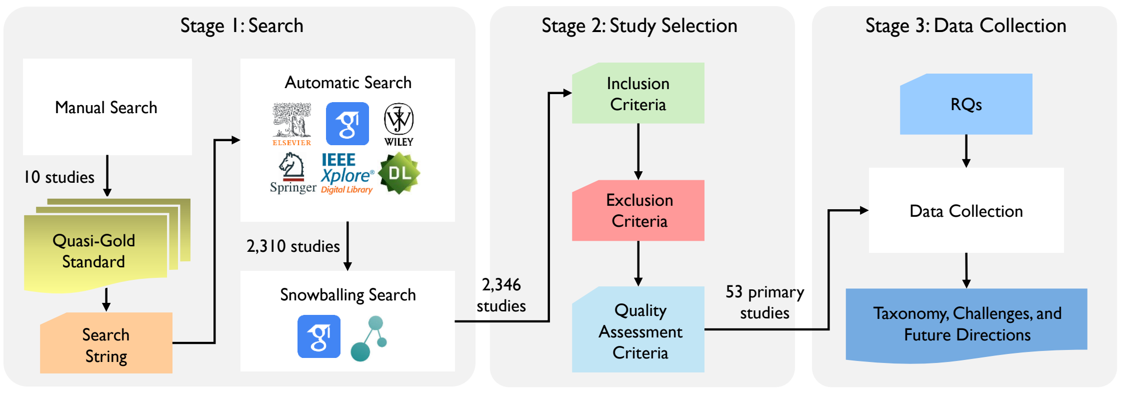

Figure 1: Overview of the survey methodology used in this study.

A Methodology

We now describe the methodology used to collect literature and data and perform the analysis. Specifically, we follow the widely recognized guidelines for SLRs in Software Engineering proposed by Kitchenham and Charters [12], which have also been adopted by numerous SLRs [4, 8, 19, 25, 21]. The methodology comprises three key stages, as illustrated in Figure 1. Note that throughout all stages, we follow the iterative procedures among co-authors of the article:

1. The first author performs initial actions and shares the results with one of the co-authors;
2. The co-author assists the first author with all needed actions and reviews the results;
3. If the two authors make an agreement, the results of all actions are presented and reviewed by all the rest of the co-authors; otherwise, go to (1);
4. If agreements are achieved, go to the next stage, other wise, restart from (1).

In the following sub-sections, we will introduce each stage in detail.

A.1 Search Strategy

A.1.1 Search string

To ensure a comprehensive and accurate search, we began by defining a search string using the "quasi-gold standard" (QGS) [24], an approach commonly used in modern surveys [8, 19, 25, 21, 4]. In particular, we first manually collected 10 high-quality studies based on their impact, citation count, and relevance to the topic of code optimization using LMs. From these QGS studies, we extracted keywords and phrases central to our research objectives and created the first draft search string. Then, we iteratively performed searches using Google Scholar and refined the search string until the sensitivity (%) s of the search is higher than 80% [24], which is computed using the equation as follows:

$$s = \frac{\#\text{ relevant studies searched}}{\text{Total }\#\text{ relevant studies in the QGS}} \times 100 \tag{1}$$

Formally, the search string produced by the QGS methodology is:

Search String

"language models" AND ("code optimization" OR "compiler optimization"OR "optimize code" OR "optimizing code" OR "code performance optimization")

A.1.2 Automatic search

Using the defined search string, we conducted automatic searches on seven popular academic indexing engines [8, 19, 25, 21, 1], as shown in Figure 1, i.e., Google Scholar, IEEE Xplore, ACM Digital Library, Science Direct, Springer, and WILEY.

To ensure maximum coverage, we included both American and British spellings for keywords like "optimizing" and "optimising", and replaced "code" with "program" and "software". Since the field of language models is developing quickly, we regularly updated the search results until November 2024 to ensure the most recent advancements were captured. This process resulted in 2,310 searched studies in total.

A.1.3 Snowballing search

Following the automatic search, we conducted a manual snowballing search based on the 10 QGS studies using Google Scholar and Connected Papers, involving both forward snowballing (identifying papers that cite the QGS) and backward snowballing (reviewing the references cited by the QGS). This is effective to uncover studies that are relevant but might not be captured through keyword searches [8, 25, 21, 1, 20], such as code generation studies that optimize code performance as part of their iterative workflow [15, 6]. In summary, the snowballing process led to 36 extra searched studies.

A.2 Study Selection

A.2.1 Inclusion and exclusion criteria

In the second stage, we filtered duplicate and irrelevant searched studies, and performed rigorous study selection using a list of inclusion and exclusion criteria, following the SLR guideline and previous SLR studies [12]. By adhering to these criteria, we aim to identify a specialized and reliable subset of the current state of research and practice in this area, serving as the primary studies to be reviewed. In particular, the inclusion criteria include:

- ✓ The paper presents a novel code optimization method.
- ✓ The proposed method uses language models.
- ✓ The paper aims to optimize at least one of the non-functional performance metrics of the original code.
- ✓ The paper conducts rigorous experiments to evaluate the proposed method.

Nevertheless, this still led to irrelevant studies to our research topic, therefore, we performed further filtering using the following exclusion criteria:

- ✗ The paper does not use code as input in any part of the pipeline.
- ✗ The paper is focused on improving functionalities of code, or fixing code bugs.
- ✗ The paper is a poster or abstract, without technical details and experimental results.
- ✗ The paper is not written in English.

A.2.2 Quality assessment

Moreover, since many of the primary studies are not peer-reviewed, we conducted a comprehensive quality assessment to examine their quality and to ensure the quality of our survey and the

Table 1: Quality Assessment Criteria (QAC) and scoring metrics. The detailed scoring results can be found at: https://github.com/gjz78910/CodeOpt-SLR

ID	Quality Assessment Criteria	Scoring Metric
QCA_1	How many times has the paper been cited by other studies?	No=0, No more than 10 citations=1, More than 10 citations=2.
QCA_2	Does the paper provide a clear definition of the code optimization problem?	Not provided=0, Implicitly provided=1, Clearly provided=2.
QCA_3	Does the study provide a clear description of the proposed code optimization technique?	Not provided=0, Implicitly provided=1, Clearly provided=2.
QCA_4	Does the study compare the proposed model with other state-of-the-art baseline models?	No=0, Only one baseline=1, More than one baselines =2.
QCA_5	Are the experimental settings such as experimental environments, evaluation metrics, and dataset information described in detail?	Not described=0, Partially described=1, Fully described=2.
QCA_6	Are the key contributions and limitations of the study discussed?	Not discussed=0, Partially discussed=1, Fully discussed=2.
QCA_7	Does the study clearly outline the experimental findings?	Not outlined=0, Implicitly outlined=1, Clearly outlined=2.

reliability of the findings. Specifically, we removed studies with scores equal to or less than one, and Table 1 presents the Quality Assessment Criteria (QAC) we used. As a result, 53 primary studies were selected for the next stage of detailed review and data collection.

A.3 Research Questions and Data Collection

In the third stage, to guide our systematic literature review, we formulated a set of four primary research questions aimed at addressing the key aspects of using LMs for code optimization, to answer this, we identified a few key sub-questions, carefully reviewed the primary studies and collected data items accordingly.

***RQ1:** What were the characteristics of the LMs used for code optimization?*

The first RQ aims to explore the foundational attributes of the LMs employed for code optimization, which includes collecting data about the base models, their sizes, whether they are open source, and how they were pre-trained and fine-tuned.

***RQ2:** How were LMs applied to code optimization tasks?*

This question seeks to identify and categorize the practical roles and applications of LMs in the code optimization pipeline. It involved gathering insights into common challenges addressed by LMs, the specific optimization techniques employed, and the role of LMs within the optimization workflow.

***RQ3:** What programming languages and performance metrics were considered in LM-based code optimization?*

The aim of this question is to provide a comprehensive view of the code optimization problem. To address this, we collected data on the programming languages and performance metrics involved in the primary studies.

***RQ4:** How were the proposed code optimization methods evaluated?*

This final question focuses on understanding the evaluation methodologies used to validate LM-driven code optimization techniques. It involved analyzing the datasets utilized, the application of optimization methods to real-world programs, and the evaluation metrics.

A.4 Statistics of Primary Studies

Figure 2 provides a detailed view of the distribution of the reviewed primary studies across various publication venues. Notably, the largest portion of studies (24 or 45%) were sourced from arXiv

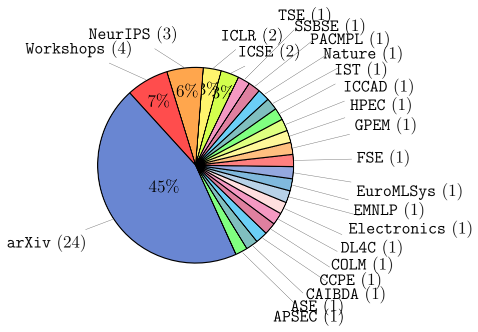

Figure 2: Distribution of publication venues.

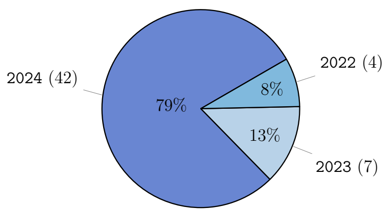

Figure 3: Distribution of primary studies over years (till November 2024).

as preprints, suggesting ongoing and rapidly developing work in this area. Three other significant venues include NeurIPS [9, 13, 17], ICSE [11, 3], and ICLR [18, 10], reflecting the relevance of machine learning and software engineering in the context of code optimization. The remaining studies are spread across 22 conferences and journals (one study each), such as Nature [16], TSE [2], PACMPL [23], and HPEC [5], and four workshops like LAD [22, 7, 14]. This diversity highlights that the topic is addressed not only in AI and SE-specific forums but also in broader scientific and engineering communities.

Moreover, Figure 3 presents the distribution of studies by year, where the majority of studies, 79%, were published in 2024, totaling 42 studies. In comparison, 2023 contributed 13% (seven studies), and 2022 accounted for only 8% (four studies). This significant increase suggests a surge in interest in LM-based code optimization research recently.

References

[1] Sicong Cao, Xiaobing Sun, Ratnadira Widyasari, David Lo, Xiaoxue Wu, Lili Bo, Jiale Zhang, Bin Li, Wei Liu, Di Wu, and Yixin Chen. A systematic literature review on explainability for machine/deep learning-based software engineering research. 2024.

[2] Zimin Chen, Sen Fang, and Martin Monperrus. Supersonic: Learning to generate source code optimizations in C/C++. *IEEE Transactions on Software Engineering (TSE)*, 2024.

[3] Shuzheng Gao, Cuiyun Gao, Wenchao Gu, and Michael Lyu. Search-based llms for code optimization. In *International Conference on Software Engineering (ICSE)*, pages 254–266. IEEE, 2024.

[4] Jingzhi Gong and Tao Chen. Deep configuration performance learning: A systematic survey and taxonomy. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2024.

[5] Zifan Carl Guo and William S. Moses. Enabling transformers to understand low-level programs. In *High Performance Extreme Computing Conference (HPEC)*, pages 1–9. IEEE, 2022.

[6] Xu Han, Qiannan Yang, Xianda Chen, Xiaowen Chu, and Meixin Zhu. Generating and evolving reward functions for highway driving with large language models. *arXiv:2406.10540*, 2024.

[7] Charles Hong, Sahil Bhatia, Altan Haan, Shengjun Kris Dong, Dima Nikiforov, Alvin Cheung, and Yakun Sophia Shao. Llm-aided compilation for tensor accelerators. pages 1–14, 2024.

[8] 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.

[9] Dong Huang, Jianbo Dai, Han Weng, Puzhen Wu, QING Yuhao, Heming Cui, Zhijiang Guo, and Jie Zhang. Effilearner: Enhancing efficiency of generated code via self-optimization. 2024.

[10] Shu Ishida, Gianluca Corrado, George Fedoseev, Hudson Yeo, Lloyd Russell, Jamie Shotton, Joao F Henriques, and Anthony Hu. Langprop: A code optimization framework using large language models applied to driving, 2024.

[11] Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. Jigsaw: Large language models meet program synthesis. In *International Conference on Software Engineering (ICSE)*, pages 1219–1231, 2022.

[12] Barbara Kitchenham and Stuart M. Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[13] Aman Madaan, Niket Tandon, Prakhar Gupta, and et al. Self-refine: Iterative refinement with self-feedback. In *Neural Information Processing Systems (NeurIPS)*, 2023.

[14] Yue Pan, Xiuting Shao, and Chen Lyu. Measuring code efficiency optimization capabilities with aceob, 2025.

[15] Huiyun Peng, Arjun Gupte, Nicholas John Eliopoulos, Chien Chou Ho, Rishi Mantri, Leo Deng, Wenxin Jiang, Yung-Hsiang Lu, Konstantin Läufer, George K Thiruvathukal, et al. Large language models for energy-efficient code: Emerging results and future directions. *arXiv:2410.09241*, 2024.

[16] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[17] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2023.

[18] Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob R. Gardner, Yiming Yang, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. In *The Twelfth International Conference on Learning Representations, ICLR*. OpenReview.net, 2024.

[19] Simin Wang, Liguo Huang, Amiao Gao, Jidong Ge, Tengfei Zhang, Haitao Feng, Ishna Satyarth, Ming Li, He Zhang, and Vincent Ng. Machine/deep learning for software engineering: A systematic literature review. *IEEE Trans. Software Eng.*, 49(3):1188–1231, 2023.

[20] Cody Watson, Nathan Cooper, David Nader-Palacio, Kevin Moran, and Denys Poshyvanyk. A systematic literature review on the use of deep learning in software engineering research. *ACM Trans. Softw. Eng. Methodol.*, 31:32:1–32:58, 2022.

[21] Hanxiang Xu, Shenao Wang, Ningke Li, Kailong Wang, Yanjie Zhao, Kai Chen, Ting Yu, Yang Liu, and Haoyu Wang. Large language models for cyber security: A systematic literature review. 2024.

[22] Haocheng Xu, Haotian Hu, and Sitao Huang. Optimizing high-level synthesis designs with retrieval-augmented large language models. In *IEEE LLM Aided Design Workshop (LAD)*, pages 1–5. IEEE, 2024.

[23] Fangke Ye, Jisheng Zhao, Jun Shirako, and Vivek Sarkar. Concrete type inference for code optimization using machine learning with smt solving. *Proceedings of the ACM on Programming Languages (PACMPL)*, 7:773–800, 2023.

[24] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53:625–637, 2011.

[25] 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.