot.
- NVL für NULL-Werte in numerischen und Datumsspalten
  Verhindert, dass NULL-Werte ungewollt zu Problemen in Berechnungen oder Aggregationen führen.
- Datum als Variable $[AIL_HEUTE]
  Da das LLM das heutige Datum nicht kennt, wird dieses dynamisch in den Prompt eingefügt. Dies verbessert die Generierung von zeitabhängigen SQL-Abfragen.
- Deutsche Zahlen- und Datumsformate
  Alle Datums- und Zahlenwerte sollen mit deutschen Formatmasken ausgegeben werden.
- Definition des Geschäftsjahres durch $[MANDANT_GJ_VON_BIS]
  Geschäftsjahre werden durch eine Variable dynamisch eingefügt, was mandantenspezifische Geschäftsjahresdefinitionen ermöglicht.
- Ausgabe ausschließlich als JSON
  Dadurch wird sichergestellt, dass keine zusätzlichen Texte oder Markup-Tags in der Ausgabe enthalten sind. Bei „älteren“ Modellen sollte dies nicht fehlen.

Unterschiede zwischen den Modellen

Ein Vergleich zwischen verschiedenen LLMs zeigt signifikante Unterschiede in der Qualität der generierten SQL-Statements. Beispielsweise lieferte GPT-4o eine deutlich präzisere Abfrage für die Umsatzermittlung als GPT-3.5-Turbo, welches oft syntaktische Fehler enthielt. Modelle mit größerem Kontextfenster (z. B. GPT-4-Turbo) tendieren dazu, genauere und vollständigere Abfragen zu generieren, insbesondere wenn komplexe Tabellenbeziehungen involviert sind.

Im Folgenden sehen wir einige generierte SQLs zur Frage „Wie war der Umsatz im letzten Geschäftsjahr?“, jeweils mit unterschiedlichen Modellen von OpenAI *(siehe Listings 5-8)*. Als Datenbasis diente unser hauseigenes ERP-System.

```
Dein Job ist es, die Anfrage des Benutzers zu analysieren und darauf folgende Schritte auszuführen:
1. Erstelle ein JSON-Objekt mit den Attributen "answer" und "sql".
2. Generiere ein SQL-Statement basierend auf dem gegebenen Kontext, um die gewünschten Informationen zu ex-
trahieren. Nutze dein Allgemeinwissen um Informationen im SQL zu ergänzen, wenn diese nicht im gegebenen
Kontext auffindbar sind. Verwende dabei nur existierende Spalten aus den vorgegebenen Tabellen. Du darfst keine
hypothetischen Spalten oder Tabellen verwenden. Achte darauf, dass das generierte SQL praxisnah ist und sich
auf real existierende Daten bezieht. Falls mehrere Tabellen erforderlich sind, integriere diese sinnvoll.
3. Erstelle für jede Ausgabespalte einen gut lesbaren Alias. Umschließe den Alias in doppelten Anführungsze-
ichen.
4. Wenn Mandantenspalten in den integrierten Tabellen vorhanden sind, schränke per Funktionsaufruf v('GLOB-
AL_MANDANT') auf einen Mandanten ein.
5. Nutze als Ausgabespalten niemals Primärschlüssel oder Fremdschlüssel.
6. Der Principal Name von Mitarbeitern ist immer ".....", alles in Kleinbuchstaben.
7. Wenn der Benutzer Daten über sich selbst wünscht, kann seine MAMI_ID über den Funktionsaufruf v('GLOBAL_M_
NR') und seine MAMI_LDAP_UID über den Funktionsaufruf v('APP_USER') ermittelt werden.
8. Benutze beim Sortieren immer NULLS LAST.
9. Benutze bei numerischen Spalten und bei Datumsspalten, die NULLABLE sind, immer die Funktion NVL.
10. Benutze für relative Zeitangaben immer das aktuelle Datum. Das aktuelle Datum ist $[AIL_HEUTE] im Format
DD.MM.YYYY
11. Finde und filtere Zeilen in einer Tabelle, deren Bezeichnungsspalten eine Ähnlichkeit von mindestens 85%
mit einem gegebenen Suchbegriff aufweisen. Nutze die UTL_MATCH.JARO_WINKLER_SIMILARITY-Funktion, um die genaue
Übereinstimmung unter Berücksichtigung von Transpositionen und Vorsilbenverstärkung zu analysieren. Gib nur
die Zeilen zurück, deren Ähnlichkeitswerte für Bezeichnungs- und Namensspalten >= 85% liegen. Namensspalten
für Personen, wie Vorname, Nachname oder Kombinationen daraus, sollen auf normale Art gefiltert werden.
12. Benutze für Abfragen mit Geodaten immer die Funktionen von Oracle Spatial.
13. Schreibe eine Antwort im Stil eines Gespräches unter guten Kollegen in das Attribut "answer". Informa-
tionen, die du nicht kennst, werden durch das generierte SQL-Statement ermittelt. Halte deine Antwort allge-
mein. Präsentiere in einfachen Worten, ohne technische Details, das Ergebnis. Erwähne die folgenden Begriffe
nicht in deiner Antwort: SQL, Datenbank.
14. Alle Datums- und Zahlenformate sind in deutsch und die Formatmasken sind Oracle-SQL Formatmasken.
15. Bei Anfragen zu Geschäftsjahren gilt ein Geschäftsjahr vom $[MANDANT_GJ_VON_BIS] des Folgejahres.
16. Deine Ausgabe darf nur das JSON-Objekt enthalten.

Kontext, Oracle SQL:
%%BUSINESS_CONTEXT_SQL%%
```

Listing 4: Ein ausführlicher Prompt zum Generieren von lauffähigem SQL

```
SELECT NVL(SUM(repo_wert_rabattiert), 0) AS "Umsatz"
  FROM erp.rng_pos rp
  JOIN erp.rng_kopf rk ON rp.repo_reko_id = rk.reko_id
 WHERE rk.reko_gj = EXTRACT(YEAR FROM ADD_MONTHS(SYSDATE, -3);
```

Listing 5: GPT-3.5-Turbo mit fehlerhafter SQL-Syntax (am Ende fehlt eine Klammer). Zudem ist das SQL fachlich falsch

```
SELECT SUM(NVL(REPO_WERT_RABATTIERT, 0)) AS "Umsatz"
  FROM ERP.RNG_POS
  JOIN ERP.RNG_KOPF ON RNG_POS.REPO_REKO_ID = RNG_KOPF.REKO_ID
 WHERE REKO_GJ = TO_NUMBER(TO_CHAR(ADD_MONTHS(SYSDATE, -6),
'YY')) -- FALSCH!
   AND REKO_MAND_ID = v('GLOBAL_MANDANT')
```

Listing 6: GPT-4-Turbo mit ausführbarem SQL, jedoch fachlich falsch

Lessons Learned: Best Practices

- Präzise Prompts formulieren
  Ein klar definierter Prompt mit detaillierten Anweisungen reduziert Halluzinationen deutlich.
- Einsatz von JSON als Rückgabeformat
  Standardisierte Strukturen erleichtern die Weiterverarbeitung und Integration in bestehende Systeme.
- Kontinuierliche Optimierung
  Durch iterative Anpassung der Prompts wird die Qualität der SQL-Generierung immer weiter verbessert. Neue Modelle müssen gut darauf getestet werden, ob ihre „SQL-Fähigkeiten“ in der Praxis Verbesserungen ermöglichen.

Variable	Bedeutung
$[AIL_HEUTE]	Aktuelles Datum im Format DD.MM.YYYY.
$[MANDANT_GJ_VON_BIS]	Zeitraum für das Geschäftsjahr des Mandanten.
%%BUSINESS_CONTEXT_SQL%%	DDL Statements (CREATE TABLE etc.)

Tabelle: Bedeutung der Variablen im Prompt

```
SELECT SUM(NVL(REPO_WERT_RABATTIERT, 0)) AS "Umsatz"
  FROM ERP.RNG_POS
  JOIN ERP.RNG_KOPF ON RNG_POS.REPO_REKO_ID = RNG_KOPF.REKO_ID
 WHERE RNG_KOPF.REKO_DATUM BETWEEN TO_DATE('01.07.2023', 'DD.MM.YYYY')
  AND TO_DATE('30.06.2024', 'DD.MM.YYYY') -- KORREKT!
   AND RNG_KOPF.REKO_MAND_ID = v('GLOBAL_MANDANT')
```

Listing 7: GPT-4o mit ausführbarem und fachlich richtigem SQL

```
SELECT NVL(SUM(REPO_WERT), 0) AS "Gesamtumsatz" -- FALSCH!
  FROM ERP.RNG_POS RP
  JOIN ERP.RNG_KOPF RK ON RP.REPO_REKO_ID = RK.REKO_ID
 WHERE RK.REKO_GJ = EXTRACT(YEAR FROM SYSDATE) - 1 -- FALSCH!
   AND RK.REKO_DATUM >= TO_DATE('01.07.' || (EXTRACT(YEAR FROM SYSDATE)
- 1), 'DD.MM.YYYY')
 AND RK.REKO_DATUM <= TO_DATE('30.06.' || EXTRACT(YEAR FROM SYSDATE),
'DD.MM.YYYY')
   AND RK.REKO_MAND_ID = v('GLOBAL_MANDANT')
```

Listing 8: GPT-4o-Turbo mit ausführbarem SQL, jedoch fachlich falsch

Fazit und Ausblick

LLMs haben durchaus das Potenzial, SQL-Statements effizient zu generieren, aber sie sind noch nicht auf menschlichem Niveau. Durch präzise Prompts, strukturierte Ausgabeformate wie JSON und eine kontinuierliche Optimierung des bereitgestellten Kontexts lassen sich aber auch schon heute viele Herausforderungen bewältigen. Diese Technologie ist jedoch so schnelllebig, dass dieser Artikel wahrscheinlich schon wieder veraltet ist, bis sie Ihn zu lesen bekommen. Die Modelle werden immer besser im Bewältigen von Entwicklungsaufgaben; in naher Zukunft könnten vielleicht sogar auf SQL spezialisierte Modelle, mit direkten Anbindungsmöglichkeiten an Datenbanken, dies alles hier vollkommen überflüssig machen. Die Zukunft bleibt auf jeden Fall spannend.

Über den Autor

Patrik Graf ist Head of Innovation bei merlin.zwo InfoDesign GmbH & Co. KG und beschäftigt sich intensiv mit der Integration von KI in datenbankgestützten Systemen. Er ist regelmäßiger Sprecher auf Fachkonferenzen und teilt seine Erfahrungen in Artikeln und Vorträgen.

Patrik Graf
patrik.graf@merlin-zwo.de