Research Report

OmniscientDB: A Large Language Model-Augmented DBMS That Knows What Other DBMSs Do Not Know

WE PRESENT OUR VISION OF OMNISCIENTDB, A NOVEL DATABASE THAT LEVERAGES THE IMPLICITLY STORED KNOWLEDGE IN LARGE LANGUAGE MODELS TO AUGMENT DATA SETS FOR ANALYTICAL QUERIES OR MACHINE LEARNING TASKS. OMNISCIENTDB EMPOWERS USERS TO AUGMENT DATA SETS BY MEANS OF SIMPLE SQL QUERIES AND THUS HAS THE POTENTIAL TO DRAMATICALLY REDUCE THE MANUAL OVERHEAD ASSOCIATED WITH DATA INTEGRATION. IT USES AUTOMATIC PROMPT ENGINEERING TO CONSTRUCT APPROPRIATE PROMPTS FOR GIVEN SQL QUERIES AND PASSES THEM TO A LARGE LANGUAGE MODEL LIKE GPT-3 TO CONTRIBUTE ADDITIONAL DATA, AUGMENTING THE EXPLICITLY STORED DATA. OUR INITIAL EVALUATION DEMONSTRATES THE GENERAL FEASIBILITY OF OUR VISION, EXPLORES DIFFERENT PROMPTING TECHNIQUES IN GREATER DETAIL, AND POINTS TOWARDS FUTURE RESEARCH.

Matthias Urban

Duc Dat Nguyen

Carsten Binnig

Introduction

Traditionally, relational databases are required to explicitly capture the reality of a particular domain, meaning that all relevant facts need to be stored in the database in order to be queried by the user. However, this so-called closed-world assumption significantly limits the ways in which the information in a database can be used. For example, think of a database that stores information about movies and their revenues. While a breakdown of the revenue by director might be an interesting query a user wants to issue, this information might not be stored in the database. Today, the only way to make additional information available for querying or other downstream tasks (e.g., training a Machine Learning (ML) model) is to explicitly integrate additional data sources into the database, which requires extensive manual efforts for every data source.

Idea and Simple Example

In this paper, we thus present our vision of OmniscientDB, an open-world Data Base Management System (DBMS) that can automatically augment existing databases with world knowledge for the execution of Structured Query Language (SQL) queries. To do so, OmniscientDB can not only generate additional tables on-the-fly but also complete existing tables with user-requested rows or columns. To enable this, OmniscientDB makes use of the world knowledge that is implicitly stored in large language models (LLMs, e.g. GPT-3 (Brown et al., 2020)).

To illustrate the benefits of OmniscientDB by an example, imagine a data scientist trying to analyze the revenues of recent movies as mentioned before. For the questions of the data scientist, important information like the starring actors or directors, however, is not contained in the data set, despite their potentially large impact on the movies' revenues. While traditionally such information would need to be explicitly integrated first, with OmniscientDB the data scientist could simply use the knowledge stored in the LLM for augmentation. For instance, OmniscientDB could automatically generate the missing information about the actors starring in the movies, allowing a more extensive analysis.

Virtual Tables

OmniscientDB leverages the knowledge implicitly stored in the parameters of LLMs for augmentation and makes it available for querying via SQL using so-called virtual tables. Virtual tables can be treated by users just like traditional tables in relational databases, e.g. they can be used as operands in traditional query operators.

However, they are not explicitly stored in the database but instead act as a proxy for the knowledge stored in LLMs. For instance, in the example above, the user is able to join the movies table with the virtual actors table to gen-

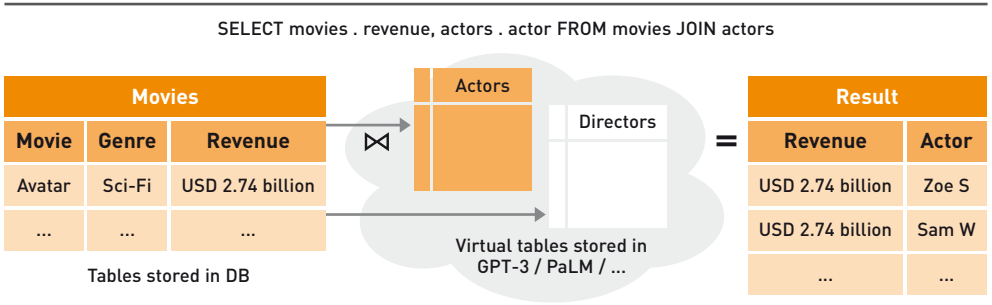

Figure 1: SQL Query That Joins a Movies Table Stored in the Database with an Actors Table Implicitly Stored in the Weights of a LLM (e.g., GPT-3).

erate additional information about actors, as shown in Figure 1. The information about actors is materialized on-the-fly during the execution of the query.

Virtual tables are an easy way for the data scientist to also take actor information into account to perform the analysis.

In particular, it is important to mention that this way of augmenting data sets comes with almost zero manual overhead. The data scientist just has to formulate a SQL query involving virtual tables, and the database will automatically generate the necessary information to perform the join.

The knowledge available in OmniscientDB via virtual tables is obviously bound by the knowledge stored in LLMs. However, recent LLMs have been trained on huge data sets and by now scale to hundreds of billions of parameters (Brown et al., 2020; Chowdhery et al., 2023), leading to tremendous amounts of knowledge stored in their parameters. In the Challenges Section, we will discuss further opportunities how LLMs can tap into knowledge not stored in their parameters.

Case Study

To show the general feasibility of OmniscientDB, we present a case study in which we examine the quality of materialized LLM knowledge for virtual tables. To materialize knowledge from LLMs, we automatically generate so-called prompts, which are short strings we provide as input to the LLM. The LLM is trained to complete such prompts and thereby reveal the knowledge stored inside of it. In the case study, we examine different variations on how prompts for virtual tables can be generated on different real-world data sets. All experiments for the case study are performed with the GPT-3 davinci model by OpenAI.

Structured Prompts vs. NL Prompts

Our default way of creating prompts to extract data for columns of a virtual table is to linearize entire table rows to obtain a (structured) prompt as follows:

Title: Ant man | Year: 2017 | ... | Actors:

This prompt would let the LLM generate all main actors appearing in Ant man. However, since LLMs are pre-trained on natural language, we might obtain better results by using prompts that are the beginning of natural language sentences (NL prompts). Prompts in NL for database operations are hard to obtain though since they require formulating the information in the table as NL sentences. They either have to be created manually, or by using templates, or they could be the output of a sequence-to-sequence ML model. On the other hand, they might result in generated values of higher quality.

To find out if it makes sense to put the effort into constructing good NL prompts, we compare structured prompts with NL prompts in a simplified setting. We consider a table with a single explicitly stored column and would like to add more columns using LLM knowledge. For data sets, we use the cities data set (1,000 largest cities; Kaggle, 2017), where the single column is the *City Name* and we want to add columns for *Longitude* and *Latitude*.

Additionally, we use the IMDB Movies data set (first 1,000 movies; OpenML, 2022), where the single column is the movie *Title*, and the LLM should generate values for the *Release Year*, *Genre*, *Runtime*, *Actors* and *Director*. Hence, we cover two wildly different domains and a variety of data formats such as text, float numbers, and dates. In this case, the structured prompts, as introduced before, result in prompts like *Title: Avatar | Runtime:*, while the NL prompt is *The runtime of Avatar is*. For this initial evaluation, we measure how many cells are filled correctly and report an accuracy value averaged over all cells. Longitude and Latitude (for the city data set) are considered correct if the difference to the ground truth value is below 0.1 and for columns where we expect multiple values (e.g., Genre) cells are considered correctly filled if the model predicts at least one correct value.

The result (Table 1) shows that, perhaps surprisingly, structured prompts perform better than NL prompts. For the movies table the difference is almost 20% in accuracy. Hence, we decide to use structured prompts, since they are easier to generate and yield more accurate generated values.

	Cities	Movies
Natural Language Prompt	64.65%	51.70%
Structured Prompt	71.65%	71.58%

Table 1: Accuracy of Natural Language Prompts and Structured Prompts on the Two Data Sets Cities and Movies.

Variants of Structured Prompts

Next, we investigate what information should be part of the prompt to obtain generated values of high quality. We fix the type of prompt to structured prompts and consider two kinds of information to add to the prompt: row information and example values. In our prompt, as described in the beginning of this section, we linearize entire rows to obtain the prompts and thus already include the full row information in the prompt (Prompt: *Title: Ant man | Year: 2017 | ... | Runtime:*).

We compare this prompt with a minimal prompt that only includes the value for the first attribute (Prompt: *Title: Ant man | Runtime:*). On top of that, we explore how additional example values (e.g., runtimes of other movies) affect the quality of the extraction. Hence, we construct prompts that always begin with two example rows that contain a value for the column to be materialized. For the movies data set, a prompt might look like this:

Title: Avengers: Endgame | ... | Runtime: 2h 23m
Title: Spiderman: Homecoming | ... | Runtime: 2h 13m
Title: Avatar | Runtime:

Table 2 shows that prompts including row information help the model identify the piece of knowledge that it should generate and provide example values whenever they are available.

	Cities	Movies
minimal	71.65%	71.58%
+ row info	75.00%	80.18%
+ examples	73.65%	79.78%

Table 2: Accuracy of Different Structured Prompts on the Two Data Sets Cities and Movies (+ Row Info Includes Additional Information from the Same Row; + Examples Includes Example Values for the Column to Be Materialized).

Challenges

Due to the vast amount of knowledge stored in the parameters of LLMs, they present ample opportunities to augment existing data sets as presented in our case study. However, they also pose unique challenges as we explore in the following:

External Knowledge

While large LLMs store large amounts of publicly available knowledge, they are still far from being omniscient. In particular, knowledge has to be present sufficiently often in the pre-training data set such that the LLM is able to recall it. To make OmniscientDB live up to its name, it has to be able to also tap into external knowledge not stored in its weights. For instance, if LLMs are able to access open data sets on the Web or private data sets in data lakes, they would be able to augment the data sets at hand much more effectively. Fortunately, work on retrieval-augmented language models (Guu et al., 2020) has shown that it is in principle possible to let LLMs utilize external knowledge. More recently, researchers have even proposed language models that search the Web (Nakano et al., 2023). However, it is not yet clear how LLM can be used to retrieve structured data sets (i.e., tables) and not only NL passages from external corpora.

Trust & Hallucination

Probably the most critical challenge is that LLMs are known to suffer from hallucination, a phenomenon where LLMs generate non-factual statements. In the context of OmniscientDB, this means that it might generate values during the materialization of virtual tables that sound plausible but are not actually correct. This not only reduces users' trust in the augmentation of OmniscientDB but might also negatively affect downstream analytical queries or ML applications. However, it is already possible to extract well-calibrated certainty scores from LLMs (Kadavath et al., 2023), which could be used to detect hallucinations. As such, there already exist techniques to mitigate the risks of LLMs and future research will further reduce the amount of generated values that are factually wrong. For applying these trends to the generation of structured data like table rows or columns, as considered here, again additional research will be needed.

Conclusion

OmniscientDB is our vision of how the knowledge stored in the parameters of LLMs can be integrated into databases. The concept of virtual tables allows users to seamlessly integrate such knowledge into their existing data sets with just a few SQL queries and without any manual overhead. In our case study, we showed the general feasibility of our ideas by experimenting with different automatically-generated prompts to materialize virtual tables and columns. However, several interesting research challenges lie ahead to further enhance the usefulness of OmniscientDB. In particular, we are interested in how language models that are able to independently search the Web could enable an entirely automated data augmentation process, where materialized columns and tables are not bound by the knowledge that can be stored in model parameters, but by knowledge accessible via the Internet.

References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; et al.:
Language Models Are Few-Shot Learners.
In: Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS); Vancouver, Canada, 2020.

Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; et al.:
Palm: Scaling Language Modeling with Pathways.
Working Paper, 2023.

Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M.:
Retrieval Augmented Language Model Pre-training.
In: Proceedings of the 37th International Conference on Machine Learning (ICML); Vienna, Austria, 2020.

Kadavath, S.; Conerly, T.; Askell, A.; Henighan, T.; Drain, D.; et al.:
Language Models (Mostly) Know What They Know.
Working Paper, 2023.

Kaggle:
World Cities Database, https://www.kaggle.com/datasets/max-mind/world-cities-database?resource=download, 2017.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; et al.:
Webgpt: Browser-Assisted Question-Answering with Human Feedback.
Working Paper, 2023.

OpenML:
IMDB Movies Dataset, https://www.openml.org/search?type=data&status=active&id=43603&sort=runs, 2022.