Can LLMs substitute SQL? Comparing Resource Utilization of Querying LLMs versus Traditional Relational Databases

Xiang Zhang[1], Khatoon Khedri[2], and Reza Rawassizadeh[1]

[1]Metropolitan College, Department of Computer Science, Boston University
[2]Independent Scientist
Email ids: [1]xz0224@bu.edu, [2]khatoon.khedri1985@gmail.com, [1,3]rezar@bu.edu

Abstract

Large Language Models (LLMs) can automate or substitute different types of tasks in the software engineering process. This study evaluates the resource utilization and accuracy of LLM in interpreting and executing natural language queries against traditional SQL within relational database management systems. We empirically examine the resource utilization and accuracy of nine LLMs varying from 7 to 34 Billion parameters, including Llama2 7B, Llama2 13B, Mistral, Mixtral, Optimus-7B, SUS-chat-34B, platypus-yi-34b, NeuralHermes-2.5-Mistral-7B and Starling-LM-7B-alpha, using a small transaction dataset. Our findings indicate that using LLMs for database queries incurs significant energy overhead (even small and quantized models), making it an environmentally unfriendly approach. Therefore, we advise against replacing relational databases with LLMs due to their substantial resource utilization.

1 Introduction

The advent of Large Language Models (LLMs) has revolutionized several scientific and engineering disciplines, including software development tasks. Many software development related tasks could be done or automatized by LLMs. The satisfactory performance of LLM in search and query led to the introduction of specific LLM databases such as Vector database (Zhang et al., 2023) auxiliary knowledge information retrieval methods, a.k.a., Retrieval Augmented Generation (Shao et al., 2023).

Relational databases are one of the oldest and most common components of software applications. These databases manage structured data using interconnected tables in tabular form. Structured Query Language (SQL) is the query language used to interact with relational databases.

There are two widely known and significant limitations of using LLMs: (i) factual mistakes and hallucinations caused by neural networks (Tian et al., 2023), and (ii) token size limitations (Hoffmann et al., 2022), which do not allow them to load a large dataset into their prompt, and thus have a limited data size. There are ongoing efforts to prove that the factuality and coverage of LLMs are quickly improving with new training architectures and the increasing amount of text used as input (Elazar et al., 2021; Tam et al., 2022). Besides, there are continuous efforts to increase or remove the token size limitation, such as using Structured state space models (S4), e.g., Mamba (Gu and Dao, 2023) instead of Transformers.

Our work does not quantify or tackle any of these two known challenges. It focuses on benchmarking resource utilization using LLM instead of traditional SQL. In this research, we intend to investigate whether LLMs could replace traditional database management systems to search and query tabular data. Assuming even though the capability to generate SQL queries exists in LLMs, we should measure resource consumption and how accurately it identifies the correct answers from tabular datasets.

An essential consideration in our exploration is the environmental impact of LLMs. There are ongoing discussions [1][2][3][4] on the huge electricity and water cooling supply, underscoring sustainability-related challenges brought about by the new existence and overall being of the LLMs. Our results testify that even using a small-size trained LLM still consumes a high amount of

[1]https://www.theatlantic.com/technology/archive/2024/03/ai-water-climate-microsoft/677602

[2]https://www.oregonlive.com/silicon-forest/2022/12/googles-water-use-is-soaring-in-the-dalles-records-show-with-two-more-data-centers-to-come.html

[3]https://www.bloomberg.com/news/articles/2023-07-26/thames-water-considers-restricting-flow-to-london-data-centers

[4]https://www.washingtonpost.com/business/2024/03/07/ai-data-centers-power

energy in comparison to a native SQL engine running on a relational database. Besides, we have observed the inferior accuracy of LLMs in comparison to SQL engines. However, larger models might resolve the accuracy problem in the near future, but the energy issue remains open.

2 Literature review

There are recent reports on the water and electricity consumption of Generative Artificial Intelligence (AI) models, especially LLMs. However, their approach is mostly holistic and does not provide a comparative analysis of doing a particular task with LLM and without LLM (Dodge et al., 2022; de Vries, 2023; Luccioni et al., 2023; Li et al., 2023). On the other hand, interest in adopting LLMs for general tasks like database querying has grown in the natural language processing community; there are several promising works in this direction, which we have categorized into two main groups. One group of work passes the query in natural language and data into an LLM and, as a result, gets the SQL query back. These works are known as *Text-to-SQL* (Xu et al., 2019; Tang et al., 2021; Wang et al., 2019; Baig et al., 2022; Ferreira et al., 2020). The latter group (Rawassizadeh and Rong, 2023; Deutch et al., 2017) provides the data and the query in natural language as input into an LLM. Then, they get the result in natural language as well, we call them *NLQuery-to-NLAnswer*. In this section, we briefly describe each group of work.

2.1 Text-to-SQL approaches

Text-to-SQL approaches focus on transforming natural language queries into structured SQL commands, enabling users to interact with databases without needing SQL knowledge. The introduction of Google's SQL-PaLM model (Sun et al., 2023) marks a pivotal development in natural language to SQL translation. SQL-PaLM model efficiently refines LLMs to understand the natural language query and convert it into SQL commands.

Baig et al. (2022) reviewed existing frameworks for processing natural language to SQL queries. The use of the attention mechanisms in neural networks for natural language interfaces to databases (NLIDB) was evaluated by Ferreira et al. (2020). Wang et al. (2019) proposed the RAT-SQL framework, based on the relation-aware self-attention mechanism, to address schema encoding, schema linking, and feature representation within a Text-to-SQL encoder. RAT-SQL modeled the database schema as a directed graph. NADAQ (Xu et al., 2019) merged specialized encoder-decoder architecture with traditional database parsing techniques for querying databases using natural language.

2.2 NLQuery-to-NLAnswer approaches

Recently, Rawassizadeh and Rong (2023) proposed ODSearch, which retrieves data from wearable and mobile devices through natural language processing. It employs data compression and Bloom filters to enable real-time responses to natural language queries.

Deutch et al. (2017) presented a system that extends the generation of natural language interfaces to databases by generation of the natural language answer. It operates based on the provenance of the query result tuples. The provenance information is converted into natural language by structuring the originating query such that the user is delivered an informative response. Dries et al. (2009) also suggested a data model and query language designed specifically for network analysis in their research on a Query Language for Analysis Networks.

These works foster an interactive and less scripted interaction of a database system with the users. With those considerations, both Text-to-SQL and NLQuery-to-NLAnswer approaches highlight the importance of studying the resource usage of these systems. To our knowledge, except for ODSearch (Rawassizadeh and Rong, 2023), which does not use an LLM, none of the other works investigate the resource utilization of queries.

2.3 Energy consumption of LLM

Recently, the environmental impact of artificial intelligence has garnered significant attention from the research community, especially on water and electricity usage.

Large Language Models such as GPT-3 require substantial computational power for training, leading to significant Execution Energy Consumption and associated carbon emissions[5]. Dodge et al. (2022) present a method for calculating the carbon footprint of AI operations in the cloud, focusing on the energy consumption and CO2 emissions of machine learning models. The

[5]https://projectmanagers.net/top-10-disadvantages-of-large-language-models-llm

research highlights the importance of geographic location in selecting cloud instances to minimize carbon intensity. Luccioni et al. (2023) conducted a systematic comparison of the energy and carbon costs associated with deploying various machine learning models. It reveals that multi-purpose, generative AI models, such as those used in LLM, are significantly more resource-intensive than task-specific models, even when accounting for model size. Their study calls for more intentional consideration of energy and emissions costs in the deployment of AI tools. de Vries (2023) explores AI's electricity use, considering both pessimistic and optimistic scenarios for global data center electricity consumption, and emphasizes the need for cautious adoption of AI technologies and understanding their energy implications. In addition to studies focused on energy utilization, Li et al. (2023) examine the often-overlooked water footprint of AI corporations, particularly the substantial freshwater consumption by LLM models like GPT-3 during training in data centers. They estimate that global AI demand could lead to significant water withdrawal by 2027, emphasizing the urgency of addressing AI's water use.

The most related works to ours are proposed by Tang et al. (2021). They use machine learning to estimate SQL queries' CPU and memory demands, broadening evaluation beyond accuracy to include resource consumption, which is crucial for assessing LLMs' efficiency in database queries.

3 Methodology

In this work, we evaluate nine open-source LLMs that operate as NLQuery-to-NLAnswer. In particular, we measure their accuracy and resource utilization compared to SQL queries. Our study assesses how effectively LLMs are generating not only SQL but also direct answers from natural language queries. As an SQL engine, we choose to use (SQLite) [6], which is a common SQL engine used in devices that have resource constraints, such as Android phones. There are promising tools available to measure the resource utilization of LLMs (Samsi et al., 2023; MLE). However, we have used our scripts to have enough flexibility to measure different resources[7].

3.1 Test Dataset

The dataset is a synthetic representation of stock transactions in a real-world scenario built by SQLite[8]. SQLite's efficiency and minimal resource requirements make it suitable for scenarios where computational resources are limited, such as on battery-powered devices (Rawassizadeh and Rong, 2023). The synthetic dataset we built comprises 100 records across five stock symbols, such as AAPL, GOOGL, AMZN, MSFT, and TSLA, with the transaction type being BUY or SELL. Date of transactions, type, stock symbol, amount, and cost data attributes used in our queries. The transaction date was extracted along with its time from a series that spanned over a range of dates for seven consecutive days. Due to the small size of the test dataset, we do not encounter the token size limitation issue of LLM.

Amounts and costs were randomized using *random* library to create a more realistic and diverse dataset [9]. Instead of structuring our dataset with a schema, we directly feed 100 records into our framework. This decision reflected the more dynamic, real-world conditions under which non-expert users might interact with databases. Based on the foundational concepts presented in *Fundamentals of Database Systems* (Elmasri and Navathe, 2016), the following are ten SQL queries designed to assess querying capabilities. These queries utilize `COUNT`, `SUM`, `MAX`, and `AVG`, apply condition filtering using `WHERE`, and implement grouping with `GROUP BY`.

A. Count transactions per stock symbol.

B. Total quantity sold per symbol.

C. Total revenue from sales.

D. Maximum sale price per symbol.

E. Average purchase price per symbol.

F. Several unique stock symbols.

G. Quantities bought and sold per symbol.

H. Total investment in buy transactions.

I. The transaction quantity is on a specific date (2023-9-23).

[6]https://sqlite.org/index.html

[7]https://github.com/XiangZhang-zx/LLM-StockQuery-Dataset/blob/main/LLM_Generation_Comparison.ipynb

[8]https://github.com/XiangZhang-zx/LLM-StockQuery-Dataset/blob/main/dataset.csv

[9]https://docs.python.org/3/library/random.html

J. The highest transaction price for a stock on a specific date (google, 2023-9-24).

3.2 Example Prompt Template and Generated SQL

Listing1 is a portion of the prompt template used, along with an example query and the corresponding SQL script. The full dataset contains 100 records.

Listing 1: Prompt Template and Generated SQL

```
<s>[INST]
Date         Transaction   Symbol   Quantity   Price
2023-09-23   BUY           AMZN     99         2089
2023-09-24   BUY           MSFT     84         67
2023-09-25   SELL          AAPL     27         684
...
(100 records in total)
...
Give me the SQL script to count the number of
     transactions for each stock symbol.
[/INST] </s>

Generated SQL Script:
SELECT Symbol, COUNT(*) as Transaction_Count FROM
     stocks GROUP BY Symbol;
```

3.3 Experimental LLMs

As shown in Table 1, in our evaluation, we specifically chose a selection of large language models (LLMs), including Llama2 (7B and 13B versions), Mistral, and Mixtral, Optimus-7B, SUS-chat-34B, platypus-yi-34b, NeuralHermes-2.5-Mistral-7B, and Starling-LM-7B-alpha. These models were chosen based on ranking at the top of the Huggingface open LLM leaderboard (back in late 2023), and also our infrastructure can execute them. The traditional transformer stack was already designed to adapt them in terms of performance and efficiency. For Llama2 (7B and 13B), SUS-chat-34B, and platypus-yi-34b, we adhere to the traditional transformer stack. For Mistral, Mixtral, Optimus-7B, NeuralHermes-2.5-Mistral-7B, and Starling-LM-7B-alpha, we adhere to the pipeline produced by Hugging Face, tuned to a quantized 4-bit configuration.

3.4 Experiment Setup

Our hardware infrastructure includes two NVidia RTX 4090 GPU 24GB, with 256 GB RAM and 3.30 GHz Intel Core i9 CPU. The operating system is Ubuntu 20.04 LTS, and we used CUDA Version 12.0 for GPU computations.

To evaluate the performance of the LLM, we implemented a custom Python function that automates the process of measuring the time, CPU, and memory usage of the model. The function records these metrics before and after the model generates responses on natural language input using the *tracemalloc* and *time* libraries[10][11]. Then, our function calculates the differences between the start and end values of the metrics and reports the execution time, CPU utilization, and memory consumption. To quantify energy consumption per process, the *Turbostat* utility was employed to monitor the *pkgwatt* (package power)[12]. This package, combined with the execution time, was used to calculate the model's energy consumption in Joule (J).

In our experiments we use two pipelines, the Transformers Pipeline allows explicitly setting text generation performance and relevance with *torch* library, combined with options control on temperature, max_new_tokens, as well as repetition_penalty values[13]. The Hugging Face Pipeline contains quantized models to reduce

Table 1: Comparison of Large Language Models by Parameters and Configuration

Model	Parameters	Configuration
Llama2 7B	7 Billion	Traditional Transformer
Llama2 13B	13 Billion	Traditional Transformer
Mistral	7 Billion	Hugging Face Pipeline, 4-bit Quantized
Mixtral	7 Billion	Hugging Face Pipeline, 4-bit Quantized
Optimus-7B	7 Billion	Hugging Face Pipeline, 4-bit Quantized
SUS-chat-34B	34 Billion	Traditional Transformer
platypus-yi-34b	34 Billion	Traditional Transformer
NeuralHermes-2.5-Mistral-7B	7 Billion	Hugging Face Pipeline, 4-bit Quantized
Starling-LM-7B-alpha	7 Billion	Hugging Face Pipeline, 4-bit Quantized

[10]https://docs.python.org/3/library/tracemalloc.html

[11]https://docs.python.org/3/library/time.html

[12]https://www.linux.org/docs/man8/turbostat.html

[13]https://pytorch.org/docs/stable/index.html

resource consumption using different options impacting output response sharpness and speed, such as max_new_tokens, top_k, and eos_token_id values.

4 Experimental Evaluation

We examine the resource usage of SQL engine compared to LLMS to query tabular data, the proficiency of LLMs in generating SQL-equivalent queries from natural language, and their effectiveness in obtaining semantically accurate responses from structured datasets.

To establish a baseline for the evaluation of LLMs, we measure both the execution time and memory consumption for queries (A-J) associated with direct SQL query execution. Based on our measurement of the direct SQL query execution on the SQL engine, the average execution time is *0.41 ms*, and the average memory usage is *1641 B.* As we have described earlier, the SQL engine we used is SQLite.

The average execution time and memory utilization for direct query results and query generation of LLM models are presented in Tables 2 and 4. Moreover, we display the accuracy of direct query results by LLM models in Table 3 and the overall accuracy of them in Table 5.

In the results shown in Tables 3 and Tables 5, symbols used are ✓ for correct generation and ✗ for incorrect or incomplete generation.

4.1 Natural Language Query Performance Analysis

We present the results of our comparison by focusing on different aspects of the models, including execution time and accuracy. As shown in Table 2, the average execution time varied significantly across the models, from as quick as 23 seconds for Mistral to 260 seconds for SUS-chat-34B. It indicates that the size and architecture of the models have a significant impact on the execution of the tasks. SUS-chat-34B also showed the highest memory usage in the transformer pipeline, highlighting the scalability concerns of using large and complex models for natural language processing tasks. Notably, in the Hugging Face pipeline, models like Optimus-7B demonstrated efficiency with minimal memory increase, proving that using quantization techniques can reduce the resource consumption of the models. Our results suggest that larger LLMs can achieve higher accuracy for natural language processing tasks but also pose challenges in terms of execution time and resource utilization.

According to Table 2, Llama2 7B was the most resource-efficient model across the tasks, with reasonable execution times and resource usage. SUS-chat-34B, on the other hand, had high resource consumption, raising questions about its practicality in larger datasets. Optimus-7B, which employs quantization techniques to reduce model size and complexity, comes closest to achieving the execution time and resource efficiency of SQLite.

In Table 3, platypus-yi-34b accurately interpreted straightforward queries, such as identifying the total number of unique stock symbols. However, models often predict or complete questions rather than providing the requested information, highlighting a propensity for these models to engage in dialogue rather than execute database queries accurately. Regarding inconsistencies, Llama2 7B and Llama2 13B sometimes generated irrelevant responses, indicating a need for improved training focused on database querying capabilities.

Table 2: Average execution time and memory utilization of direct query results of LLM models

Model	Execution Time (s)	Memory Usage (kB)
Llama2 7B	60	64
Llama2 13B	106	70
SUS-Chat-34B	260	63
platypus-yi-34b	235	70
Mistral	23	301
NeuralHermes-2.5-Mistral-7B	78	464
Optimus-7B	33	247
Starling-LM-7B-alpha	41	263
Mixtral	116	571

4.2 SQL Query Generation Results

We evaluated listed LLMs for generating SQL queries from natural language inputs, and Table 4 displays the average execution time and memory utilization of SQL query generation using our

Table 3: Accuracy of direct Query Results of LLM Models (acc. refers to accuracy)

Model	A	B	C	D	E	F	G	H	I	J	acc.
Llama2 7B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
Llama2 13B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
SUS-Chat 34B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
platypus-yi-34b	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	10%
Mistral	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
Neural Hermes 2.5-Mistral 7B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
Optimus 7B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
Starling LM 7B-alpha	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%
Mixtral	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	0%

experimental LLMs.

Llama2 7B, Llama2 13B, and Mistral 7B showed mixed results in translating natural language to SQL, ranging from partially accurate to essentially reiterating the initial query. Another important observation from the experiments was that most of the models, including Mistral 7B, SUS-Chat-34B, platypus-yi-34b, Optimus-7B, and Starling-LM-7B-alpha, failed to include the condition of transaction is equal to SELL or BUY in their SQL queries. Table 4 and table 5 show that in the transformer pipeline, while SUS-chat-34B and platypus-yi-34b demonstrated high success in correct script generation, but their high resource consumption is a challenge. Conversely, within the Hugging Face pipeline, Optimus-7B and Starling-LM-7B-alpha achieved accurate SQL generation with lower resources.

Table 5 shows meaningful variability in model performance, with some models excelling in accuracy while others struggled with resource utilization and generating precise SQL queries.

4.3 Energy Utilization

Figures 1 and 2 present the average energy utilization for direct SQL query execution along LLM models. We can observe that SQL engine consumes the least energy, quantified at 8.22×10^{-6}J. In the assessment of LLM models for both direct query execution and SQL query generation, Platypus-yi-34b was identified as the most energy-intensive, recording energy utilization of 2181.8J and 734.2J, respectively. In contrast, Optimus-7B exhibited the lowest energy consumption for direct query execution at 0.163J, while Mistral registered the lowest for SQL query generation, consuming 0.234J. Therefore, we can conclude that the larger the model, the more utilized energy is used to run a query.

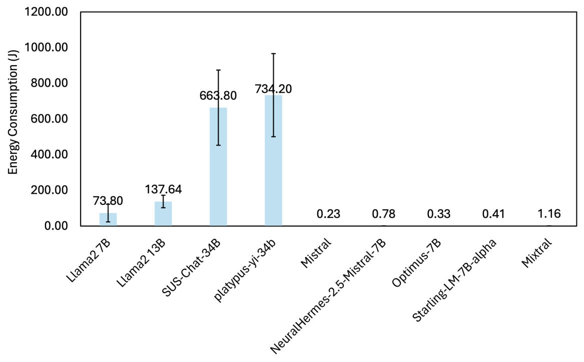

Figure 1: The average energy consumption (J) for direct query results of LLM models

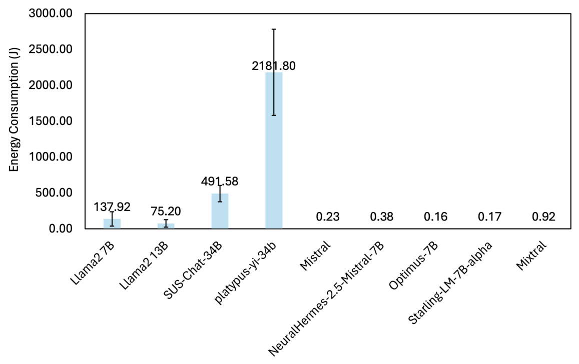

Figure 2: The average energy consumption (J) for SQL query generation of LLM models

5 Discussion

Direct query results from LLM models show disappointingly low accuracy. These findings highlight a significant challenge: LLMs struggle to query databases effectively without additional engineering. Specifically in generating SQL queries, Models often misinterpreted complex requests, incorrectly applying SQL clauses.

Our findings also point out that energy efficiency varies among LLM models used for SQL query generation, with larger models consuming more energy. Quantized models, such as Optimus-7B, performed well in the execution time and

Table 4: Average execution time and memory utilization of SQL query generation using LLM models

Model	Execution Time (s)	Memory Usage (kB)
Llama2 7B	106	70
Llama2 13B	61	55
Mistral	23	232
SUS-Chat-34B	200	57
platypus-yi-34b	597	93
NeuralHermes-2.5-Mistral-7B	38	266
Optimus-7B	16	206
Starling-LM-7B-alpha	17	204
Mixtral	92	488

Table 5: Detailed Accuracy of Query Generation (acc. refers to accuracy)

Model	A	B	C	D	E	F	G	H	I	J	acc.
Llama2 7B	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	10%
Llama2 13B	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	20%
SUS-Chat 34B	✓	✓	✓	✗	✓	✓	✗	✗	✓	✓	70%
platypus-yi-34b	✓	✓	✗	✗	✗	✓	✓	✗	✗	✓	50%
Mistral	✗	✓	✗	✗	✗	✓	✗	✗	✓	✓	40%
Neural Hermes 2.5-Mistral 7B	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	30%
Optimus 7B	✓	✓	✗	✗	✗	✓	✓	✗	✓	✗	50%
Starling LM 7B-alpha	✓	✓	✓	✗	✗	✓	✓	✗	✗	✓	60%
Mixtral	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓	60%

resource use, but limitations in scalability and token size question their efficacy on larger datasets. Nonetheless, LLMs could enhance database management system (DBMS) querying alongside traditional methods, improving accessibility for non-experts. Further research should aim at hybrid methodologies that combine LLM capabilities with traditional SQL parsing technologies.

6 Conclusion and Future Work

LLMs offer a radically new perspective on database querying and the nature of computational systems. In this work, we measure the accuracy and resource utilization of nine small open-source LLMs in querying tabular data. Our results present the significant resource expense of employing LLMs, even small models that are highly compressed with quantization. Besides, the accuracy of using LLM (at least not the very large and commercialized ones) for querying tabular data is low. As the model gets larger, the accuracy improves, but we did not experiment with larger models. Potential further research can investigate fine-tuning existing models with SQL schema, toward reducing the misinterpretations made by the LLM models in querying databases.

References

Ml energy. https://ml.energy. [Last accessed on April 11, 2024].

Mirza Shahzad Baig, Ali Imran, Abdul Usman Yasin, Arslan Haider Butt, and Muhammad Imran Khan. 2022. Natural language to sql queries: A review. *International Journal of Innovations in Science Technology*, 4:147–162.

Alex de Vries. 2023. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194.

Daniel Deutch, Nerya Frost, and Amir Gilad. 2017. Provenance for natural language queries. *Proc. VLDB Endow.*, 10:577–588.

Jesse Dodge, Tess Prewitt, Remi Tachet des Combes, Emily Odmark, Roy Schwartz, Emma Strubell, et al. 2022. Measuring the carbon intensity of ai in cloud instances. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1877–1894.

Anton Dries, Siegfried Nijssen, and Luc De Raedt. 2009. A query language for analyzing networks. In *Proceedings of the 18th ACM conference on Information and Knowledge Management*, pages 485–494. ACM.

Yoav Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pre-trained language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031.

Ramez Elmasri and Shamkant B. Navathe. 2016. *Fundamentals of Database Systems*, 7 edition. Addison-Wesley.

Sara Ferreira, Gonçalo Leitão, Igor Silva, Anabela Martins, and Piero Ferrari. 2020. Evaluating human-machine translation with attention mechanisms for industry 4.0 environment sql-based systems. In *2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT*, pages 229–234. IEEE.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time Sequence Modeling with Selective State Spaces. *arXiv preprint arXiv:2312.00752*.

Josh Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Tingfeng Cai, Eliza Rutherford, et al. 2022. Training compute-optimal large language models. *ArXiv*, abs/2203.15556.

Peifeng Li, Jie Yang, Md Amirul Islam, and Suzhen Ren. 2023. Making ai less" thirsty": Uncovering and addressing the secret water footprint of ai models. *arXiv preprint arXiv:2304.03271*.

Alexandra Sasha Luccioni, Yacine Jernite, and Emma Strubell. 2023. Power hungry processing: Watts driving the cost of ai deployment? *arXiv preprint arXiv:2311.16863*.

Reza Rawassizadeh and Yu Rong. 2023. Odsearch: Fast and resource efficient on-device natural language search for fitness trackers' data. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(4):1–25.

Siddharth Samsi, Dongfang Zhao, John McDonald, Bo Li, Antonio Michaleas, Michael Jones, et al. 2023. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE.

Zhezheng Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.

Ruiqi Sun, Sercan O. Arik, Hootan Nakhost, Hang Dai, Rishabh Sinha, Peng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language model adaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

Daniel Tam, Sachin Mascarenhas, Sheng Zhang, Stephen Kwan, Mohit Bansal, and Colin Raffel. 2022. Evaluating the factual consistency of large language models through summarization. *arXiv preprint arXiv:2211.08412*.

Chuan Tang, Bo Wang, Zhenxiao Luo, Huaxin Wu, Sanket Dasan, Min Fu, and Pranav Mishra. 2021. Forecasting sql query cost at twitter. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 154–160. IEEE.

Kevin Tian, Eric Mitchell, Huang Yao, Christopher Manning, and Chelsea Finn. 2023. Fine-tuning language models for factuality. *ArXiv*, abs/2311.08401.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Bin Xu, Ruijiang Cai, Zijian Zhang, Xiaochun Yang, Zhifeng Hao, Zhenhui Li, and Zhiqiang Liang. 2019. Nadaq: Natural language database querying based on deep learning. *IEEE Access*, 7:35012–35017.

Yanzhao Zhang, Zhiwei Yu, Wei Jiang, Yelong Shen, and Jingjing Li. 2023. Long-term memory for large language models through topic-based vector database. In *2023 International Conference on Asian Language Processing (IALP)*, pages 258–264. IEEE.