A Related Works

In this section, we introduce related studies on similar tasks that might be confused with code optimization, as well as a number of related surveys that investigate topics similar to this study.

A.1 LM-related tasks

LMs have been employed for a number of code-related tasks, including code generation, refactoring and repair, and performance modeling, described as follows.

LM-based code generation translates a program specification into program code or executable binaries [4]. For example, Li et al. [16] presented AlphaCode, a novel LM-based system for code generation that achieves competitive performance based on natural language programming problems. He et al. [9] designed CoCoST to use LM in real-world programming scenarios by mimicking human coding processes, such as online searching and test case creation. Moreover, Liu et al. [17] introduced EoH, combining LMs and evolutionary computation methods for automatic algorithm design.

LM-based code refactoring, similar to code editing [8], code rewriting [6] or code transformation [19], focuses on restructuring the design, structure, and components of code, without explicitly addressing performance metrics [5]. For instance, Li et al. [14] proposed CodeEditor, which generates code mutations from real-world code and learns to edit them back to the original, thereby capturing effective refactoring patterns. Cummins et al. [6] emphasized generating refactoring transformations rather than directly rewriting code, allowing for easier inspection, debugging, and validation.

Additionally, efforts have been focused on **LM-based code repair**, where LMs are used to identify and fix defects, bugs, or errors in code to restore its intended functionality and ensure correctness [27]. As examples, Jin et al. [13] employed a retrieval-based prompt augmentation technique and task-oriented fine-tuning in InferFix, leveraging bug-type annotations and extended source code context to enhance bug detection and repair processes, and Zhang et al. [28] leveraged code representation through abstract syntax trees (ASTs) and employs spectrum-based fault localization to generate effective patches, resolving software issues efficiently.

Other works have targeted **LM-based code performance modeling**, which involves analyzing and understanding the performance characteristics of code, serving as a potential component of code optimization [2]. Among others, Wang et al. [23] introduced PerfSense, which leverages LM agents and prompt chaining techniques to analyze source code and classify configurations as performance-sensitive or insensitive, thereby enhancing performance analysis. Besides, in the work by Nichols et al. [18], an LM was fine-tuned on a curated dataset containing HPC and scientific codes, and is employed to predict the relative performance impact of changes made to the source code.

While the above studies are related to code optimization to some extent, they are not included in this survey because their primary focuses are different from our definition of code optimization[1]. This survey aims to address this gap by focusing exclusively on the application of LMs for optimizing the performance of existing code or programs.

A.2 Related surveys

To the best of our knowledge, the most relevant surveys to ours are those focusing on **LMs for evolutionary algorithms (EAs)**, which involve solving optimization problems by iteratively

[1] Note that although some studies do not explicitly define their task as code optimization, we have included them in this survey if in any steps the performance of existing code is improved, typical examples will be some studies for code editing, code refinement, and some code generation studies with iterative performance improvements.

improving a population of candidates through mechanisms inspired by biological evolution, such as selection, mutation, and crossover. Specifically, Huang et al. [11] discussed the roles of LMs in solving complex optimization problems through evolutionary mechanisms and provided future directions in this application, and Wu et al. [26] explored the collaborative strengths of LMs and EAs in three aspects: LM-enhanced EA, EA-enhanced LM, and their applications in optimization tasks. However, these surveys only cover a broad range of optimization problems, not specializing in enhancing the performance of existing programs.

Besides, previous surveys have typically focused on **code optimization using traditional methods** involving compilers, machine learning (ML), and deep learning (DL) techniques [15, 24, 1, 22, 21]. For instance, Wang et al. [24] discussed the integration of ML into compiler optimization, highlighting its evolution over the past 50 years and its current status as a mainstream research area. More recently, Wan et al. [22] conducted a comprehensive survey on deep learning for code intelligence, covering aspects such as code representations, DL techniques, application tasks, and public datasets.

Moreover, a few surveys have investigated the utilization of **LMs for code-related tasks**, including general code intelligence [30, 25], code generation [12, 3], and code repair [27]. In particular, Zhang et al. [29] reviewed the evolution of LM-based techniques for code processing tasks from pre-trained Transformer to advanced retrieval and agentic approaches, and Chen et al. [3] survey current methods and metrics for assessing code generation capabilities, identifying their limitations and proposing directions for future improvements.

Additionally, there have been several surveys on the use of **LMs in general software engineering tasks**. For example, Hou et al. [10] provided a comprehensive understanding of how LMs can optimize SE processes and outcomes, addressing gaps in existing research. Similarly, Fan et al. [7] provided a comprehensive overview of the current state of research and future directions in LM-based software engineering, and Shi et al. [20] proposed a vision for the future of LLM4SE that includes a roadmap for research directions aimed at improving efficiency and reducing carbon emissions associated with LMs.

Despite the existence of comprehensive surveys, there is a notable knowledge gap when it comes to a focused survey on the use of LMs specifically for code optimization. Therefore, this survey addresses this gap by systematically reviewing the current state of research and applications of LMs in code optimization, identifying the strengths and limitations of existing approaches, and providing guidance for future research directions.

References

[1] K Manasvi Bhat, Pratiksha P Anchalia, Rushali Mohbe, and A Parkavi. A survey of machine learning and deep learning techniques for compiler optimization. *International Journal of Research in Engineering, Science and Management*, 2019.

[2] Junkai Chen, Zhiyuan Pan, Xing Hu, Zhenhao Li, Ge Li, and Xin Xia. Reasoning runtime behavior of a program with llm: How far are we? In *IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 140–152, 2025.

[3] Liguo Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, et al. A survey on evaluating large language models in code generation tasks. *arXiv:2408.16498*, 2024.

[4] Tristan Coignion, Clément Quinton, and Romain Rouvoy. A performance study of llm-generated code on leetcode. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 79–89, 2024.

[5] Jonathan Cordeiro, Shayan Noei, and Ying Zou. An empirical study on the code refactoring capability of large language models. *arXiv:2411.02320*, 2024.

[6] Chris Cummins, Volker Seeker, Jordi Armengol-Estapé, Aram H Markosyan, Gabriel Synnaeve, and Hugh Leather. Don't transform the code, code the transforms: Towards precise code rewriting using llms. *NeurIPS*, 2024.

[7] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. Large language models for software engineering: Survey and open problems. In *International Conference on Software Engineering: Future of Software Engineering, ICSE-FoSE 2023*, pages 31–53. IEEE, 2023.

[8] Priyanshu Gupta, Avishree Khare, Yasharth Bajpai, Saikat Chakraborty, Sumit Gulwani, Aditya Kanade, Arjun Radhakrishna, Gustavo Soares, and Ashish Tiwari. Grace: Language models meet code edits. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE*, pages 1483–1495. ACM, 2023.

[9] Xinyi He, Jiaru Zou, Yun Lin, Mengyu Zhou, Shi Han, Zejian Yuan, and Dongmei Zhang. Cocost: Automatic complex code generation with online searching and correctness testing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19433–19451, 2024.

[10] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 2023.

[11] Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. When large language model meets optimization. *arXiv:2405.10098*, 2024.

[12] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. 2024.

[13] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE*, pages 1646–1656. ACM, 2023.

[14] Jia Li, Ge Li, Zhuo Li, Zhi Jin, Xing Hu, Kechi Zhang, and Zhiyi Fu. Codeeditor: Learning to edit source code with pre-trained models. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 32:1–22, 2023.

[15] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. The deep learning compiler: A comprehensive survey. *Transactions on Parallel and Distributed Systems*, 32:708–727, 2020.

[16] Yujia Li, David Choi, Junyoung Chung, and et al. Competition-level code generation with alphacode. *Science*, 378:1092–1097, 2022.

[17] Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Forty-first International Conference on Machine Learning*, 2024.

[18] Daniel Nichols, Aniruddha Marathe, Harshitha Menon, Todd Gamblin, and Abhinav Bhatele. Modeling parallel programs using large language models. 2023.

[19] Marek Palkowski and Mateusz Gruzewski. Gpt-driven source-to-source transformation for generating compilable parallel cuda code for nussinov's algorithm. *Electronics*, 13:488, 2024.

[20] Jieke Shi, Zhou Yang, and David Lo. Efficient and green large language models for software engineering: Vision and the road ahead. 2024.

[21] Qiushi Sun, Zhirui Chen, Fangzhi Xu, and et. al. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv:2403.14734*, 2024.

[22] Yao Wan, Zhangqian Bi, Yang He, Jianguo Zhang, Hongyu Zhang, Yulei Sui, Guandong Xu, Hai Jin, and Philip Yu. Deep learning for code intelligence: Survey, benchmark and toolkit. *Computing Surveys (CSUR)*, 2024.

[23] Zehao Wang, Dong Jae Kim, and Tse-Hsun Chen. Identifying performance-sensitive configurations in software systems through code analysis with LLM agents. 2024.

[24] Zheng Wang and Michael O'Boyle. Machine learning in compiler optimization. *Proceedings of the IEEE*, 106:1879–1901, 2018.

[25] Man-Fai Wong, Shangxin Guo, Ching Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. Natural language generation and understanding of big code for ai-assisted programming: A review. *Entropy*, 25:888, 2023.

[26] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *arXiv:2401.10034*, 2024.

[27] Quanjun Zhang, Chunrong Fang, Yang Xie, Yuxiang Ma, Weisong Sun, Yun Yang, and Zhenyu Chen. A systematic literature review on large language models for automated program repair. 2024.

[28] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604, 2024.

[29] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. Unifying the perspectives of nlp and software engineering: A survey on language models for code. *arXiv:2311.07989*, 2023.

[30] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. A survey of large language models for code: Evolution, benchmarking, and future trends. 2023.