A Demonstration of GPTuner: A GPT-Based Manual-Reading Database Tuning System

Jiale Lao[1], Yibo Wang[1], Yufei Li[1], Jianping Wang[2], Yunjia Zhang[3], Zhiyuan Cheng[4], Wanghu Chen[2], Yuanchun Zhou[5], Mingjie Tang[1], Jianguo Wang[4]
[1]Sichuan University, {laojiale, wangyibo2, evangeline}@stu.scu.edu.cn, tangrock@gmail.com
[2]Northwest Normal University, {2022222119, chenwh}@nwnu.edu.cn
[3]University of Wisconsin-Madison, yunjia@cs.wisc.edu
[4]Purdue University, {cheng443, csjgwang}@purdue.edu
[5]Computer Network Information Center, Chinese Academy of Sciences, zyc@cnic.cn

ABSTRACT

Selecting appropriate values for the configurable knobs of Database Management Systems (DBMS) is crucial to improve performance. But because such complexity has surpassed the abilities of even the best human experts, database community turns to machine learning (ML)-based automatic tuning systems. However, these systems still incur significant tuning costs or only yield sub-optimal performance, attributable to their overly high reliance on black-box optimization and an oversight of domain knowledge. This paper demonstrates GPTuner, a manual-reading database tuning system that leverages Large Language Model (LLM) to bridge the gap between black-box optimization and white-box domain knowledge. This demonstration empowers (1) regular users with limited tuning experience to gain qualitative insights on the features of knobs, and optimize their DBMS performance automatically and efficiently, (2) database administrators and experts to further enhance GPTuner by simply contributing their invaluable tuning suggestions in natural language. Finally, we offer visitors the opportunity to explore a range of DBMS and optimization metrics, coupled with the flexibility to tailor their target workloads to their specific needs.

KEYWORDS

Database Tuning, Large Language Model, Bayesian Optimization

ACM Reference Format:
Jiale Lao[1], Yibo Wang[1], Yufei Li[1], Jianping Wang[2], Yunjia Zhang[3], Zhiyuan Cheng[4], Wanghu Chen[2], Yuanchun Zhou[5], Mingjie Tang[1], Jianguo Wang[4]. 2024. A Demonstration of GPTuner: A GPT-Based Manual-Reading Database Tuning System. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Tuning configurable parameters (i.e., knobs) of modern Database Management Systems (DBMS) is crucial to improve performance. Since it is challenging to manage hundreds of knobs under diverse database instances and query workloads in the changing cloud environment, state-of-the-art methods turn to machine learning (ML) techniques to automate the tuning process. However, they either still incur significant tuning costs because they only rely on the runtime feedback of benchmark evaluations [7, 8], which is inefficient when the search space is high-dimensional (e.g., DBMS expose hundreds of knobs) and heterogeneous (e.g., a knob can be in continuous or categorical values), or they only yield sub-optimal performance since they utilize domain knowledge in a limited way (e.g., only consider suggested values of knobs from manuals) [5, 6].

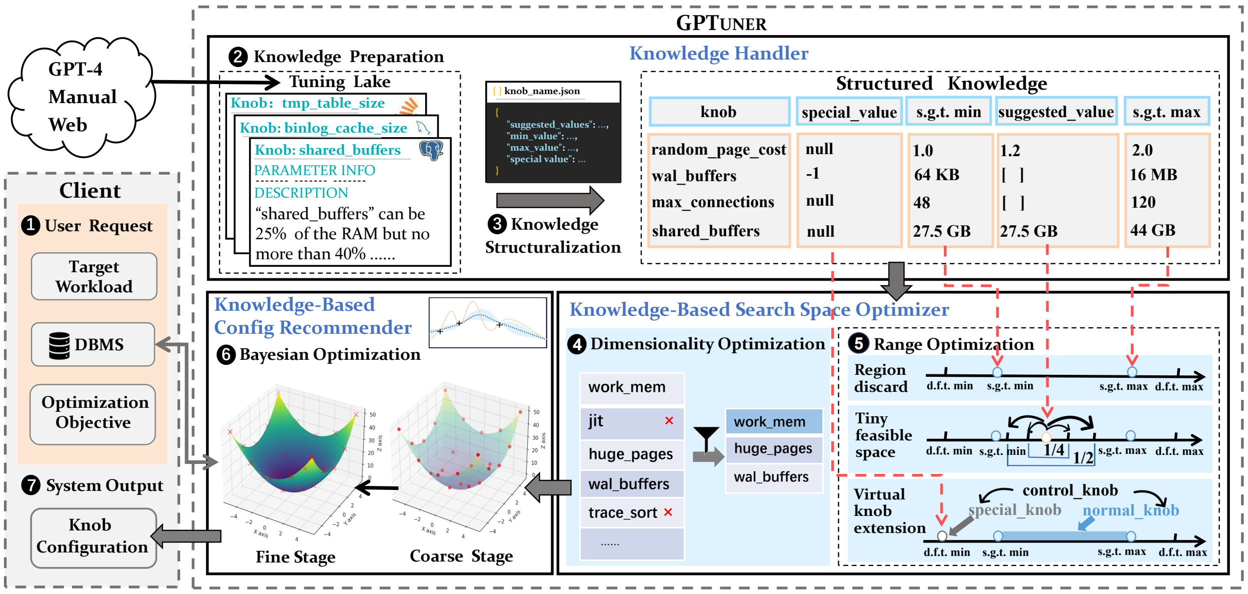
Unlike ML-based tuning methods that tune DBMS knobs based on performance statistics, human database administrators (DBAs) leverage extensive domain knowledge instead, including which knobs are worth tuning under certain scenarios[1], and which values should be considered given the unique semantics of a knob[2]. While it is widely acknowledged that such knowledge is invaluable in guiding ML-based tuning process, such wisdom seems exclusive to humans and inaccessible to machines due to the barriers in natural language understanding.

Recently, the advent of LLM makes it possible to leverage natural language knowledge. However, even equipped with the powerful LLM, it is still non-trivial to bridge the gap between black-box optimization and white-box domain knowledge, mainly due to three challenges: *C1. Lengthy and complex data pipeline.* Domain knowledge typically comes in the form of DBMS documents and discussions from web forums, which could be heterogeneous in format and noisy in contents. To leverage such knowledge, it involves a complex and lengthy data pipeline: data ingestion, data cleaning, data integration and data extraction. *C2. The brittle nature of LLM.* It is challenging to utilize LLM to solve complex and domain-specific tasks, because small modifications to the prompt can cause large variations in the model outputs, and the inevitable hallucination problem of LLM (i.e., LLM can generate answers that seem correct but are factually false). *C3. Lack of a knowledge-aware optimization framework.* The inherent design of optimization algorithms like Bayesian Optimization (BO) and Reinforcement Learning (RL) does not support the integration of external domain knowledge directly, necessitating extensive modifications to their standard workflows.

Our recent work GPTuner [3], which is under revision of VLDB 2024, addresses these challenges using a LLM-enhanced Bayesian

SIGMOD-Companion '24, June 9–15, 2024, Santiago, Chile
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

[1]For example, for an IO-intensive OLTP workload, it is recommended to adjust knob "`effective_io_concurrency`" from PostgreSQL.
[2]For instance, we should set "`random_page_cost`" to 1.x if we are using SSD disks.



knob	special_value	s.g.t. min	suggested_value	s.g.t. max
random_page_cost	null	1.0	1.2	2.0
wal_buffers	-1	64 KB	[ ]	16 MB
max_connections	null	48	[ ]	120
shared_buffers	null	27.5 GB	27.5 GB	44 GB

Figure 1: System Overview of GPTuner

Optimization (BO) approach. For *C1* and *C2*, we develop a LLM-based pipeline with two error correction mechanisms to collect and refine heterogeneous knowledge, and propose a prompt ensemble algorithm to unify a structured view of the refined knowledge. For *C3*, using the structured knowledge, we design a workload-aware and training-free knob selection strategy, develop a search space optimization technique considering the value range of each knob, and propose a Coarse-to-Fine Bayesian Optimization Framework to explore the optimized space.

This demo lets conference attendees experience GPTuner in actions. After specifying the target DBMS, the optimization metric and the customized workload, visitors can (1) invoke a LLM to help identify which knobs are worth tuning and only tune these important knobs, (2) interact with the LLM-based pipeline to see how the multi-source heterogeneous knowledge with noise is transformed into a unified structured view, (3) visualize the process of using the structured knowledge to optimize the value range of each knob, (4) execute the BO to explore the optimized space, and finally (5) visualize the optimization process and export the satisfied knob configuration in SQL format such that it can be deployed directly. Our full implementation and a video demonstration are available at https://github.com/SolidLao/GPTuner and https://youtu.be/Hz5Zck-9TlA, respectively.

2 SYSTEM OVERVIEW

Workflow. Figure 2 depicts the tuning pipeline of GPTuner. ❶ User provides the DBMS to be tuned (e.g., PostgreSQL or MySQL), the target workload (e.g., TPC-C, TPC-H or customized workload), and the optimization objective (e.g., latency or throughput). ❷ GPTuner collects and refines the heterogeneous knowledge from different sources (e.g., GPT-4, DBMS manuals and web forums) to construct *Tuning Lake*, a collection of DBMS tuning knowledge. ❸ GPTuner unifies the refined tuning knowledge from *Tuning Lake* into a structured view accessible to machines (e.g., JSON). ❹ GPTuner reduces the search space dimensionality by selecting important knobs to tune (i.e., fewer knobs to tune means fewer dimensions). ❺ GPTuner optimizes the value range of each knob based on structured knowledge. ❻ GPTuner explores the optimized space via a novel Coarse-to-Fine Bayesian Optimization framework, and finally ❼ identifies satisfactory knob configurations within resource limits (e.g., the maximum optimization time or iterations specified by users).

Components. We discuss the three components of GPTuner in the following sections: *Knowledge Handler* (Section 3), *Search Space Optimizer* (Section 4) and *Configuration Recommender* (Section 5).

3 KNOWLEDGE HANDLER

3.1 Knowledge Preparation

In the knowledge preparation stage, *Knowledge Handler* aims to prepare a reliable and consistent natural language knowledge base. To achieve this, it leverages LLM to (1) prepare multi-source knowledge from GPT-4, DBMS manuals and web forums, (2) filter out noisy contents by comparing the knowledge with its DBMS system view (e.g., *pg_settings* from PostgreSQL), (3) summarize the multi-source knowledge by handling the possible conflict between the remaining knowledge in a priority way (i.e., manuals have the highest priority while GPT-4 has the lowest due to its hallucination problem), and (4) check and revise the summarization to make sure it is factual consistent with the source contents.

3.2 Knowledge Transformation

In the knowledge transformation stage, *Knowledge Handler* intends to transform the refined natural language knowledge into a structured view (e.g., JSON) such that it can be utilized by machines. In the context of DBMS knob tuning, we primarily consider four types of attributes: *suggested_values*, *min_value*, *max_value* and *special_value*, whose roles are discussed in Section 4.2. Next, we use LLM to extract the values of the four attributes. To address the effect of the brittle nature of LLM as much as possible, we propose a *Prompt Ensemble Algorithm* since it is useful to acquire a more reliable result by aggregating multiple imperfect but effective results. Specifically, we prepare different prompts by sampling distinct examples for each prompt to utilize the in-context learning of LLM, and aggregate the results generated by each prompt with a Majority Vote strategy.

4 SEARCH SPACE OPTIMIZER

4.1 Dimensionality Optimization

In this part, *Search Space Optimizer* prunes space dimensionality by identifying knobs that have a significant impact on the DBMS performance, and only tune these important knobs. Specifically, it utilizes LLM to simulate DBA's empirical judgement in real-world scenarios and select knobs by considering the following four factors: *(1) DBMS Product.* After years of tuning practice, it is empirically known which knobs are important for a certain DBMS product. Since such wisdom is included in the corpus of GPT-4 [4], we can extract it by prompting GPT-4 to recommend knobs based on the DBMS product. *(2) Workload Characteristics.* Different workloads have distinct requirements on DBMS resources, which are regulated by knobs. For example, given an I/O-intensive OLTP workload, DBAs would consider tuning disk-related knobs like "`effective_io_concurrency`". *(3) Query Bottleneck.* Given LLM's powerful analysis ability, LLM is capable of delving into the execution plans of queries, and choosing knobs related to the performance bottleneck. *(4) Knob Dependency.* Some knobs need to be tuned at the same time to take effect, and LLM can capture such dependency by reading DBMS manuals.

4.2 Range Optimization

In this part, *Search Space Optimizer* utilizes the structured knowledge in Section 3.2 to optimize the value range of each knob.

Region Discard. For numerical knobs, *Search Space Optimizer* limits the value scope of a knob between *min_value* and *max_value*. Based on tuning experience, DBAs summarized such values in manuals or forums, and we extract these values to discard meaningless value regions, including values that are unlikely to result in promising performance, that could seize too many system resources, and that could even make the DBMS crash.

Tiny Feasible Space. *Search Space Optimizer* uses *suggested_values* to define a reliable discrete space. Such values are valuable since they performed well in the past and can serve as good starting points for the new scenario. However, they may not be suitable for all cases, as the optimal knob setting depends on the specific environment, which is diverse. Instead of relying on these static values only, we dynamically apply a set of multiplicators on each suggested value. We conduct above deviation process for all numerical knobs, and the resulting discrete space is our *Tiny Feasible Space*, where *Tiny* means the possible number of values is significantly reduced, and *Feasible* indicates the chosen values are promising.

Virtual Knob Extension. In modern DBMS, there are knobs using special values to do something different from what the knobs normally do. However, optimizers may never trial these values (even though it could be the optimal) since the likelihood of sampling them is extremely low [2]. Therefore, we provide separate boolean virtual knobs to control each knob with special values. Specifically, the virtual knobs are exposed to optimizers to determine whether the "normal value range" or "special value" is activated.

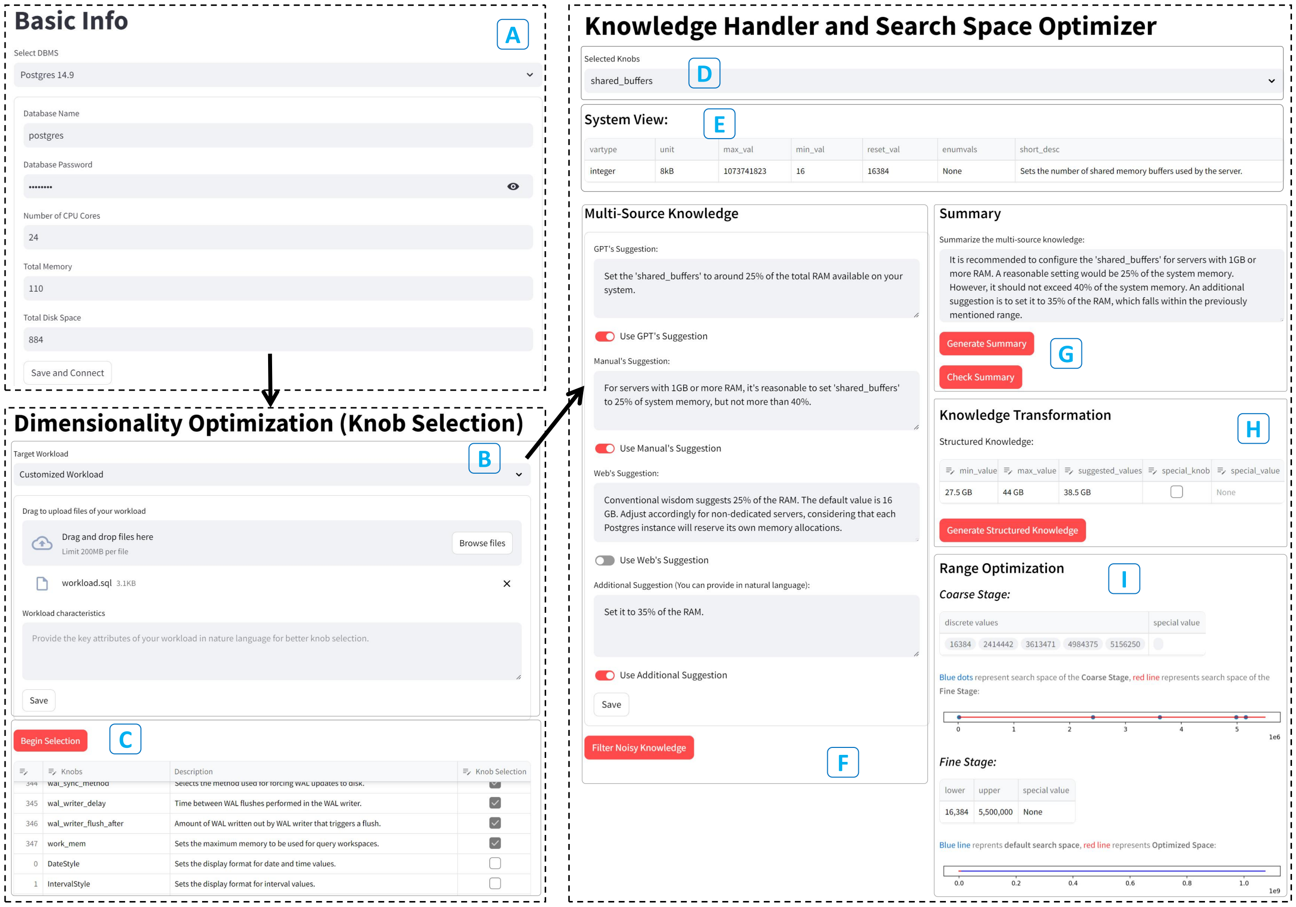
5 CONFIGURATION RECOMMENDER

Configuration Recommender utilizes a novel Coarse-to-Fine BO Framework to explore the search space under the guidance of domain knowledge, and recommends well-performing configurations. In the first stage, BO only explores a discrete subspace of the whole heterogeneous space (i.e., the *Tiny Feasible Space* defined in Section 4.2). This subspace is small in size but promising to contain good configurations since we generate it based on the reliable domain knowledge. In the next stage, in order to avoid the overlooking problem of coarse-grained search (i.e., it is inevitable to lose some useful configurations for any space reduction technique), BO explores the heterogeneous space thoroughly with the optimizations in Section 4.2 (i.e., Region Discard and Virtual Knob Extension). Moreover, we use the samples from the first stage to bootstrap the surrogate model of BO in the second stage. After the two stages, the recommender outputs the best-performing knob configurations found within the budget limits specified by users.

6 DEMONSTRATION SCENARIOS

This section demonstrates how visitors interact with GPTuner to tune their DBMS in two scenarios. Visitors have the opportunity to invoke GPT models. However, since this takes time and money, we provide intermediate results in advance for better interactivity.

Database Administrators (DBAs). Let us now walk through a typical scenario where a DBA aims to optimize a frequently running workload in the company. She first provides basic information of the target database in A to connect to the target database, where OpenAI API key can be optionally offered for personal model invocation. Then she uploads the workload in B and describes the characteristics of the workload for better knob selection. When clicking the button in C, GPTuner identifies important knobs and DBA can check and modify the selection, and the final selection will be displayed in D. When she selects one of the knobs (e.g., "`shared_buffers`"), the page updates to present the knowledge process pipeline for that knob. She can collect and offer tuning knowledge from multiple sources, and provides her unique tuning insight as "Additional Suggestion". By clicking the button in F, GPTuner filters noisy knowledge contradicting the system view in E, and the remaining consistent knowledge will be summarized in G. Next, GPTuner extracts the values of attributes from the summary, and generates the structured knowledge in H. At the same time, the optimized search spaces of the two stages are calculated and visualized in I, where she can check whether they are reasonable. Finally, when she clicks "Start Tuning" in J, GPTuner executes BO to explore the search spaces from coarse granularity to fine granularity. The tuning process is visualized and the best performance is reported in real time. Details about configurations of all iterations are presented in K, and she can export the satisfied configuration in SQL format to deploy it on her database instance directly.


Basic Info
A
Dimensionality Optimization (Knob Selection)
B
C
Knowledge Handler and Search Space Optimizer
D
E
F
G
H
I
Browse files
Save and Connect
Save
Begin Selection
Generate Summary
Check Summary
Generate Structured Knowledge
Filter Noisy Knowledge

Select DBMS: Postgres 14.9

Database Name: postgres

Database Password: ••••••••

Number of CPU Cores: 24

Total Memory: 110

Total Disk Space: 884

Target Workload: Customized Workload

Drag to upload files of your workload

Drag and drop files here
Limit 200MB per file

workload.sql 3.1KB

Workload characteristics

Provide the key attributes of your workload in nature language for better knob selection.

	Knobs	Description	Knob Selection
344	wal_sync_method	Selects the method used for forcing WAL updates to disk.	✓
345	wal_writer_delay	Time between WAL flushes performed in the WAL writer.	✓
346	wal_writer_flush_after	Amount of WAL written out by WAL writer that triggers a flush.	✓
347	work_mem	Sets the maximum memory to be used for query workspaces.	✓
0	DateStyle	Sets the display format for date and time values.	☐
1	IntervalStyle	Sets the display format for interval values.	☐

Selected Knobs: shared_buffers

System View:

vartype	unit	max_val	min_val	reset_val	enumvals	short_desc
integer	8kB	1073741823	16	16384	None	Sets the number of shared memory buffers used by the server.

Multi-Source Knowledge

GPT's Suggestion:

Set the 'shared_buffers' to around 25% of the total RAM available on your system.

Use GPT's Suggestion

Manual's Suggestion:

For servers with 1GB or more RAM, it's reasonable to set 'shared_buffers' to 25% of system memory, but not more than 40%.

Use Manual's Suggestion

Web's Suggestion:

Conventional wisdom suggests 25% of the RAM. The default value is 16 GB. Adjust accordingly for non-dedicated servers, considering that each Postgres instance will reserve its own memory allocations.

Use Web's Suggestion

Additional Suggestion (You can provide in natural language):

Set it to 35% of the RAM.

Use Additional Suggestion

Summary

Summarize the multi-source knowledge:

It is recommended to configure the 'shared_buffers' for servers with 1GB or more RAM. A reasonable setting would be 25% of the system memory. However, it should not exceed 40% of the system memory. An additional suggestion is to set it to 35% of the RAM, which falls within the previously mentioned range.

Knowledge Transformation

Structured Knowledge:

min_value	max_value	suggested_values	special_knob	special_value
27.5 GB	44 GB	38.5 GB	☐	None

Range Optimization

Coarse Stage:

discrete values	special value
16384 2414442 3613471 4984375 5156250	

Blue dots represent search space of the **Coarse Stage**, red line represents search space of the Fine Stage:

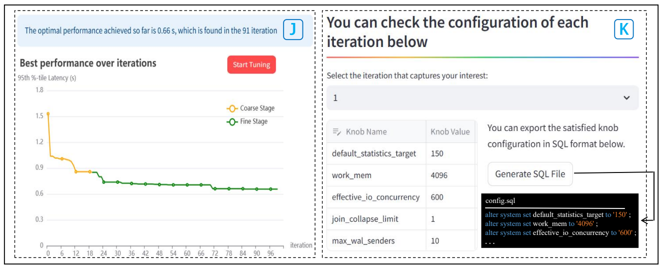
Fine Stage:

lower	upper	special value
16,384	5,500,000	None

Blue line reprents **default search space**, red line reprents **Optimized Space**:

Figure 2: GPTuner Demonstration

Regular Engineers. Let us now walk through the second scenario where a regular engineer wants to try out GPTuner for automatic DBMS optimization. In $\boxed{B}$, she can choose TPC-C or TPC-H as target workload, which are supported by Benchbase [1] without providing workload file and description manually. She can just leave knob selection default and learn about the tuning knowledge of each knob in $\boxed{E}$ and $\boxed{F}$. She can see how important attributes can be extracted from $\boxed{G}$ to obtain the structured knowledge in $\boxed{H}$, and see how efficiently GPTuner can reduce the search space in $\boxed{I}$, where the blue line denotes the default search space provided by DBMS vendors, the red line and blue dots denote the search spaces of our two-stage BO algorithm. Finally, by clicking "Start Tuning" in $\boxed{J}$, her target DBMS is optimized by GPTuner automatically.



The optimal performance achieved so far is 0.66 s, which is found in the 91 iteration

You can check the configuration of each iteration below

Select the iteration that captures your interest: 1

Knob Name	Knob Value
default_statistics_target	150
work_mem	4096
effective_io_concurrency	600
join_collapse_limit	1
max_wal_senders	10

You can export the satisfied knob configuration in SQL format below.

```
config.sql
alter system set default_statistics_target to '150' ;
alter system set work_mem to '4096' ;
alter system set effective_io_concurrency to '600' ;
. . .
```

Figure 3: GPTuner Result Page

REFERENCES

[1] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7 (2013), 277–288.
[2] Konstantinos Kanellis, Cong Ding, Brian Kroth, Andreas Müller, Carlo Curino, and Shivaram Venkataraman. 2022. LlamaTune: Sample-Efficient DBMS Configuration Tuning. *Proc. VLDB Endow.* 15 (2022), 2953–2965.
[3] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2023. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. arXiv:2311.03157 [cs.DB]
[4] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[5] Immanuel Trummer. 2021. The Case for NLP-Enhanced Database Tuning: Towards Tuning Tools That "Read the Manual". *Proc. VLDB Endow.* 14, 7 (2021), 1159–1165.
[6] Immanuel Trummer. 2022. DB-BERT: A Database Tuning Tool That "Reads the Manual" *(SIGMOD '22)*. Association for Computing Machinery, 190–203.
[7] Bohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shuli Jiang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J. Gordon. 2018. A Demonstration of the Ottertune Automatic Database Management System Tuning Service. *Proc. VLDB Endow.* 11 (2018), 1910–1913.
[8] Xinyi Zhang, Zhuo Chang, Yang Li, Hong Wu, Jian Tan, Feifei Li, and Bin Cui. 2022. Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation. *Proc. VLDB Endow.* 15 (may 2022), 1808–1821.