Enhancing the Configuration Tuning Pipeline of Large-Scale Distributed Applications Using Large Language Models (Idea Paper)

Gagan Somashekar*
gsomashekar@cs.stonybrook.edu
PACE lab, Stony Brook University
Stony Brook, NY, USA

Rajat Kumar*
rajat.kumar@alumni.stonybrook.edu
Department of Biomedical Engineering, Stony Brook University
Stony Brook, NY, USA

ABSTRACT

The performance of distributed applications implemented using microservice architecture depends heavily on the configuration of various parameters, which are hard to tune due to large configuration search space and inter-dependence of parameters. While the information in product manuals and technical documents guides the tuning process, manual collection of meta-data for all application parameters is laborious and not scalable. Prior works have largely overlooked the automated use of product manuals, technical documents and source code for extracting such meta-data. In the current work, we propose using large language models for automated meta-data extraction and enhancing the configuration tuning pipeline. We further ideate on building an in-house knowledge system using experimental data to learn important parameters in configuration tuning using historical data on parameter dependence, workload statistics, performance metrics and resource utilization. We expect productionizing the proposed system will reduce the total time and experimental iterations required for configuration tuning in new applications, saving an organization both developer time and money.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; **Software configuration management and version control systems**; • **Computing methodologies → Information extraction**; **Information extraction**.

KEYWORDS

Microservice Architecture, Parameter Tuning, Large Language Models, Information Retrieval

*Both authors contributed equally to this research.

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3585032

ACM Reference Format:
Gagan Somashekar and Rajat Kumar. 2023. Enhancing the Configuration Tuning Pipeline of Large-Scale Distributed Applications Using Large Language Models (Idea Paper). In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578245.3585032

1 INTRODUCTION

Optimizing the performance and efficiency of systems via configuration tuning is a classical problem [22, 34, 35]. Recently, configuration tuning of modern distributed applications has garnered significant interest [23, 36, 37]. These modern distributed applications usually leverage the microservice architectural style, where an application or a service is decomposed into independent, fine-grained services that communicate via well-defined APIs agnostic to the implementation [17, 27].

The microservice architecture is rapidly gaining traction in the industry as it offers many advantages, including scalability, fault-isolation, and ease of deployment [17]. It is particularly well-suited for building online, customer-facing cloud applications where the revenue is contingent upon the performance and efficiency of the applications [13, 24, 27]. Nevertheless, the performance and efficiency of these applications are heavily dependent on the configuration of their various parameters, which must be carefully tuned to ensure optimal results.

Configuration tuning of such large and complex applications is challenging. An online application can consist of a frontend (e.g., Nginx), database (e.g., MongoDB) and caching microservices (e.g., Redis) along with services that implement the business logic where each component has its own set of tunable parameters. Since modern cloud services contain hundreds and thousands of such microservices [27], the configuration search space is *extremely large*. Moreover, to obtain optimal results, tuning parameters across the software stack (e.g., orchestration layer, Operating System, etc.) is essential [11, 23], which exponentially increases the configuration search space. Dimensionality reduction, a common technique to tackle large search spaces, is complicated due to the interdependence between parameters of the same and different microservices and between parameters across the software stack. The interference among colocated microservices and the non-linear relationship between the parameters and the performance metric exacerbate the problem. Additionally, the optimal configuration for an online application varies depending on the workload characteristics (requests per second, workload composition, payload size, etc.). Importantly,

the exploration of optimal configuration has to happen without disrupting the performance or availability of the application.

The information in product manuals, technical documents[1], and feedback (e.g., performance metrics) from the experiments are essential tools to address these challenges. However, manually collecting the necessary meta-data for all application parameters from different microservices' product manuals is a laborious and often overlooked step in configuration tuning [39]. Moreover, the list of parameters and their meta-data has to be continuously updated as a) change in the architecture of the application, i.e., addition (removal) of a microservice can increase (decrease) the application parameters, b) newer versions of the individual microservice (e.g., MongoDB) can update the meta-data of parameters or add new and deprecate existing parameters.

Typically, a practitioner "understands" crucial information in the product manuals and, guided by empirical observations and telemetry, tunes the application to obtain optimal results. For a single sub-system like a database with so many parameters, this is time-consuming and inefficient [39]. Naturally, this approach is laborious and almost impractical for applications with numerous such sub-systems. Prior works on configuration tuning have often overlooked exploiting these important sources of information, namely product manuals, technical documents, source code, and experimental data, together [6–8, 11, 23, 36, 37, 47]. Some prior works that consider these sources of information have either utilized a limited amount of information from manuals [38, 39] or completely disregarded the information provided in the manuals and technical documents by solely relying on source code to extract meta-data [45]. This is surprising given that product manuals, and technical documents provide a plethora of information regarding parameters. This information plays a vital role in comprehending the correlation between performance and parameters, ultimately identifying the most suitable configuration settings for a specific workload and application architecture.

This work proposes how recent advances in Natural Language Processing (NLP), specifically Large Language Models (LLMs) [9, 14, 25, 32], can enhance the configuration tuning pipeline. Firstly, we believe that LLMs can learn from expert knowledge readily available in product manuals, developer forums and other textual data corpus, along with leveraging information available in the source code to extract valuable meta-data required for runtime configuration tuning of large-scale cloud applications. Secondly, a language model that is adapted to a specific domain and fine-tuned gradually with feedback from experiments can learn associative and causal representations among parameters across a suite of cloud applications. Specifically, we want to leverage the fact that different enterprise applications implemented using microservice architecture not only have microservices of the same type (e.g., a different instance of MongoDB as a backend for each application), but they also have shared services. For illustration, if applications are represented as graphs with microservices as nodes and a possible interaction between microservices as edges, then common sub-graphs will correspond to the shared services. The existence of common sub-graphs implies that what is learned for configuration tuning for a given application may be applied to another due to shared services. To the best of our knowledge, this commonality has not been exploited in any previous work. The specific ideas that the current work aims to explore to enhance the configuration tuning pipeline are:

- ***Meta-data extraction using targeted language model:*** Given a collection of documents (product manuals, technical documents, source code), we aim to learn a targeted language model to extract meta-data listed in Table 1 for all the microservices of the application.
- ***Enhance the configuration tuning pipeline using the learned language model:*** Using the collected meta-data, workload forecasting [28, 29] and application characterization, we aim to translate different tasks in the configuration tuning pipeline as text completion tasks facilitated via a language model operating on engineered input prompts (e.g., finding the most impactful parameters of an application for the current workload).
- ***Building An In-house Knowledge System:*** As an ambitious goal, we aim to demonstrate that service providers can use such models to learn tasks across different applications considering the similarities among such applications [27]. For each application and associated configuration tuning, we propose logging experimental data for fine-tuning the language model to enable associative and causal relationship learning over time.

Meta-data	Comments
Name	Name of the parameter
Type	Categorical/numeric/binary
Default	Default value
Range	Valid range or list of allowed values
Online	Feasibility of runtime configuration
Online Cost	Performance cost of runtime configuration
Dependencies	List of other parameters it depends on
Units	The unit of measurement
Set/Update Parameter	Steps to set/update the parameter

Table 1: Meta-data necessary for effective online tuning of large-scale cloud services.

The rest of this paper is structured as follows: in Section 2, we provide an overview of the background information on the configuration tuning pipeline, parameter meta-data, and LLMs, as well as discuss related works. The proposed approach, including details on the model, it's training, and use case is discussed in Section 3. In Section 4, we briefly touch on our planned evaluation strategy. Finally, we conclude the paper in Section 5.

2 BACKGROUND AND RELATED WORK

In this section, we first briefly mention the different stages of a typical configuration tuning pipeline, followed by a detailed description of the meta-data useful during configuration tuning. We also provide a brief introduction to LLMs and end the section with a brief discussion of closely related works.

[1]Technical documents refer to any natural language text related to the parameters, including articles written by product developers, source code documentation, public blogs and posts from third-party sources.

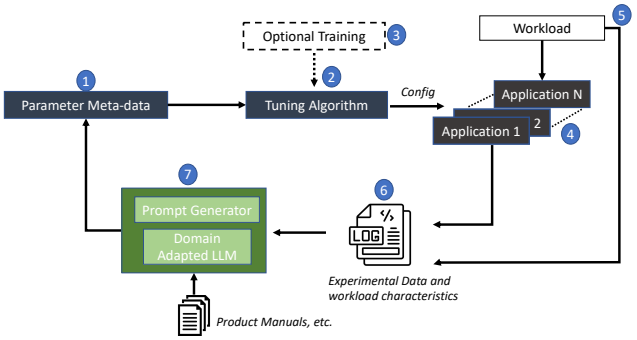

Figure 1: The envisioned configuration tuning pipeline. "Prompt Generator" and "Domain Adapted LLM" are proposed additional components that we believe will enhance the configuration tuning pipeline.

2.1 Configuration Tuning Pipeline

In this section, we discuss critical stages or components of a configuration tuning pipeline, followed by optional stages. We briefly mention the stages we plan on adding to enhance the configuration tuning pipeline.

A typical configuration tuning pipeline consists of the following critical stages: 1) Parameter meta-data collection, 2) An algorithm that suggests the next configuration coupled with an optional training phase, and 3) An application deployed using the suggested configuration that provides feedback (e.g., performance metrics like latency) on the chosen configuration. In Figure 1, these different stages correspond to (1), (2), (4), respectively. The optional training stage ((3)), as the name suggests, trains the tuning algorithm on offline data before deployment. The workload component in the figure ((5)) corresponds to either synthetic workload or real-world traffic. In order to enhance the configuration tuning pipeline, we propose adding a Prompt Generator and a Domain-adapted LLM ((7)) which use meta-data from the experiments stored in (6). These stages are discussed in detail in Section 3.

2.2 Parameter Meta-data

Product manuals contain a wealth of information about the parameters. Product manuals, coupled with technical documents (along with source code when manuals are incomplete or outdated [31]), can provide the meta-data listed in Table 1 that are essential for runtime configuration tuning. While the role of most of this meta-data is straightforward, we briefly mention here for completion:

- *Type*, which can be categorical, numeric or binary, is required for downstream optimization algorithms.
- *Default*, the default value of the parameter is usually a good starting point for exploration.
- *Range*, the range (allowed values) of the numeric (categorical) parameter is required for the exploration of the search space and to avoid passing invalid values to the parameters.
- *Online*, a binary variable that indicates whether a runtime configuration of the parameter is possible without affecting the application availability.
- *Online Cost*, a measure of performance impact (without any application unavailability) on runtime update of the parameter.
- *Dependencies*, a list of other parameters it depends on.
- *Units*, the unit of measurement useful when resolving dependencies in different units.
- *Set/Update Parameter*, the set of commands or steps to set the parameter or update the runtime value of the parameter.

While most of these are essential for configuration tuning, dependencies, which are not usually considered by prior works [10, 23, 30, 36], are necessary for reducing the dimension, a critical task considering the size of the configuration space. We define a dependency between two parameters, `P1` and `P2`, as a directional edge from `P1` to `P2`, indicating that `P2`'s knowledge is necessary to configure `P1` accurately. We further classify dependencies into:

- *Absolute:* Parameter `P2` has to be set/unset for parameter `P1` to have any effect. For example, Redis' `maxmemory` parameter has to be set for Redis' `maxmmeory-policy` to have any effect [3].
- *Partial:* Parameter `P2` affects the valid range or possible values of parameter `P1`. The valid range of Memcached's `max-memory (-m)` depends on the value of the container's `memory-limit` parameter [2].
- *Performance:* Parameters `P1` and `P2` have no explicit relationship, but tuning them jointly is necessary to obtain the best configuration. For example, the number of concurrent read transactions in MongoDB can hit the limit and affect the performance if the size of the cache is low [1].

The manuals and technical documents also contain information on the relationship between parameters and workload or application characteristics. For instance, tuning Nginx' `threads` and `max_queue` parameters when the workload does not involve any file system access is unnecessary as offloading of tasks to thread pools is only supported when the workload results in file system access [40]. The manuals also contain information that relates workload characteristics with the parameters and the tradeoffs of tuning the parameter, which is valuable when tuning different metrics (e.g., performance vs. resource savings).

2.3 Large Language Models

Large language models (LLMs) [9, 14, 25, 32] are trained on massive and heterogenous corpora, with texts ranging from Wikipedia, books to web content. Representations learned by these models are thus effective across a multitude of natural language tasks. While these models perform well on generalized language text, they may exhibit poor task performance on smaller target domain specific tasks (like scientific literature, code repositories etc.) due to differences in underlying distribution of the training and the test data (domain shift) [33]. Domain adaptation can help mitigate the domain shift by exposing the model to unsupervised secondary pretraining [20, 33], wherein the usual model pretraining is followed by domain/task-specific pretraining. LLMs have found rapid adoption in solving various software engineering problems [4, 5, 16, 18, 19, 39, 43, 44]. In the current proposal, our goal is to ideate domain adaptation of a pretrained large language model for integrating its use in configuration tuning pipeline for cloud services.

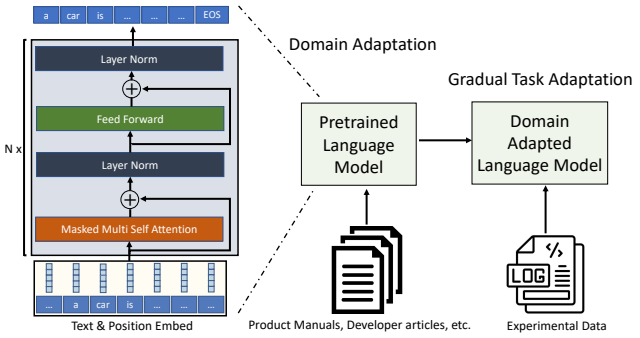

Figure 2: (left) Transformer architecture and training objective for autoregressive language modeling (GPT) [32]; (center) Domain adaptation of the pretrained GPT model using product manuals and other domain-relevant text (section 3.2); (right) Periodic finetuning of the language model using experimental logs (section 3.4).

2.4 Related Work

Optimization of systems through configuration tuning is a well-studied problem [11, 15, 23, 26, 28, 36–39, 41, 42, 45, 47]. Prior works [23, 30, 36, 37] that target large-scale cloud applications fail to utilize the information present in natural language and source code. The techniques to reduce the dimensions of the configuration space using system characteristics don't consider the dependencies of the parameters [30, 36].

Among the closely related works, SPEX [45] extracts most of the parameters' meta-data in Table 1 with the help of developer annotations and source code analysis. The authors acknowledge that the product manuals consist of all these meta-data but are limited by the NLP technology of the period. Recent works [21, 38, 39] use NLP in their configuration tuning pipeline. Trummer [38] trains the Transformer model to extract parameters and the suggested values from the product manuals. DB-BERT [39] is a database tuning tool that uses a pre-trained BERT [14] model to extract information from product manuals and other relevant documents to get suggestions on the best value for different parameter values. Both these works use NLP to get the parameter and value pair from text documents. He et al. [21] process technical documents using NLP to warn users about side-effects on non-performance intentions (e.g., reliability, security). However, these methods fail to fully utilize the potential of natural language understanding as i) they are not able to perform automated and exhaustive mining of text, and ii) they do not utilize the language models for learning new associations and dependencies based on experimental feedback i.e., no knowledge update occurs.

3 PROPOSED APPROACH

The configuration tuning pipeline we envision is shown in Figure 1. We add a Prompt Generator and a Domain-adapted LLM (⑦ in Figure 1) in addition to the typical stages (① to ⑤ in Figure 1) to enhance the configuration tuning pipeline. In this section, we discuss the motivation behind the choice of our model, followed by a discussion of how we plan to perform domain adaptation and prompt engineering. We end the section with a discussion of our goal to use the model as an in-house knowledge system for tuning different applications in an enterprise.

3.1 Large Language Model Selection

LLMs can be classified into two primary categories based on their training objective, namely – the masked language modeling (MLM) and autoregressive language modeling. In a typical MLM, some tokens are replaced by a special token '[MASK]', with an objective to predict the original token using the context around the '[MASK]' (for example, BERT [14]). On the other hand, autoregressive modeling works by predicting the next token in a sentence given the previous tokens i.e., these models do not have access to the future/upcoming tokens when generating the current token in a sentence. As autoregressive models learn a sequential generative process for text data, these models naturally perform better for natural language generation (for example, GPT-3 [9, 32] and Transformer-XL [12]). We propose to use autoregressive models of the GPT-x family. This choice is based on the fact that text completion requires learning the generative process. Additionally, GPT-x have code compatible models like Codex and Code-davinci-002, which is advantageous when learning text representation of user manuals that have unnatural tokens (from natural language perspective) like commands along with sample code snippets and for extracting metadata from source code when product manuals are incomplete or have errors. Codex and Code-davinci-002 are trained on both natural language and billions of lines of public code from GitHub.

3.2 Domain Adaptation

Once a pretrained LLM has been selected, we will perform domain adaptation using a collection of documents containing information on microservices. Domain adaptation will allow the language model to mitigate against domain shift and also learn targeted representations, which may not have been learnt by the pretrained LLM (for example, ChatGPT [openAI's chat bot based on GPT-3.5 model] incorrectly finds no relationship between the MongoDB's `wiredTigerConcurrentReadTransactions` and `cache_size` parameters [1]). We will start by preprocessing the data (remove URLs, special tokens and symbols, concatenate the whole corpus and divide into equal-sized chunks to prevent truncation for long text samples, and tokenization), setting hyperparameters, loading and compiling the pretrained model, followed by Domain Adaptive Pretraining (DAPT) [20, 33]. For evaluating quality of language modeling during domain adaptation, we will use metrics such as perplexity, BLEU score or ROUGE score. Additional evaluation will be performed via human evaluation ratings of the generated text.

3.3 Prompt Engineering

Once we have a domain-adapted model, we can use it to obtain metadata and complex inter-parameter relationships for experimental tuning via text completion of developer-generated prompts. We will obtain a parameter list by using simple prompts such as: "the exhaustive list of parameters in MongoDB (without description)

are as follows:." Once we have extracted parameters, we can obtain their values and attributes (Table 1) with prompts similar to: "The default value of `ldapUserCacheStalenessInterval` is." The prompts will be more complex for capturing parameter dependencies within and across different sub-components of the distributed application. They would require engineering prompts using developer knowledge and the required use case.

3.4 Building An In-house Knowledge System

For every configuration tuning experiment performed for various enterprise applications across an organization, we propose logging experimental data like performance metrics, CPU, memory and other resource utilization numbers, and workload statistics (requests per second, request composition, etc.). We also assume the existence of design documents describing the application's architecture. We will convert these experimental logs and the design documents to paired prompts of the form {"prompt": "<verbal description of the workload for the application>", "completion": "<optimal subset of parameters>"} and will use them for fine-tuning the domain adapted LLM (as obtained in section 3.2). GPT-x family language models require pairs of prompt and completion in the form {"prompt":<>, "completion":<>} for fine-tuning (gradual task adaptation in figure 2). Fine-tuning will allow the language model to learn about dependencies between different components, such as frontend, backend, etc., which are usually not captured in formal documentation and require experimentation. We envision collecting logs for every experiment followed by periodic fine-tuning of the domain-adapted LLM for a knowledge update.

4 PLANNED EVALUATION

We plan to evaluate our approach on small-scale, on-premises deployments and large-scale cloud deployments of the widely used DeathStarBench benchmarking suite [17] and the train ticket application [46]. We will specifically evaluate the following aspects:

- Compare the quality of meta-data generated using our model with that generated by program synthesis.
- A study on developer hours saved by automating parameter meta-data extraction.
- Compare the list of impactful parameters generated by our model with other techniques to find impactful parameters [30, 36].
- Measure the generalization of the fine-tuned language model to new applications versus manual tuning without the language model, using the number of trials required to reach the optimal configuration as the metric.

5 CONCLUSION

We discussed and presented a case for using LLMs to enhance the configuration tuning of large cloud services implemented using the microservice architecture. Additionally, we ideated on how to use experimental feedback for learning language models that may be able to generalize across applications for configuration tuning.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for providing valuable feedback that has improved the quality of this paper. Additionally, we would like to thank Prof. Anshul Gandhi and Akshata Bhat, whose discussions helped shape the direction of our research. This work was supported by the NSF grant CNS-1750109.

REFERENCES

[1] 2015. MongoDB Jira. jira.mongodb.org/browse/SERVER-19911.
[2] Accessed in January 2023. memcached(1). https://linux.die.net/man/1/memcached.
[3] Accessed in January 2023. Redis configuration. https://redis.io/topics/config.
[4] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. https://doi.org/10.48550/ARXIV.2103.06333
[5] Toufique Ahmed, Supriyo GHOSH, Chetan Bansal, Tom Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *ICSE 2023*. https://www.microsoft.com/en-us/research/publication/recommending-root-cause-and-mitigation-steps-for-cloud-incidents-using-large-language-models/
[6] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. Cherrypick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*.
[7] Muhammad Bilal, Marco Canini, and Rodrigo Rodrigues. [n. d.]. Finding the Right Cloud Configuration for Analytics Clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*.
[8] Muhammad Bilal, Marco Serafini, Marco Canini, and Rodrigo Rodrigues. 2020. Do the Best Cloud Configurations Grow on Trees? An Experimental Evaluation of Black Box Algorithms for Optimizing Cloud Workloads. *Proc. VLDB Endow.* (2020).
[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. https://doi.org/10.48550/ARXIV.2005.14165
[10] Zhen Cao, Geoff Kuenning, and Erez Zadok. [n. d.]. Carver: Finding Important Parameters for Storage System Tuning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*.
[11] Stefano Cereda, Stefano Valladares, Paolo Cremonesi, and Stefano Doni. 2021. CGPTuner: A Contextual Gaussian Process Bandit Approach for the Automatic Tuning of IT Configurations under Varying Workload Conditions. *Proc. VLDB Endow.* 14, 8 (oct 2021), 1401–1413. https://doi.org/10.14778/3457390.3457404
[12] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. https://doi.org/10.48550/ARXIV.1901.02860
[13] J. Dean and L. A. Barroso. 2013. The Tail at Scale. *Communications of ACM* 56, 2 (2013), 74–80.
[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805
[15] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. 2009. Tuning Database Configuration Parameters with ITuned. *Proc. VLDB Endow.* 2, 1 (aug 2009), 1246–1257. https://doi.org/10.14778/1687627.1687767
[16] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139
[17] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayantara Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Yuan He, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
[18] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. https://doi.org/10.48550/ARXIV.2203.03850
[19] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. https://doi.org/10.48550/ARXIV.2009.08366

[20] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. https://doi.org/10.48550/ARXIV.2004.10964

[21] Haochen He, Zhouyang Jia, Shanshan Li, Yue Yu, Chenglong Zhou, Qing Liao, Ji Wang, and Xiangke Liao. 2022. Multi-Intention-Aware Configuration Selection for Performance Tuning. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1431–1442. https://doi.org/10.1145/3510003.3510094

[22] Chiaki Ishikawa, Ken Sakamura, and Mamoru Maekawa. 1982. Dynamic tuning of operating systems. In *Operating Systems Engineering*, Mamoru Maekawa and Laszio A. Belady (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 119–142.

[23] Ajaykrishna Karthikeyan, Nagarajan Natarajan, Gagan Somashekhar, Lei Zhao, Ranjita Bhagwan, Rodrigo Fonseca, Tatiana Racheva, and Yogesh Bansal. 2023. SelfTune: Tuning Cluster Managers. In *To Appear in NSDI*. https://www.microsoft.com/en-us/research/uploads/prod/2022/10/SelfTune_NSDI2023_Cameraready.pdf.

[24] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. [n. d.]. Practical Guide to Controlled Experiments on the Web: Listen to Your Customers Not to the Hippo. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[25] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. https://doi.org/10.48550/ARXIV.1910.13461

[26] Zhao Lucis Li, Chieh-Jan Mike Liang, Wenjia He, Lianjie Zhu, Wenjun Dai, Jin Jiang, and Guangzhong Sun. 2018. Metis: Robustly Tuning Tail Latencies of Cloud Systems. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 981–992. https://www.usenix.org/conference/atc18/presentation/li-zhao

[27] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. *Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis*.

[28] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*.

[29] Per-Olov Östberg, Thang Le Duc, Paolo Casari, Rafael García Leiva, Antonio Fernández Anta, and Jörg Domaschka. 2020. *Application Optimisation: Workload Prediction and Autonomous Autoscaling of Distributed Cloud Applications*. Springer International Publishing, Cham, 51–68. https://doi.org/10.1007/978-3-030-39863-7_3

[30] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*.

[31] Ariel Rabkin and Randy Katz. 2011. Static Extraction of Program Configuration Options. In *Proceedings of the 33rd International Conference on Software Engineering* (Waikiki, Honolulu, HI, USA) *(ICSE '11)*. Association for Computing Machinery, New York, NY, USA, 131–140. https://doi.org/10.1145/1985793.1985812

[32] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. [n. d.]. Improving language understanding by generative pre-training. ([n. d.]).

[33] Alan Ramponi and Barbara Plank. 2020. Neural Unsupervised Domain Adaptation in NLP—A Survey. https://doi.org/10.48550/ARXIV.2006.00632

[34] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis. 1998. Automatic TCP Buffer Tuning. *SIGCOMM Comput. Commun. Rev.* 28, 4 (oct 1998), 315–323. https://doi.org/10.1145/285243.285292

[35] Dennis Shasha. 1997. Lessons from Wall Street: Case Studies in Configuration, Tuning, and Distribution. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data* (Tucson, Arizona, USA) *(SIGMOD '97)*. Association for Computing Machinery, New York, NY, USA, 498–501. https://doi.org/10.1145/253260.253368

[36] G. Somashekar, A. Suresh, S. Tyagi, V. Dhyani, K. Donkada, A. Pradhan, and A. Gandhi. 2022. Reducing the Tail Latency of Microservices Applications via Optimal Configuration Tuning. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE Computer Society, Los Alamitos, CA, USA, 111–120. https://doi.org/10.1109/ACSOS55765.2022.00029

[37] Akshitha Sriraman and Thomas F. Wenisch. 2018. Mtune: AutoTuned Threading for OLDI Microservices. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI 2018)*. USENIX Association, USA, 177–194.

[38] Immanuel Trummer. 2021. The case for NLP-enhanced database tuning: towards tuning tools that" read the manual". *Proceedings of the VLDB Endowment* 14, 7 (2021), 1159–1165.

[39] Immanuel Trummer. 2022. DB-BERT: a database tuning tool that "reads the manual". In *SIGMOD*.

[40] Valentin Bartenev (F5 networks). 2015. Thread Pools in Nginx Boost Performance 9x! https://www.nginx.com/blog/thread-pools-boost-performance-9x/. Online; accessed November, 2022.

[41] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1009–1024. https://doi.org/10.1145/3035918.3064029

[42] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

[43] Xin Wang, Yasheng Wang, Fei Mi, Pingyi Zhou, Yao Wan, Xiao Liu, Li Li, Hao Wu, Jin Liu, and Xin Jiang. 2021. SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation. https://doi.org/10.48550/ARXIV.2108.04556

[44] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[45] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. 2013. Do Not Blame Users for Misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (Farminton, Pennsylvania) *(SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 244–259. https://doi.org/10.1145/2517349.2522727

[46] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chenjie Xu, Chao Ji, and Wenyun Zhao. 2018. Poster: Benchmarking Microservice Systems for Software Engineering Research. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*.

[47] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*. 338–350.