The Case for NLP-Enhanced Database Tuning: Towards Tuning Tools that "Read the Manual"

Immanuel Trummer
Database Group, Cornell University
Ithaca, New York
itrummer@cornell.edu

ABSTRACT

A large body of knowledge on database tuning is available in the form of natural language text. We propose to leverage natural language processing (NLP) to make that knowledge accessible to automated tuning tools. We describe multiple avenues to exploit NLP for database tuning, and outline associated challenges and opportunities. As a proof of concept, we describe a simple prototype system that exploits recent NLP advances to mine tuning hints from Web documents. We show that mined tuning hints improve performance of MySQL and Postgres on TPC-H, compared to the default configuration.

PVLDB Reference Format:
Immanuel Trummer. The Case for NLP-Enhanced Database Tuning: Towards Tuning Tools that "Read the Manual". PVLDB, 14(7): 1159 - 1165, 2021.
doi:10.14778/3450980.3450984

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://tinyurl.com/9crrjezv.

1 INTRODUCTION

A large body of work is available on the topic of database tuning. This work comes in the form of vendor-provided documentation, blog entries, or scientific papers. It covers various aspects of database tuning, different systems, hardware, and software configurations, as well as different use cases. This treasure trove of tuning knowledge is mainly targeted at human database administrators. Hence, it is represented as natural language text. Up to date, automated tuning tools do not benefit from it directly.

We propose to leverage recent advances in natural language processing (NLP) to make this information accessible to database tuning tools. The state of the art in NLP has recently advanced significantly by the advent of pre-trained language models. Those language models correspond to large neural network that have been pre-trained using large text corpora. As a result, training them for custom NLP tasks is relatively cheap and requires only modest amounts of training data [10]. This makes them particularly well suited for domains such as database tuning where labeling training data requires highly specialized knowledge (making it hard to obtain large collections of samples).

We see multiple avenues for leveraging NLP for database tuning. First, there are myriads of tuning hints available in the form of natural language text (e.g., on the Web or in manuals). Those hints specify for instance how to set values for various database system parameters to optimize performance. They cover different scenarios in terms of platform (e.g., different settings are often recommended for different operating systems or hardware setups) or workload (e.g., the best settings may differ between analytical and transactional workloads). Often, such hints propose formulas, rather than fixed parameter values, allowing them to generalize across a range of situations. In principle, automated tuning tools may identify optimal parameter settings by trying options in a principled manner (e.g., via reinforcement learning [16]) or based on large amounts of training data. However, mining hints from text documents can complement those approaches by speeding up convergence or reducing the amount of required training data.

Second, NLP may enable tuning tools to gain a deeper understanding of parameter semantics. For instance, certain parameters trade performance for reduced consistency guarantees. Such dependencies are typically pointed out in the manual, or outlined in related tuning hints on the Web. The detrimental effects of reduced consistency guarantees manifest rarely, making it hard to identify them based on experiments alone. Hence, parsing text documents can save the need for additional, system-specific user input, prior to automated tuning. Third, beyond additional text documents, already the names of parameters or database elements can hold valuable information. For instance, an experienced DBA may form an educated guess about parameter semantics based on parameter names alone. This can help forming priors for tuning options (e.g., increasing parameters related to memory and buffer space tends to improve performance on analytical workloads). We propose to enable database tuning tools to apply a similar kind of reasoning.

In the remainder of this paper, we first provide additional background on natural language processing and on database tuning (see Section 2). Then, we outline the aforementioned research opportunities in more detail in Section 3. Next, in Section 4, we discuss first experimental results on mining tuning hints from text documents. Our early prototype parses text documents and uses pre-trained language models to identify key sentences, proposing parameter settings. Further, we classify key sentences according to tuning hint type, and extract relevant context as well as parameter names and proposed values. We evaluate our prototype for MySQL and Postgres, using tuning hints mined from Web documents. We show that mined configuration recommendations improve performance

Proceedings of the VLDB Endowment, Vol. 14, No. 7 ISSN 2150-8097.
doi:10.14778/3450980.3450984

on TPC-H, compared to default settings. Finally, we summarize and point out future work directions in Section 5.

2 BACKGROUND

We discuss prior work in NLP in Section 2.1. Giving a complete overview of NLP methods is beyond our scope. Instead, we focus on recent developments that connect to this work. Section 2.2 discusses prior work in database tuning and optimization.

2.1 Natural Language Processing

For many NLP tasks, achieving state of the art performance nowadays requires large neural networks with many parameters [27]. Training such models is expensive and requires large training sets. In many scenarios, this makes it prohibitively expensive to train models from scratch for all but the most common NLP tasks. This has recently motivated large, pre-trained language models that can be specialized to new tasks with limited overheads. Arguably, some of the most impressive advances in NLP over the past few years are due to such models [24].

Pre-trained language models follow the high-level idea of transfer learning. They are trained on generic tasks for which large amounts of training data are available. Then, they are specialized to new tasks using small amounts of training data and with moderate overheads. For instance, the recently proposed BERT model [6] is pre-trained on the task of masked language modeling. Here, the goal is to infer masked words in input sentences. Doing so successfully requires a certain "understanding" of syntax and semantics as well as some commonsense knowledge. All of those are important for other NLP tasks as well which makes language modelling a suitable pre-training objective (together with the large amounts of available training data).

A pre-trained model can be applied to new tasks either by fine-tuning (i.e., adapting weights of a pre-trained model via task-specific training) or by using output of the pre-trained model as features for another, task-specific model. It has been shown that using pre-trained models can reduce the number of task-specific samples required for competitive performance by multiple orders of magnitude [10]. This seems particularly important when using text analysis for database tuning, as labeling samples often requires specialized knowledge (e.g., by database administrators) or expensive data processing (if labels are based on performance results gained via experiments).

Pre-trained models, and NLP methods more broadly, have been used in the context of database systems before [15, 17, 25]. However, typically, they are used on the interface side, to make data more accessible. Here, we propose to use NLP methods for performance optimization instead.

2.2 Database Tuning and Optimization

The performance of a database system can be influenced via various parameters. These include system configuration parameters, typically specified in configuration files [16, 23, 33]. More broadly, they include parameters related to physical design (e.g., which index structures or materialized views to generate [3, 5, 9, 30]), scheduling decisions [14], and query planning [26].

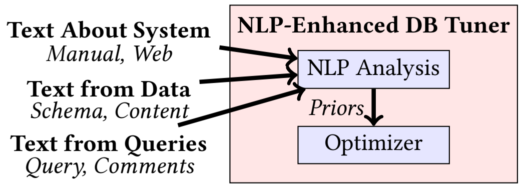

Figure 1: NLP-enhanced database tuner: system, data, and query related text are used to form priors for optimization.

Most of those tuning problems are difficult for two reasons. First, many of them are NP-hard [4, 5, 12]. This implies high computational cost for finding optimal solutions. Second, it is often difficult to model performance in data processing as a function of tuning choices. For instance, it is notoriously hard to estimate the cardinality of intermediate results during query processing [8]. This makes it hard to assess the execution cost of query plans during planning. Already for that reason, it is difficult to predict the impact of all tuning choices (e.g., index creations) that are aimed at speeding up query processing.

Challenges due to unreliable cost estimates have recently motivated a new wave of tuning approaches, based on machine learning [13, 19, 20, 22, 23]. We use the term "tuning" in a broad sense, encompassing all of the aforementioned tuning and optimization problems. The approach proposed in this paper belongs into the same broad category, as it is implicitly based on machine learning. It differs from prior work as it uses machine learning for text analysis. We envision NLP-enhanced database tuning to be used not as an alternative but in combination with other tuning methods. It unlocks one additional source of tuning knowledge that can be used for generating constraints, priors, or priorities for optimization.

3 RESEARCH VISION

We introduce the idea of NLP-enhanced database tuning in general in Section 3.1. In Sections 3.2 and 3.3, we describe instances of the aforementioned ideas, based on two concrete use cases.

3.1 NLP-Enhanced Database Tuning

A treasure trove of database tuning knowledge is available in the form of natural language text. For instance, any diligent database administrator will first consult the manual when maximizing performance for an unfamiliar system. Beyond the manual, myriads of tuning hints have been published on the Web in blog entries, online forums, or scientific papers.

Relevant knowledge is hidden not only in text documents. Even the names of tuning parameters in configuration files, together with associated comments, may already provide some intuition for their semantics and promising values to try. The names of database schema elements, as well as short associated descriptions in a data dictionary, may carry useful information on their semantics. Even queries themselves, containing user-defined column names and comments, may contain text fragments that are helpful to understand query intent and ultimately optimize their execution.

We argue that recent advances in NLP enable automated tuning tools to benefit from knowledge contained in text fragments. Figure 1 shows a template for NLP-enhanced database tuning tools. We envision systems that may exploit any subset of the aforementioned types of text fragments, related to system, data, or query workload. As discussed in more detail next, we see multiple use cases for different database tuning problems. We do *not* argue to rely exclusively on text for making tuning decisions. We see NLP as a complementary mechanism, unlocking additional sources of information that complement others. Other sources of information include statistics from past query executions [1], as well as information that is collected during tuning by executing sample workloads with specific configurations (which is the case for tuning methods that rely on active learning [19] or reinforcement learning [29, 33]). Knowledge gained by text analysis could for instance serve as a prior for the optimizer, as illustrated in Figure 1. This means that optimization will initially favor tuning options that are consistent with suggestions extracted from text. Reinforcement learning (the approach adopted by several recent database tuning methods [16, 21, 31]) can integrate priors in various ways. For instance, Monte-Carlo Tree Search methods [2] typically select actions randomly during initial exploration. Here, priors can be used as a domain-specific heuristic instead. Next, we outline two concrete use cases for NLP-enhanced database tuning.

3.2 NLP-Enhanced System Configuration

Database systems nowadays feature a large set of tuning parameters. As pointed out in prior work [1, 32], the number of tuning parameters keeps growing, making it hard to find optimal configurations manually. In this context, we see multiple avenues to exploit NLP. First, there is a variety of tuning hints available in online forums. Those hints cover different workloads, tuning goals, hardware properties and software platforms. The optimal parameter values depend on that context. Those tuning hints are not always reliable (and tuning hints from different sources may conflict). We believe however that aggregating hints from a variety of textual sources yields at least a reasonable starting point for further tuning. We verify this intuition in the following section by experiments. In some cases, even the name of a parameter may yield hints on which settings to prioritize. E.g., considering a parameter with the term "buffer_size" in its name, we may first try to increase it (rather than decrease it) in order to speedup analytical workloads. Such priors could be gained by analyzing correlations between performance and parameter settings for a sufficient number of database systems.

Second, we may use NLP to gain a deeper understanding of parameter semantics. For instance, there are often parameter settings that increase performance while allowing isolation anomalies. Also, it is often possible to improve performance over extended periods by postponing expensive clean-up operations (or paying with higher overheads during a recovery for instance). Analyzing text in the manual or in forums can help to spot problematic parameter settings. For instance, parameter settings that relax isolation are often identified explicitly in associated text. If an appropriately trained text classifiers identifies such passages, the associated setting can be discarded or de-prioritized during optimization.

3.3 NLP-Enhanced Cardinality Estimation

Imagine a query filtering a large data set by two predicates. One predicate restricts column "dayOfWeek" to a constant, the other one restricts column "uniqueCustomerID" to a constant. Assuming a target table with millions of rows, and preferring evaluating more selective predicates earlier, which one would you evaluate first?

Most of us would intuitively evaluate the predicate on the second column ("uniqueCustomerID") first. Based on the name of the column, we expect unique values. If so, an equality predicate filters out all but at most one row. For the other column, based on its name ("dayOfWeek"), we expect a value domain of size seven. Assuming a uniform distribution of rows over values, the selectivity of an equality predicate is $1/7$. For large tables, we expect the predicate on customer ID to be the more selective one. Hence, using it first for filtering is more efficient.

Column names or data dictionary entries associated with schema elements can sometimes provide a prior for cardinality estimation. Such priors can be learned by considering a sufficiently large number of data sets, together with cardinality-related information, during training. In particular, certain column names may imply expectations on the size of the associated value domain. Also, correlations between columns are often indicated by column names (e.g., a data set with columns named "date" and "dayOfWeek"). Cardinality estimates are often based on assumptions on independent column content. Priors gained from column names can provide first warning signs that those assumptions are invalid.

Analyzing text fragments associated with schema elements cannot replace traditional cardinality estimation methods. Column names do not always convey useful information. Also, column names may be misleading or not paint the full picture. For instance, the data set from the prior example may not contain any entries with values "Saturday" or "Sunday" in the "dayOfWeek" column (e.g., if it describes sales of a shop closed on week-ends). Then the value domains is smaller than the name suggests.

Instead of an alternative, we consider NLP-enhanced analysis rather as a complement to existing methods. The results of the latter can be used to corroborate estimates obtained by more traditional methods. Also, they can be used to guide efforts in gathering additional information. For instance, prior work has aimed at detecting inter-column correlations based on samples [11]. If resources for correlation detection are limited, one can prioritize column groups where the names indicate a likely correlation.

4 PRELIMINARY RESULTS

We perform a proof of concept for the scenario outlined in Section 3.2. Section 4.1 describes a prototype of a simple NLP Analysis component (see Figure 1) which could be used in that scenario. This prototype mines tuning recommendations from Web documents. In Section 4.2, we describe experiments in which we use mined hints directly for configuration. In a full blown system, following the sketch from Section 3.1, mined recommendations would form the input for further optimization stages.

4.1 Prototype Description

Figure 2 shows an overview of a simple prototype system. The goal of this system is to mine tuning hints for database systems from input text. The input to the system is a collection of text documents that (potentially) contain relevant recommendations for tuning a specific database system. Additionally, the system receives names

System	Sentence with Context	Hint Expression	Type	Condition
Postgres	If you have a system with 1GB or more of RAM, a reasonable starting value for shared_buffers is 1/4 of the memory in your system	$shared\_buffers = 0.25 \cdot memory$	Assign	$memory \geq 1GB$
	There are some workloads where even larger settings for shared_buffers are effective, but given the way PostgreSQL also relies on the operating system cache, it's unlikely you'll find using more than 40% of RAM to work better than a smaller amount	$shared\_buffers \leq 0.4 \cdot memory$	Upper Bound	-
	Be aware that if your system or PostgreSQL build is 32-bit, it might not be practical to set shared_buffers above 2-2.5GB	$shared\_buffers \leq 2.5GB$	Upper Bound	32-bit
	The maintenance_work_mem parameter basically provides the maximum amount of memory to be used by maintenance operations like vacuum, create index, and alter table add foreign key operations ... It's recommended to set this value higher than work_mem; this can improve performance for vacuuming	$maintenance\_work\_mem \geq work\_mem$	Lower Bound	-
MySQL	innodb_stats_on_metadata Setting this to "OFF" avoids unnecessary updating of InnoDB statistics and can greatly improve read speeds	$innodb\_stats\_on\_metadata = OFF$	Assign	-
	innodb_flush_method ... If your disk is stored in SAN, O_DSYNC might be faster for a read-heavy workload with mostly SELECT statements	$innodb\_flush\_method = O\_DSYNC$	Assign	SAN, read-heavy

Table 1: Subset of tuning hints found on first ten Web documents retrieved via Google queries "Postgres database tuning" and "MySQL database tuning".

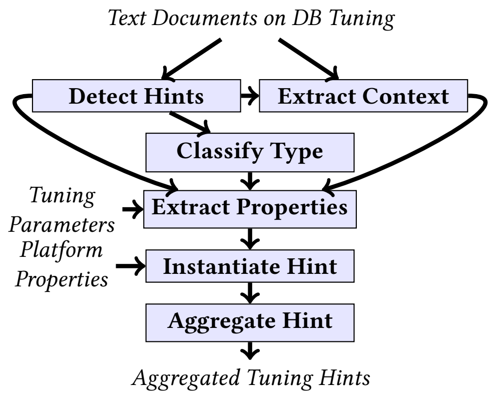

Figure 2: Overview of prototype: we parse tuning hints from text documents, classify them, instantiate parameterized hints for specific platforms, and aggregate potentially conflicting hints.

of system-specific configuration parameters, and information about platform properties (e.g., the main memory size). Next, we describe the different steps of the pipeline in detail.

First, we identify text snippets in the input that give actionable tuning advice. For that, we first decompose the input into single sentences. Then, we detect sentences that propose values or value ranges for specific parameters. We call them "key sentences" in the following. We detect such sentences using a classifier based on the Roberta language model [18]. We describe the training process in the next subsection.

Sentences with tuning recommendations often depend on prior context. In particular, we observe that parameter names are often specified in a section heading or in sentences that precede the key sentence. Hence, we extract context for each key sentence. For the moment, we use a simple heuristic that retrieves the last prior sentence mentioning a parameter name.

Example 4.1. Table 1 shows a few of the tuning hints that can be extracted from Web documents. The hints focus on Postgres and MySQL. The first stage of the pipeline aims at extracting key sentences, such as the ones shown in Table 1, from surrounding text. Some of them require context (shown in blue).

Next, we classify key sentences based on their type. Here, we use again a classifier based on Roberta [18]. We assume that each hint can be translated into an equation that involves the referenced parameter. We classify hints based on the type of formula. E.g., we distinguish hints proposing specific values from hints defining lower bounds, upper bounds, or sets of recommended values. For the moment, our prototype only considers hints that propose specific values. Exploiting value ranges requires a mechanism that explores the associated search space. We plan to do so in future versions.

System	#Sentences	#Key S.	# Context S.
Postgres	2980	46	29
MySQL	3056	35	20

Table 2: Size of labeled training data.

Stage	Postgres	MySQL
Detecting Hints	4m27s	4m45s
Classifying Hints	16s	17s

Table 3: Training times for detecting hints in text and for classifying given hints.

Task	Approach	MCC	Recall	Precision	F1
Detect	Baseline	0.27	0.31	0.25	0.28
	This	0.51	0.49	0.55	0.52
Classify	Baseline	0.04	0.05	1	0.09
	This	0.27	0.91	0.71	0.8

Table 4: Quality metrics when processing hints for MySQL after training with hints for Postgres.

Task	Approach	MCC	Recall	Precision	F1
Detect	Baseline	0.25	0.33	0.22	0.26
	This	0.6	0.48	0.76	0.59
Classify	Baseline	0	0.09	1	0.17
	This	0.21	0.82	0.53	0.64

Table 5: Quality metrics when processing hints for Postgres after training with hints for MySQL.

For each sentence classified as key sentence proposing a specific value, we extract parameter name and proposed setting (the hint "properties"). Tuning hints do not always propose concrete values for a parameter. Instead, they propose a formula that may depend on constants as well as other parameters or system properties (e.g., the number of cores, main memory, operating system, or disk space). Currently, we support extraction of specific parameter values as well as percentages of main memory (one of the most common types of formulas in our sample). After this step, we instantiate tuning hints by filling in concrete platform properties for placeholders (e.g., the precise amount of main memory).

Example 4.2. Table 1 shows hints associated with key sentences. Hints are represented by an expression that limits the set of possible values to a subset of recommended values. We distinguish different types of expressions (e.g., single value assignments versus lower bounds). Also, some hints come with conditions under which the recommendations apply.

Different text authors may disagree with regards to the best parameter settings. Additionally, the optimal setting may depend on context (e.g., tuning tools and constraints as well as workload properties). While we plan to refine our text analysis to distinguish different situations, our current prototype does not yet support this functionality. Finally, our extraction is noisy and may lead to incorrect tuning hints. For all of these reasons, extractions from different text documents may contradict each other. We aggregate hints from different documents in a final step. For the moment, we simply filter tuning hints to the ones that are mentioned at least twice by independent sources. More sophisticated resolution methods could cluster tuning hints that are not exactly but approximately equivalent. Also, we may weight documents based on the reputation of the document source.

4.2 Experimental Results

We obtained preliminary experimental results for the prototype outlined before. We focus on tuning MySQL and Postgres. We downloaded the first ten documents returned by each of the Google queries "MySQL database tuning" and "Postgres database tuning" respectively. In all 20 documents, we identified "key sentences", including the ones in Table 1, that propose specific value domains for specific parameters. Additionally, we identified sentences that provide context (e.g., the parameter name) required to understand the key sentence. Table 2 reports the size of the corresponding subsets. While relatively small, note that pre-trained models have shown competitive performance on some tasks with as little as 100 labeled training samples [10].

Our prototype uses two classifiers that require supervised training. First, we classify sentences into key and non-key sentences. Next, we classify key sentences into six categories based on the type of the associated logical expression (assignment, lower or upper bound, value range, value set assignment, warning to exclude a specific value). To make the task more challenging, we train both classifiers with hints for one system and test with data for the other system. This simulates the appearance of new database systems for which hints must be parsed, using training data from prior systems. We train on the Google CoLab platform [1], using Python 3.6.7 as programming language and the simple transformers library[2]. Starting from a pre-trained model ("Roberta-base"), we perform 20 epochs when training to detect tuning sentences and 10 epochs for classifying tuning sentences. We weight key sentences with a factor of 50 in the loss function to make up for class imbalance. We used a default GPU instance with Intel Xeon 2.2GHz CPU, 12.7 GB of main memory, 68.4 GB of hard disk, and a Tesla T4 GPU.

Table 3 shows training times for both classifiers and for both systems. Next, we test the classifiers trained with data for the first system on documents targeted at the other. We compare against a simple baseline. The baseline considers sentences containing parameter names and a numerical value as key sentences. The baseline classifies a sentence, based on the number of values that appear in it (e.g., multiple values indicate a set of recommended values, a single value may indicate an assignment or a bound). It selects the class

[1]https://colab.research.google.com/

[2]https://simpletransformers.ai/

Parameter	Baseline	This
innodb_buffer_pool_size	2.9 GB	16 GB
innodb_log_file_size	16 MB	(default)
query_cache_size	0	(default)
query_cache_type	0	(default)
innodb_buffer_pool_instances	(default)	8
innodb_flush_log_at_trx_commit	(default)	0
join_buffer_size	(default)	4 GB

Table 6: Changes to default configuration for MySQL.

Parameter	Baseline	This
shared_buffers	16 GB	16 GB
maintenance_work_mem	1 GB	(default)
checkpoint_completion_target	0.9	(default)
effective_cache_size	(default)	4 GB

Table 7: Changes to default configuration for Postgres.

System	Default	Baseline	This
MySQL	307	97	82
Postgres	141	121	119

Table 8: Total time for TPC-H queries in seconds for different configurations.

randomly among all classes that are consistent with the number of values. Tables 4 and 5 report results for the trained classifiers and the baselines. We compare according to Matthew's Correlation Coefficient (respectively the generalization to the multiclass case [7]). We also calculate precision, recall, and the F1 score, focusing on retrieving sentences that can be used by the current prototype (i.e., tuning hints of assignment class). The classifiers outperform the baselines.

Next, we evaluate the end-to-end impact of the tuning hints mined by the prototype. Again, we train classifiers with documents from one classifier and process all documents retrieved for the other system. Besides the classifiers, we use the simple heuristics outlined before. During aggregation, we only keep tuning hints that are mined from at least two independent sources. For MySQL, the system identified 15 sentences as likely tuning sentences with hints, 14 of them as likely assignments. Four of the hints extracted from those sentences appeared at least twice. For Postgres, the system identified 17 sentences as likely tuning sentences, 10 of them as likely assignments. Two extracted hints appeared at least twice.

We compare against two tuning systems that are specialized for Postgres and for MySQL respectively: the MySQL Tuner[3] and the

[3]https://github.com/major/MySQLTuner-perl

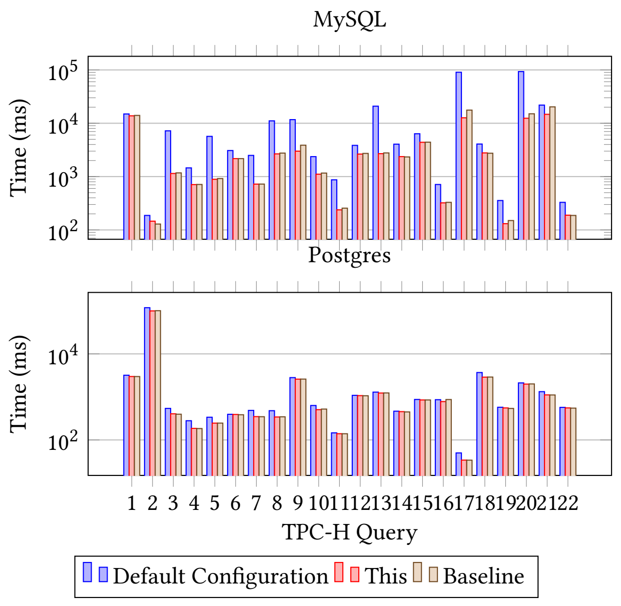

Figure 3: Comparing Postgres and MySQL performance with different configurations.

Postgresql Tuner[4]. We consider them as strong baselines as those tools have been specialized for the database systems we evaluate on. Tables 6 and 7 describe the changes to the default configuration that are proposed by the prototype ("This") as well as by the respective tuning tool ("Baseline"). We evaluate different configurations on the TPC-H benchmark [28] with scaling factor 1. We use Postgres 10.15 and MySQL 5.7.32 on an Amazon EC2 t2.2xlarge instance with 200 GB of EBS gp2 storage. The operating system is Ubuntu 18.04. For each configuration, we perform one warmup run before the actual measurement.

Figure 3 shows per-query run times and Table 8 shows total run time. Clearly, both, the configuration proposed by our prototype as well as the one proposed by the baselines, work better than the default configuration. Among the two fine-tuned configurations, the one proposed by the prototype is slightly better. The results come with several caveats. E.g., first, the system-specific tuning tools consider additional aspects beyond pure performance (e.g., security, main memory consumption). Second, the number of analyzed documents is still fairly small. Nevertheless, the results demonstrate potential for integrating text analysis into database tuning tools.

5 CONCLUSION

We advocate for the use of text documents and snippets as one additional source of information in database tuning. NLP-enhanced database tuners can exploit such information to complement or corroborate other sources of information, or to guide efforts in collecting additional information online. We presented and evaluated a simple prototype of an NLP-enhanced database tuner. We demonstrated that information parsed automatically from the Web can yield significant speedups, compared to default configurations.

[4]https://github.com/jfcoz/postgresqltuner

REFERENCES

[1] Dana Van Aken, Andrew Pavlo, and Geoffrey J Gordon. 2017. Automatic database management system tuning through large-scale machine learning. In *SIGMOD*. 1009–1024. https://doi.org/10.1145/3035918.3064029

[2] CB Browne and Edward Powley. 2012. A survey of monte carlo tree search methods. *Trans. on Computational Intelligence and AI in Games* 4, 1 (2012), 1–49. http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6145622

[3] Alberto Caprara, Matteo Fischetti, and Dario Maio. 1995. Exact and approximate algorithms for the index selection problem in physical database design. *KDE* 7, 6 (1995), 955–967. https://doi.org/10.1109/69.476501

[4] S. Chatterji and SSK Evani. 2002. On the complexity of approximate query optimization. In *PODS*. 282–292. https://doi.org/10.1145/543649.543650

[5] Surajit Chaudhuri. 2004. Index selection for databases: A hardness study and a principled heuristic solution. *KDE* 16, 11 (2004), 1313–1323. http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1339260

[6] 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

[7] J. Gorodkin. 2004. Comparing two K-category assignments by a K-category correlation coefficient. *Computational Biology and Chemistry* 28, 5-6 (2004), 367–374. https://doi.org/10.1016/j.compbiolchem.2004.09.006

[8] Andrey Gubichev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *PVLDB* 9, 3 (2015), 204–215.

[9] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. 1997. Index selection for OLAP. In *ICDE*. 208–219. https://doi.org/10.1109/icde.1997.581755

[10] 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

[11] I F Ilyas, V Markl, P Haas, P Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*. 647–658. https://doi.org/10.1145/1007568.1007641 arXiv:ISBN 0-89791-128-8

[12] Howard Karloff and Milena Mihail. 2013. On the complexity of the cinepanettone. In *PODS*. 200–213. https://doi.org/10.1057/9781137305657

[13] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: estimating correlated joins with deep learning. In *CIDR*. arXiv:1809.00677 http://arxiv.org/abs/1809.00677

[14] H Kllapi, E Sitaridi, M M Tsangaris, and Y E Ioannidis. 2011. Schedule Optimization for Data Processing Flows on the Cloud. In *SIGMOD*.

[15] Fei Li and HV Jagadish. 2014. NaLIR: an interactive natural language interface for querying relational databases. *SIGMOD* (2014), 709–712. https://doi.org/10.1145/2588555.2594519

[16] 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

[17] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. (2020), 4870–4888. https://doi.org/10.18653/v1/2020.findings-emnlp.438 arXiv:2012.12627

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv* 1 (2019). arXiv:1907.11692

[19] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *SIGMOD*. 175–191. https://doi.org/10.1145/3318464.3389768

[20] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2018. Neo: A Learned query optimizer. *PVLDB* 12, 11 (2018), 1705–1718. https://doi.org/10.14778/3342263.3342644 arXiv:1904.03711

[21] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2018. Learning State Representations for Query Optimization with Deep Reinforcement Learning. In *DEEM*. arXiv:1803.08604 http://arxiv.org/abs/1803.08604

[22] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. QuickSel: Quick Selectivity Learning with Mixture Models. In *SIGMOD*. 1017–1033. https://doi.org/10.1145/3318464.3389727 arXiv:1812.10568

[23] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, 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*.

[24] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer Learning in Natural Language Processing. In *ACL: Tutorials*. 15–18.

[25] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Ozcan. 2016. ATHENA: An ontology-driven system for natural language querying over relational data stores. *VLDB* 9, 12 (2016), 1209–1220.

[26] PG G Selinger, MM M Astrahan, D D Chamberlin, R A Lorie, and T G Price. 1979. Access path selection in a relational database management system. In *SIGMOD*. 23–34. http://dl.acm.org/citation.cfm?id=582095.582099

[27] Amirsina Torfi, Rouzbeh A. Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A. Fox. 2020. Natural language processing advancements by deep learning: A survey. *arXiv* (2020), 1–21. arXiv:2003.01200

[28] TPC. 2013. TPC-H Benchmark. http://www.tpc.org/tpch/

[29] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: regret-bounded query evaluation via reinforcement learning. In *SIGMOD*. 1039–1050.

[30] Jian Yang, Kamalakar Karlapalem, and Qing Li. 1997. Algorithms for materialized view design in data warehousing environment. In *VLDB*. 136–145. http://www.vldb.org/conf/1997/P136.PDF

[31] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. 2020. Automatic view generation with deep learning and reinforcement learning. In *ICDE*, Vol. 2020-April. 1501–1512. https://doi.org/10.1109/ICDE48307.2020.00133

[32] Bohan Zhang, Dana Van Aken, Justin Wang, Tao Dai, Shuli Jiang, Jacky Lao, Siyuan Sheng, Andrew Pavlo, and Geoffrey J Gordon. 1910. A demonstration of the OtterTune automatic database management system tuning service. *VLDB* 11, 12 (1910), 1910–1913.

[33] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, 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