OuterTuning: how are database tuning suggestions in LLMs?

Lucas Ferraz Massena[1], **Ana Carolina Almeida**[1]

[1]Department of Computer Science - State University of Rio de Janeiro (UERJ)

lucasfermas2701@gmail.com, ana.almeida@ime.uerj.br

Abstract. *Large Language Models (LLMs) have established themselves as promising tools for supporting database tuning. This work extends OuterTuning to incorporate LLMs, providing additional tuning suggestions for the DBA. These suggestions are accompanied by explanations provided by the LLMs themselves. Through experimental scenarios using the TPC-H benchmark, it is verified that LLMs are beneficial, but still have room for improvement.*

1. Introduction

Large-scale language models (LLMs) are being widely used to solve complex tasks. Database tuning is considered a complex task [Bruno 2011].

The OuterTuning framework was developed to assist database administrators (DBAs) in the task of database tuning [Almeida et al. 2018]. The main objective is to provide transparency in tuning suggestions and various alternatives for database adjustments (not just the best ones - the origin of **Outer**Tuning).

The objective of this work is to extend OuterTuning, an alternative tuning suggestion for the DBA. The quality of the tuning actions is out of the scope of this paper. Our experiments show that LLMs suggest solutions that may be good and functional, but that are not always optimal or used by database optimizers.

The contributions of this paper are: (i) the implementation of the OuterTuning framework extension with LLMs; (ii) the extension of OnDBTuning with new heuristics; (iii) using the DBMS's own optimizer to evaluate the tuning suggestion in real time and without impacting the DBMS; (iv) the discussion on the comparison of index suggestions from 3 main widely used LLMs: OpenAI's GPT, Anthropic's Claude, and Google's Gemini.

2. Background and Related Work

Large language models (LLMs) are Transformer language models trained with massive amounts of textual data, and with an enormous capacity to understand natural language and solve complex tasks [Shanahan 2024, Zhao et al. 2023].

Performing database tuning is considered a complex task. **Database tuning** involves adjusting many knobs and data structures to decrease response time or increase database system throughput [Shasha and Bonnet 2002]. Each change in one knob or structure may affect another structure or knob.

Database tuning tools use heuristics to suggest adjustments to knobs, full indexes, partial indexes, materialized views, and partitioning. Some tools are already incorporating

Table 1. Comparison of related work

Tool	Tuning target	Knowledge	DBMS support	LLM/ML
OtterTune	knobs	Runtime metrics, History of previous tuning sessions	PostgreSQL MySQL Oracle	ML
GPTuner	knobs	Database Manual, Web forums	PostgreSQL MySQL	GPT-4
λ-Tune	Knobs Indexes	Database Manual	PostgreSQL MySQL	GPT-4
DB-BERT	Knobs	Database Manual	PostgreSQL MySQL	BERT
D-Bot	Knobs DB anomalies	Diagnosis documents, DBMS Metrics	PostgreSQL	GPT-4 GPT-3.5 Baichuan2 CodeLlama
OuterTuning	Indexes MVs	Ontology, Inference Rules, Real Time Workload	PostgreSQL MySQL Oracle	GPT-4 gemini-2.5-flash claude-sonnet-4.6

the use of LLMs in an effort to improve these suggestions or provide another helpful alternative for the DBA (Database Administrator).

Table 1 provides a summary of some database tuning tools. OtterTune [Zhang et al. 2018, Van Aken et al. 2021], GPTuner [Lao et al. 2025], λ-Tune [Giannakouris and Trummer 2025], DB-BERT [Trummer 2022], and D-Bot [Zhou et al. 2024] offer suggestions for adjusting knobs for the database. Most tools have knowledge sourced from the DBMS manufacturer. All tools support one of the world's most popular DBMSs (PostgreSQL), and most support more than one DBMS. Only OtterTune does not directly use an LLM, but rather the machine learning (ML) technique. D-Bot presents the anomalies found in the database in addition to tuning suggestions. Only two tools offer more than one tuning suggestion option for the DBA: λ-Tune (knobs and indexes) and OuterTuning (indexes and materialized views (MVs)).

The knowledge used by the OuterTuning stands out for being a domain ontology. **Domain ontology** describes the classes of concepts and the relationships between these concepts that define a specific domain, that is, an application area [Guarino 1998]. In addition, OuterTuning's domain ontology, called **OnDBTuning** [Almeida et al. 2019, Perciliano et al. 2021], has inference rules defined within it. The rules correspond to existing heuristics of database tuning, published in the literature. It is important to highlight that other tools exist, but all of them use LLMs and also suggest Knobs, and they do not use ontology. No commercial tools were researched.

3. OuterTuning

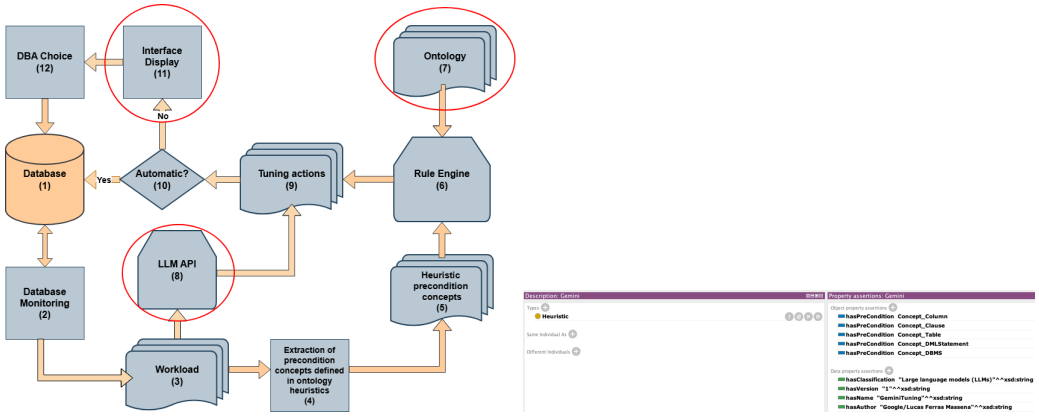
The OuterTuning framework was designed by [Almeida 2013] to provide more transparency in the database tuning process. Tuning heuristics are defined through inference rules in the ontology using concepts known to the DBA domain. This allows the DBA to understand existing heuristics as well as include new ones. Furthermore, the framework

aims to always provide justifications for tuning suggestions, regardless of the heuristic used.

3.1. Overview and extensions

Figure 1a shows the flow followed by OuterTuning to suggest tuning actions to the DBA. The parts extended by this work to accommodate the LLMs are circled in the figure.

Firstly, the framework monitors the database (2) and captures all live statements submitted to the database (1) from the users - the workload (3). The workload is used (4) to instantiate the concepts needed for each heuristic to be executed (5). The concepts and inference rules are retrieved from the ontology (7) and submitted already instantiated to the rule engine (6). The rule engine infers tuning actions (9) according to the ontology's heuristics. The workload is also submitted to the LLM APIs (8). The LLM APIs infer tuning actions (9) too. If the automatic mode (10) is active (This choice is made when the tool is launched), the actions are executed in real time in the database. Otherwise, the actions are displayed in the interface (11), so the DBA (12) can choose which actions will be executed.



(a) OuterTuning Steps (b) OnDBTuning Extension

Figure 1. OuterTuning Steps and OnDBTuning (OuterTuning Ontology) Extension

For now, OnDBTuning has been extended to include the new tuning suggestions with the LLMs (Figure 1b). To do this, new heuristic instances were created for each of the LLMs with their respective data properties: hasClassification, hasName, hasAuthor, and hasVersion. In addition, object properties corresponding to the concepts that need to be instantiated so that the LLMs can infer the tuning suggestions were defined, i.e., the workload and the DBMS to be analyzed: DMLStatement, DBMS, Clause, Column, Table.

Three agents were created in OuterTuning to include LLMs. The agents are independent and each is using the same prompt. The prompt has 5 sections:

1. **Model Role:** Provide the context and objective for the LLMs. The DBMS model is given dynamically (from metadata), so the model can use specific knowledge of the database used.
2. **Database Schema:** Give the model, context, tables, cardinality, number of lines, and columns list. This is essential so the model can identify which columns are

available to make the index, avoid index suggestions in columns that don't exist, and the table's size is a parameter to know if the index makes sense.

3. **Index Guidelines:** Instruct the model to use the following rules:
   (a) The model needs to know that the system is using HypoPG[1], so the model has to generate commands compatible with the HypoPG function.
   (b) Instructs the model to always create a suggestion for each query.
   (c) Prohibit the model to not create index that are already created in the schema.
   (d) Prohibit the ";" at the end of the command; this is a syntax restriction to fit in the `hypopg_create_index` function.
4. **Workload:** The models receive the same informations which a DBA would use to analyse the query. It gives, for each query, dynamically the number of executions, cost, and duration. It gives the execution plan obtained by "explain". It refers to the operations realized, such as seq scan and hash join, indicating that the index could help.
5. **Output format:** Controls the output format. The model is instructed that have to create only one suggestion for each query. Additionally, the answer has to be pure JSON, with no markdown or text, because the code parses it directly with `JsonParser.parse(text)`, so anything but JSON would cause problems.

Prompt Example: You are a PostgreSQL database tuning expert. Analyze the workload and schema below and suggest one index recommendation for each SQL statement. <Database Schema >The cost estimator uses HypoPG (hypothetical indexes). (1) Suggest one index for every SQL statement in the workload; (2) If no index is expected to improve performance, still return a JSON object with expectedBonus = 0 and explain the reason; (3) Prefer covering indexes by using the INCLUDE clause whenever it can enable Index-Only Scans; (4) Do not suggest indexes that already exist in the schema; (5) Do not terminate the CREATE INDEX command with a semicolon.

The OuterTuning interface pattern was maintained, with the same information, adding more tuning alternatives with LLMs (Figure 2a). The new screen created was for comparing LLMs (Figure 4) in cases where, in the future, discrepancies may occur regarding tuning suggestions when diversifying the data structures requested in the LLM prompt.

3.2. Experiments

The experiments were run on a virtual machine with the following configuration: Ubuntu (64-bit), 7 virtual CPUs, 13.37GB RAM, 61.05GB disk storage. The code for the extended OuterTuning is available at: https://github.com/Lucas-massena/outer-tuning.

The framework was run four times and limited to evaluating only the first ten queries received by the TPC-H benchmark due to cost considerations, totaling **40 observations**. Although the tool numbers the queries from Q1 to Q10, these numbers do not correspond to the benchmark numbers. The benchmark query numbers appear within the query text itself as comments, allowing for identification. The queries were submitted to the PostgreSQL database version 14, using the Docker container also available on the

[1]https://www.postgresql.org/about/news/introducing-hypopg-hypothetical-indexes-for-postgresql-1593/

Table 2. Experiment Results

LLM	Original Cost	Indexes Cost	Aggregate Reduction	Average Reduction	Median Reduction	Wins	Draws
Gemini	434.521	319.382	**26,50%**	33,86%	11,13%	**22/40**	18
GPT	434.521	402.893	7,28%	10,41%	0,00%	11/40	29
Claude	434.521	324.551	25,31%	**29,10%**	0,01%	21/40	19

same OuterTuning GitHub repository, simulating TCP-H queries. The LLM APIs used were: GPT-4, gemini-2.5-flash, and claude-sonnet-4.6.

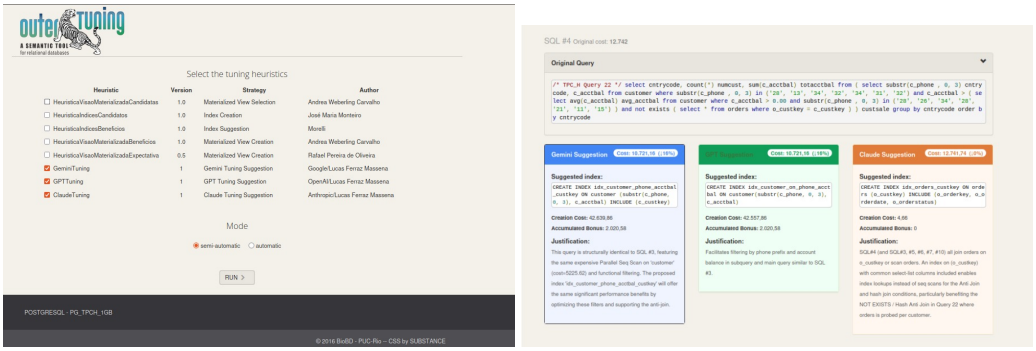
In all executions, the 3 LLMs with the task of recommending indexes and the semi-automatic mode of OuterTuning were selected, as shown in Figure 2a. The graphs presented in Figure 4 compare the costs estimated by the DBMS's own optimizer under the following conditions: (i) query executed on the schema without any index, including without a primary key; (ii) query executed after the hypothetical creation of the index suggested by the respective LLM. This value was obtained through the use of the HypoPG extension existing in the DBMS itself. This extension allows the simulation of the existence of the index without causing much impact on the database system. In addition, the evaluation is close to reality, as the DBMS's own optimizer is used. The smaller the vertical bar in the graph, the better the recommendation.

The 22 TPC-H queries were executed with varying frequency. Query 13 was the most recurrent (12 occurrences), followed by Q16 (6x) and Q5 (5x). In the description of each query, there is a comment indicating which query number it refers to in the TPC-H benchmark. The repetition of the same query within the same round is intentional to evaluate the variability of the LLM response.

Table 2 summarizes the results of the analysis of the graphs in Figure 4. Gemini leads in both aggregate cost reduction and number of wins. Claude performs similarly to Gemini has, in terms of aggregate reduction and number of wins, but has a practically zero median. From the graphs and the median value, it can be seen that Claude concentrates its gains in a few queries with very high cost reductions. GPT is considered the most conservative. In 29/40 observations (draws), even with the index, the cost was the same as the original. This scenario indicates that the index was not used by the optimizer. If the index were created, it would only occupy space in the database.

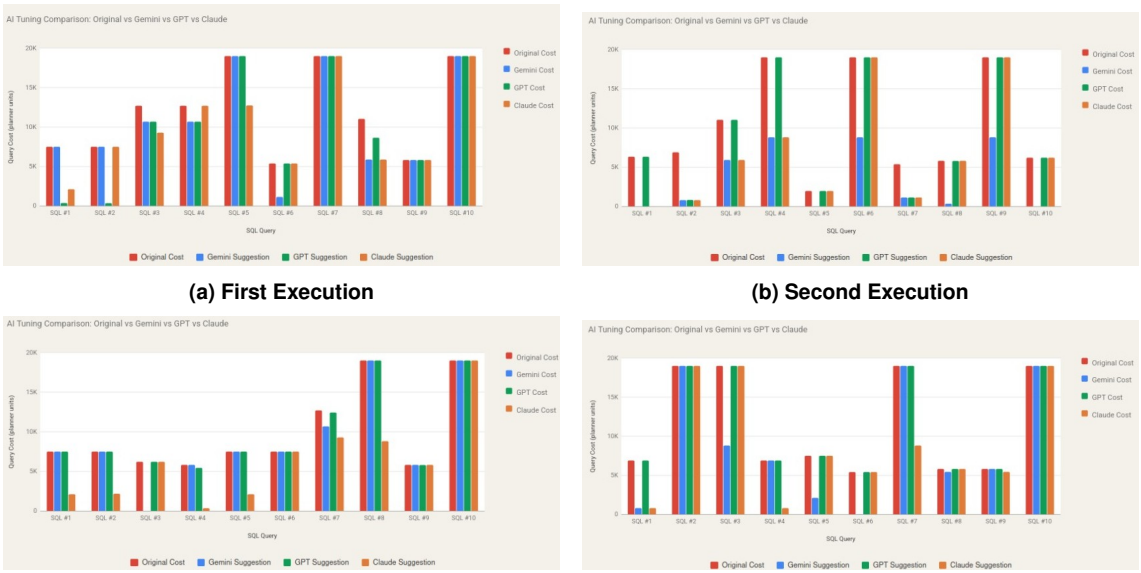
Analyzing the justifications of the LLMs, we noted that Gemini and GPT sometimes identified identical queries and referenced them in their justifications (Figure 2b). Claude, on the other hand, contextualized the workload more and sought to relate the queries more and interpret them as a more global tuning. For example, in Figure 2b, Claude mentions queries 4, 3, 5, 6, 7, and 10, which refer to TPC-H queries: 22, 22, 13, 3, 13, and 13, respectively. In the first three executions, Claude was the LLM that demonstrated a growing trend in the magnitude of cost reduction. As more queries were submitted within the same round, the average gain of the index recommended by Claude increased.

We noted that for queries with selective predicates on single tables (14, 17, and 18), Gemini wins or ties. The exception is query 6, where GPT wins outright. In queries with correlated subqueries and multiple joins (2, 7, 22, and 17), Claude performs better.



(a) OuterTuning Heuristics (b) LLM Rationale

Figure 3. OuterTuning Home Screen and Rationale for LLMs



(a) First Execution (b) Second Execution

(c) Third Execution (d) Fourth Execution

Figure 4. Four executions of the experiments

4. Conclusions

This work presented an extension of the OuterTuning framework using three LLM APIs. The experiments demonstrated that the DBA has more transparent alternatives at their disposal to make database tuning decisions. The experiments were limited to 10 queries and 4 repetitions due to cost considerations. Future work aims to expand the analysis of LLM tuning suggestions to other data structures (MVs, partial indexes), compare LLM suggestions with existing heuristics, and improve LLM knowledge using OnDBTuning.

Declaration on the use of generative AI. During the preparation of this work, the author(s) used Claude in order to improve the readability and language of the manuscript. After using this tool, the author(s) reviewed and edited the content as needed and take full responsibility for the content of the publication.

This work is derived from the undergraduate final project (TCC) of the first author, to be defended at the Department of Computer Science, State University of Rio de Janeiro, in November 2026.

References

Almeida, A. C., Campos, M. L. M., Baião, F., Lifschitz, S., de Oliveira, R. P., and Schwabe, D. (2019). An ontological perspective for database tuning heuristics. In *International Conference on Conceptual Modeling*, pages 240–254. Springer.

Almeida, A. C., Haeusler, E. H., Lifschitz, S., de Oliveira, R. P., and Schwabe, D. (2018). Outer-tuning: sintonia fina automática baseada em ontologia. In *SBBD Companion*, pages 29–34.

Almeida, A. C. B. d. (2013). *Outer-Tuning: Framework para apoiar a sintonia fina de banco de dados*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil. Orientador: Sergio Lifschitz.

Bruno, N. (2011). *Automated physical database design and tuning*. CRC Press.

Giannakouris, V. and Trummer, I. (2025). λ-tune: Harnessing large language models for automated database system tuning. *Proceedings of the ACM on Management of Data*, 3(1):1–26.

Guarino, N. (1998). *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, volume 46. IOS press.

Lao, J., Wang, Y., Li, Y., Wang, J., Zhang, Y., Cheng, Z., Chen, W., Tang, M., and Wang, J. (2025). Gptuner: An llm-based database tuning system. *ACM SIGMOD Record*, 54(1):101–110.

Perciliano, L. d. S. S., dos Santos, V., Baiao, F., Haeusler, E. H., Lifschitz, S., and Almeida, A. C. (2021). Inferencing relational database tuning actions with ondbtuning ontology. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 157–168. SBC.

Shanahan, M. (2024). Talking about large language models. *Communications of the ACM*, 67(2):68–79.

Shasha, D. and Bonnet, P. (2002). *Database Tuning: Principles, Experiments and Troubleshooting Techniques*, volume 33. Morgan Kaufmann, 1st edition.

Trummer, I. (2022). Db-bert: a database tuning tool that" reads the manual". In *Proceedings of the 2022 international conference on management of data*, pages 190–203.

Van Aken, D., Yang, D., Brillard, S., Fiorino, A., Zhang, B., Bilien, C., and Pavlo, A. (2021). An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. *Proceedings of the VLDB Endowment*, 14(7):1241–1253.

Zhang, B., Van Aken, D., Wang, J., Dai, T., Jiang, S., Lao, J., Sheng, S., Pavlo, A., and Gordon, G. J. (2018). A demonstration of the ottertune automatic database management system tuning service. *Proceedings of the VLDB Endowment*, 11(12):1910–1913.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2):1–124.

Zhou, X., Li, G., Sun, Z., Liu, Z., Chen, W., Wu, J., Liu, J., Feng, R., and Zeng, G. (2024). D-bot: Database diagnosis system using large language models. *Proc. VLDB Endow.*, 17(10):2514–2527.