ABSTRACT
DB-BERT is a database tuning tool that mines tuning hints from text documents, including the database manual. DB-BERT uses mined hints as a starting point for an iterative tuning approach, guided via reinforcement learning. This demonstration enables visitors to try out DB-BERT for tuning different database management systems, including Postgres and MySQL, on benchmarks such as TPC-C and TPC-H. Visitors can vary the input text to observe how mined hints influence DB-BERT’s behavior. The demonstration interface allows tracing back configurations selected for trial runs to the text passages that motivated them. Finally, visitors may try to beat configurations proposed by DB-BERT with their own parameter settings. The code for this demo is publicly available at https://itrummer.github.io/dbbert/.

KEYWORDS
database performance tuning; text mining; reinforcement learning

CCS CONCEPTS
• Information systems → Autonomous database administration; • Computing methodologies → Information extraction; Reinforcement learning.

1 INTRODUCTION
A plethora of useful information for database tuning is stored in the form of natural language text. This text includes the database manual as well as various blog entries and forum discussions that focus on database performance. Up until quite recently, such information has been difficult to leverage for automated tuning tools. Now, with the advent of a new generation of powerful natural language processing methods, leveraging such information has become both possible and interesting.

DB-BERT, described in more detail in an upcoming SIGMOD paper [11], is a database tuning tool that exploits tuning hints mined from natural language text. DB-BERT exploits pre-trained language models for text analysis (also known as foundation models), in particular the BERT model [1]. Pre-trained language models have recently advanced the state of the art across various NLP benchmarks [14]. First, they are trained in an expensive step on tasks for which large amounts of training data are easily available (e.g., predicting obfuscated words in Web text). Second, they are fine-tuned for more specialized tasks for which training data may be sparse. Pre-training enables such models to achieve excellent performance on various natural language processing (NLP) tasks with little task-specific training [2].

DB-BERT uses such models to mine tuning hints from collections of text documents. Tuning hints recommend specific parameter settings in specific scenarios. Of course, hints found on the Web may be imprecise, and hints from different sources may conflict. This is why DB-BERT combines information mined from text with the results of trial runs. Trial runs measure performance, according to a user-specified metric, on specific workloads and platforms.

Trial runs cost valuable tuning time. Hence, DB-BERT selects configurations to try carefully. It uses reinforcement learning to determine which configuration to try next, balancing the need for exploitation (further analyzing configurations that seem promising) and exploration (trying out configurations about which little is known). Over the course of a tuning session, DB-BERT integrates information gained via trial runs to learn how to adapt tuning hints found in text, and how to weight hints from different sources against each other.

The proposed demonstration allows visitors to try out DB-BERT in different scenarios. Visitors are able to vary scenario properties and observe the behavior of DB-BERT. First, visitors may choose the database management system being tuned, selecting between Postgres and MySQL. Second, visitors may pick the workload and performance metric for tuning, selecting between TPC-H with different scaling factors (and latency as tuning goal) and TPC-C (optimizing transaction throughput). Finally, visitors may vary the input text to study how the quality and relevance of text hints impacts DB-BERT’s performance. Note that DB-BERT aims at short tuning time frames in the order of few minutes (this is possible due to using mined hints as starting point). This means that visitors can try out different settings and observe DB-BERT’s behavior live over the duration of a typical demonstration.

The remainder of this paper is organized as follows. Section 2 discusses the architecture of DB-BERT in more detail. Section 3

Immanuel Trummer
Cornell University
Ithaca, New York, USA
itrummer@cornell.edu
DB-BERT first performs a pre-processing step (Step A and B in Figure 1) in which the input text is analyzed. Initially (Step A), DB-BERT extracts candidate hints from text using pre-trained language models. This step involves matching text passages to database parameters (whose names may not be explicitly mentioned) as well as extracting text snippets with recommended settings for each parameter. Currently, DB-BERT supports hints that recommend either absolute values or recommend ratios, relative to the main memory size, disk size, or the number of CPU cores. Next, DB-BERT uses a simple heuristic to determine the order in which hints are analyzed more deeply (Step B).

Example 2.1. Table 1 (taken from the full paper [11]) illustrates tuning hints supported by DB-BERT. These hints include relative hints (the first and the third hint) that propose settings which are relative to system properties (here: the amount of main memory). Also, they include absolute recommendations (the second hint) that propose a fixed value. Note that the third tuning hint does not explicitly mention the name of the parameter it refers to. Here, DB-BERT infers the most likely parameter by comparing parameter names to hint text.

After pre-processing, DB-BERT iteratively constructs configurations (i.e., settings for a subset of parameters), based on tuning hints, and evaluates them in trial runs (Step C to H). In each iteration, DB-BERT translates text hints into arithmetic formulas specifying the recommended value (Step C), decides how to deviate from given hints (Step D), and (Step E) assigns weights to hints determining their relative importance (in case of conflicting recommendations from multiple sources). All of those decisions are made by a reinforcement learning agent, guided by a neural network whose weights are fine-tuned over the course of a tuning session. Weighted hints are aggregated (Step F) to form configurations that are evaluated in trial runs (Step G). After a user-specified time limit, the best configuration found is returned.

3 EXTRACT OF EXPERIMENTS

DB-BERT finds promising parameter settings quickly as it exploits text as an additional source of information. DB-BERT has been compared to multiple baselines in a detailed experimental evaluation [11]. This section presents an extract from those experiments. Figure 2 (taken from the full paper [11]) shows how DB-BERT improves run time on the TPC-H benchmark (with scaling factor one) over the course of a tuning session. For comparison, the figure also shows performance of multiple baselines.

The x-axis of the figure represents tuning time. The y-axis represents execution time (for all TPC-H queries) of the best configuration found over the corresponding tuning period. Each data point represents the arithmetic average of five runs (error bars represent the 20th and 80th percentile respectively). The upper plot shows performance on Postgres 13.2 (with 232 tuning parameters) while the lower plot refers to MySQL 8.0 (with 266 tunable parameters). The plot compares DB-BERT against the DDPG++ algorithm [12], a state-of-the-art reinforcement learning approach. The three DDPG versions differ by the value range considered for tuning parameters (ranging from a factor of two up to a factor of 100 around the default settings). Also, the plot contains results for two previously proposed methods (Prior-Simple and Prior-Main) that mine tuning

Table 1: Example tuning hints with extractions.

| Text Snippet | Extraction |
|--------------|------------|
| The default value of shared_buffer is set very low ... The recommended value is 25% of your total machine RAM. [8] | shared_buffers = 0.25 · RAM |
| I changed ‘random_page_cost’ to 1 and retried the query. This time, PostgreSQL used a Nested Loop and the query finished 50x faster. [6] | random_page_cost = 1 |
| On a dedicated database server, you might set the buffer pool size to 80% of the machine’s physical memory size. [5] | innodb_buffer_pool_size = 0.8 · RAM |

presents an extract of an experimental evaluation. Section 4 compares DB-BERT against prior work and Section 5 describes the demonstration setup.

2 SYSTEM OVERVIEW

Figure 1 (this figure is taken from the full paper on DB-BERT [11]) shows an overview of the DB-BERT system. As input, DB-BERT obtains the tuning goal (i.e., a workload and a performance metric to optimize), a collection of text documents (used for mining hints), and a description of the hardware of the tuned system (this is needed to translate relative tuning hints, recommending for instance using a certain percentage of RAM). The output is a recommended configuration, i.e. a mapping from database system parameters to specific parameter values.
related work

DB-BERT aims at finding database parameter settings that optimize performance on given workloads. This problem has received significant attention in the database research community over the past years [3, 4, 7, 13, 15, 16]. It is motivated by the ever-growing number of tuning parameters, allowing to fine-tune for instance the amount of main memory allocated for the database system or the logging strategy. Many recently proposed approaches in this space use reinforcement learning [9] to repeatedly select parameter settings for trial runs [4, 12, 13]. Based on the performance measured during trial runs, the tuning approach learns over time which configurations maximize performance. DB-BERT also uses reinforcement learning and trial runs. The difference to prior work lies in the fact that DB-BERT explores configurations that are motivated by hints, mined from natural language text. In that, DB-BERT connects to another recent approach [10] that relies on natural language text alone to propose database configurations. However, DB-BERT does not rely on text alone but complements hints mined from text by measurements obtained via trial runs. Overall, the proposed demonstration is the first to showcase a database tuning tool that exploits hints mined from text.

5 demonstration proposal

The proposed demonstration enables visitors to try out DB-BERT in different scenarios. The demonstration features multiple database management systems (i.e., Postgres and MySQL), multiple workloads (i.e., TPC-H and TPC-C) and performance metrics (i.e., latency and throughput), as well as multiple input text documents. The set of available document collections includes generic recommendations (e.g., text from 100 Web documents retrieved via the Google query "Postgres performance tuning") as well as more specialized text documents (e.g., a Stackoverflow forum discussion about how to tune Postgres specifically for TPC-H). The set of input documents used influences the performance of DB-BERT. For instance, using a small document collection with specialized hints that match the current tuning goals well can enable DB-BERT to find good configurations faster. Beyond using pre-selected document collections, visitors will also be able to add new documents and to experiment with them.

The interface allows visitors to select all of the aforementioned scenario properties. Next, visitors can choose a tuning time which determines the number of iterations performed by DB-BERT. For the selected benchmarks, DB-BERT typically converges within a few minutes. It tends to achieve significant improvements, according to the selected performance metric, already during the first few iterations. This enables visitors to try out multiple settings over the typical duration of a demonstration.

DB-BERT continuously generates output over the course of a tuning session. This output is interesting for visitors and illustrates how the system works. Among other things, the output includes the following components. First, DB-BERT informs users on candidate hints extracted during pre-processing, as well as their order of priority. Second, DB-BERT produces output describing, during iterations, how hints are translated, adapted, and weighted. Those decisions may change, even for the same hint, over the course of a tuning session. Furthermore, DB-BERT generates output summarizing configurations selected for trial runs, the measured performance, as well as the hints that were used to construct configurations.

Finally, visitors may try out different variants of DB-BERT. Specifically, visitors can change DB-BERT’s behavior in the following ways. First, visitors can influence the pre-processing phase, e.g. by selecting different heuristics to prioritize mined hints. Second, visitors can influence the types of hints DB-BERT is mining for. For instance, they can choose whether to consider "implicit hints"
DB-BERT Demonstration

DB-BERT uses hints mined from text for database tuning. Select DBMS:

- PostgreSQL

Select Benchmark:

- TPC-H

Select Metric:

- Latency

Enter Path to Text:

[/Users/immanueltrummer/git/iteratedDBtuners/tuning_docs/log_tpcch_single]

Enter Iteration Limit:

1 +

Enter Timeout (s):

600 – +

Start Tuning

Figure 3: Screenshot of DB-BERT interface used for demonstration.

in which the parameter name is not explicitly mentioned. Third, visitors can influence the pre-trained language models used for text mining as well as the reinforcement learning algorithm used for selecting configurations. By observing the impact of different settings on DB-BERT’s behavior, visitors gain a deeper understanding of the factors that contribute to its performance. Finally, visitors may try out parameter settings themselves and evaluate performance on different benchmarks.

Figure 3 shows a screenshot of the Web interface used for the demonstration. This interface enables visitors to select the most important parameters conveniently. Visitors select the database management system, the benchmark, the tuning metric, as well as the text documents used for tuning. Also, they select a limit on the maximal number of iterations (during each iteration, DB-BERT constructs and benchmarks parameter settings) as well as a timeout.

After hitting the “Start Tuning” button, DB-BERT starts the tuning session. It first analyzes the input text documents and generates output that summarizes extracted hints (the parameter to which hints refer to, the recommended value, and the text passage from which the hint was extracted). Next, DB-BERT starts iterations, using extracted hints as a starting point.

While iterating, DB-BERT regularly generates output in the Web interface. This output summarizes the best and worst configurations found so far. It summarizes parameter settings (focusing only on parameters where settings were changed, compared to the defaults) as well as observed performance (e.g., latency).

The Web interface allows visitors to tune the most important parameters and summarizes the most important results. Visitors who are interested in gaining more insights into the internals of DB-BERT are able to access more detailed logs and to tune more parameters using configuration files. The additional parameters include, for instance, whether or not DB-BERT tries to infer implicit hints or which heuristic is used for prioritizing extracted hints. Also, interested visitors can gain further insights into the decision-making process used by DB-BERT. To that purpose, visitors are able to access logs that explain each decision made for each hint in each of the tuning iterations.

6 SUMMARY

The proposed demonstration focuses on the DB-BERT system, a tuning tool that extracts tuning hints from text. Visitors will be able to interact with DB-BERT, and to observe its behavior in various scenarios. Visitors can vary the input text, the database management system being tuned, as well as benchmark and performance metrics.

REFERENCES

[1] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL. Vol. 1. 4171–4186. arXiv:1810.04805
[2] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In ACL. 328–339. https://doi.org/10.3760/cma.j.issn.04124081.2010.02.006
[3] Konstantinos Kanellis, Ramnathan Alagappan, and Shivvaram Venkataraman. 2020. Too many knobs to tune? Towards faster database tuning by pre-selecting important knobs. PVLDB 2020 - 12th USENIX Workshop on Hot Topics in Storage and File Systems, co-located with USENIX ATC 2020 (2020).
[4] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2018. QTune: A QueryAware database tuning system with deep reinforcement learning. PVLDB 12, 12 (2018), 2118–2130. https://doi.org/10.14778/3352063.3352129
[5] Oracle. 2021. MySQL 8.0 Reference Manual. 1692 pages. dev.mysql.com
[6] Pavan Patibandla. 2017. https://amplitude.engineering/how-a-single-postgresql-config-change-improved-slow-query-performance-by-50c-8559f8b9f1b0.
[7] Andrew Pavlo, Gustavo Angulo, Jey Andruh, Haimin Lin, Juree Lin, Lin Ma, Prashanth Menon, Todd C. Morew, Matthew Perron, Ian Quash, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-driving database management systems. In CIDR.
[8] PERCONA. 2018. https://www.percona.com/blog/2018/08/31/tuning-postgresql-database-parameters-to-optimize-performance/.
[9] RS Richard S Sutton and Andrew G Ag Barto. 1998. Reinforcement learning: an introduction. https://doi.org/10.1109/TNN.1998.712392
[10] Immanuel Trummer. 2021. The case for NLP-enhanced database tuning: towards end-to-end automatic cloud database tuning system using deep reinforcement learning. PVLDB 14, 11 (2021), 2104–2120. https://doi.org/10.14778/3450980.3450992
[11] Immanuel Trummer. 2022. DB-BERT: a database tuning tool that “reads the manual”. In SIGMOD. 1–13.
[12] Dana Van Aken, Dongsheng Yang, Sébastien Brillard, Ari Fiorino, Bohan Zhang, Christian Bilien, and Andrew Pavlo. 2021. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. Proceedings of the VLDB Endowment 14, 7 (2021), 1241–1253. https://doi.org/10.14778/3450980.3450999
[13] Junxiong Wang, Immanuel Trummer, and Debabrata Basu. 2021. UDO: Universal Database Optimization using Reinforcement Learning. In VLDB. Vol. 1. Association for Computing Machinery, 1–13. arXiv:2104.01744 https://arxiv.org/abs/2104.01744
[14] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick Von Platen, Clara Ma, Vacine Jerunke, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Guggier, Marigana Drane, Quentin Lhoest, and Alexander M Rush. 2020. Transformers : State-of-the-Art Natural Language Processing. (2020). 38–45.
[15] Rohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shu Liang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J Gordon. 2018. A demonstration of the OtterTune automatic database management system tuning service. VLDB 11, 12 (2018), 1910–1913.
[16] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhihui Xiao, Bin Cheng, Jiang Xu, Yangtao Wang, Tianheng Cheng, Li Liu, Minwei Ran, and Zekang Li. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In SIGMOD. 415–432. https://doi.org/10.1145/3299869.3300085