Efficient and Green Large Language Models for Software Engineering: Vision and the Road Ahead

Jieke Shi, Zhou Yang, and David Lo

School of Computing and Information Systems, Singapore Management University, Singapore

{jiekeshi, zyang, davidlo}@smu.edu.sg

ABSTRACT

Large Language Models (LLMs) have recently shown remarkable capabilities in various software engineering tasks, spurring the rapid development of the Large Language Models for Software Engineering (LLM4SE) area. However, limited attention has been paid to crafting efficient LLM4SE solutions that demand minimal time and memory resources, as well as green LLM4SE solutions that reduce energy consumption and carbon emissions.

This 2030 Software Engineering position paper aims to redirect the focus of the research community towards the efficiency and greenness of LLM4SE, while also sharing potential research directions to achieve this goal. It commences with a brief overview of the significance of LLM4SE and highlights the need for efficient and green LLM4SE solutions. Subsequently, the paper presents a vision for a future where efficient and green LLM4SE revolutionizes the software engineering tool landscape, benefiting various stakeholders, including industry, individual practitioners, and society. The paper then delineates a roadmap for future research, outlining specific research paths and potential solutions for the research community to pursue. While not intended to be a definitive guide, the paper aims to inspire further progress, with the ultimate goal of establishing efficient and green LLM4SE as a central element in the future of software engineering.

KEYWORDS

LLM4SE, Efficient, Green, Vision, and Roadmap

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have spurred their widespread adoption in various fields, notably within software engineering [18, 29, 76, 89, 93]. Trained on massive amounts of data, LLMs have shown their prowess in understanding, processing, and generating code written in various programming languages, with exceptional performance across a spectrum of code-related tasks, ranging from code generation [9, 11, 83] and code summarization [2, 3, 41] to vulnerability detection [55, 71, 94], program repair [33, 34, 82], and more [14, 17, 19, 59, 90]. This success has fueled the rapid growth of the Large Language Models for Software Engineering (LLM4SE) area, which has recently become one of the most active areas within software engineering.

To date, certain LLM4SE solutions have been translated into real-world applications, such as GitHub Copilot [22], leveraging OpenAI's GPT-4 [1] to aid developers in writing code, and Google VirusTotal Code Insight [60], which uses the Sec-PaLM model [13] for malware analysis. These applications have brought a new level of automation to the software development process, significantly improving developers' productivity and software product quality. However, due to the computationally-intensive [28] and carbon-demanding [40, 58] nature of LLMs, most of these LLM4SE applications are facilitated by large companies with abundant computational resources and carbon credits (i.e., allowable amount of carbon emissions) [23, 77], which are not readily available to the broader software engineering community, including startups and individual developers. Before the benefits of LLMs become accessible to all software engineering practitioners, several key challenges remain in current LLM4SE solutions, most notably their unsatisfactory efficiency (i.e., time and memory resources consumed) and greenness (i.e., energy consumption and carbon emissions).

Need for Efficient LLM4SE. Efficient LLM4SE involves developing and operating LLM4SE solutions with minimal computational resources, particularly regarding time and memory. Today, developing LLM4SE solutions typically involves a time-consuming LLM training process, taking weeks to months [53, 57, 73]. This lengthy training process, along with the subsequent operation of LLMs, demands substantial computational resources from thousands of high-performance devices like GPUs [26] and thousands of gigabytes of memory, translating into high costs. For example, the training of OpenAI's GPT-3 [7] cost over $4 million [38]. Such prohibitive costs often hinder software practitioners from independently developing and operating LLM4SE solutions, compelling them to rely on cloud services offered by large or well-funded companies. However, this reliance also incurs significant expenses, peaking at $200,000 per month [38], and raises privacy concerns due to the necessity of sharing sensitive data with service providers [47, 84].

Even setting aside cost and privacy concerns, users often have a suboptimal experience with cloud-hosted LLM4SE solutions due to high response latency caused by slow LLM inference and network delays [21, 63, 64, 78]. While deploying LLMs on users' devices can mitigate latency [47, 63, 64], accomplishing this task remains challenging without efficient LLM4SE solutions, as current LLMs demand substantial memory beyond what many personal devices, such as laptops, can accommodate [64, 78]. Beyond all the above,

the efficiency of LLM-generated code also emerges as a critical concern, as it can lag behind human-written code in terms of execution time and memory usage [30, 74]. Consequently, software products developed using LLM4SE solutions may exhibit poor performance, reducing their competitiveness in the marketplace. Building efficient LLM4SE solutions can address the above challenges and make LLM4SE solutions more accessible to all software engineering practitioners, regardless of their computational resources.

Need for Green LLM4SE. Green LLM4SE aims to minimize the environmental impact of LLM4SE solutions, with a particular focus on reducing energy consumption and carbon emissions. Training LLMs typically demands substantial energy, leading to high carbon emissions. For example, training LLaMA [73] consumes 2,638,000 kilowatt-hours of electricity, equivalent to the annual electricity usage of 1,648 Danes, and emits 1,015 tons of carbon dioxide, equivalent to the annual emissions of 92 Danes [50]. The energy consumption of LLM inference is also considerable. Each ChatGPT inference consumes 2.9 watt-hours (Wh) of electricity, about ten times the 0.3 Wh consumption of a Google search [15, 65]. Additionally, a recent study shows that the energy consumption of LLM-generated code can far exceed that of human-written code [74], potentially making software products developed using LLM4SE solutions less green. These high energy consumption and carbon emissions not only make LLM4SE solutions expensive to develop and operate but also contribute to climate change and environmental degradation. Therefore, it is essential to build green LLM4SE solutions to mitigate these negative impacts.

Need for Synergy between Efficient and Green LLM4SE. Efficient and green LLM4SE solutions are closely related. For example, Shi et al. [63] show that reducing the memory usage of LLMs can lead to their lower energy consumption. Similarly, Wei et al. [78] apply quantization to LLM4SE solutions to reduce carbon emissions while also equipping them with faster inference speed and lower memory usage. These examples suggest that developing efficient LLM4SE solutions can lead to green ones, and vice versa. However, while they are related, they are not identical; thus, achieving one does not always guarantee the full optimization of the other. For instance, Shi et al. [64] propose an efficient LLM4SE solution by compressing LLMs. Nonetheless, their compressed LLMs still have considerable energy consumption and carbon emissions, leaving room for improvement in their greenness, as demonstrated in their subsequent work [63]. Thus, we advocate the synergy of efficient and green LLM4SE solutions, achieving the best of both worlds to make LLM4SE solutions both accessible and sustainable.

2 CURRENT DEVELOPMENT

This section provides an overview of the current state of LLM4SE solutions regarding efficiency and greenness. Here, we group the literature based on the key techniques used.

Model Compression. Model compression aims to reduce the size of a model, thereby optimizing other relevant metrics such as inference latency, memory usage, and energy consumption. As the pioneering work in compressing code LLMs, Shi et al. [64] propose Compressor, which uses a genetic algorithm-based method for simplifying models and applies knowledge distillation to compress LLMs to 3 megabytes (MB). Their compressed LLMs improve

inference latency by 4.23× on average. Subsequently, Shi et al. [63] also present Avatar, which finds Pareto-optimal compressed LLMs that balance model size, inference latency, energy consumption, and carbon footprint, instead of solely focusing on model size like Compressor. The models produced by Avatar significantly reduce energy consumption (up to 184× less), carbon footprint (up to 157× less), and inference latency (up to 76× faster). Additionally, Su and McMillan [66] use knowledge distillation on GPT-3.5 to obtain a smaller and more efficient model for code summarization. Wei et al. [78] and Kaushal et al. [35] use quantization and low-rank decomposition to compress LLMs for code generation, respectively, both achieving significant efficiency improvements.

Dynamic Inference. Dynamic inference manipulates the inference process of LLMs to improve model efficiency. An early work by Sun et al. [68] introduces an early rejection mechanism to reject prompts that cannot generate useful completions during inference. This mechanism saves 23.3% of the computational cost of LLMs in code completion. After that, Sun et al. [67] discover that 54.4% of tokens can be accurately generated using only the first layer of the GPT-2 model. They develop a method capable of skipping an average of 1.7 layers out of 16 in the models, resulting in a speedup of 11.2% in code completion. Similarly, Grishina et al. [24] show that only 3 of the 12 layers of CodeBERT are sufficient for vulnerability detection with a 3.3× speedup in fine-tuning.

Input Pruning. Pruning the input tokens can improve the efficiency of LLMs, since their inference time is affected by the input length. Zhang et al. [91] introduce DietCode, which identifies statements and tokens with the highest attention values, resulting in a 40% reduction in the computational cost of CodeBERT inference. In addition, Hidvégi et al. [27] optimize prompts to reduce token cost by 62% in program repair tasks. Shi et al. [61] simplify input for LLMs, resulting in up to 40% reduction in inference time.

Parameter-Efficient Fine-Tuning (PEFT). PEFT aims to improve the training efficiency of LLMs. During training, it updates only a small set of LLM parameters while freezing the rest, thereby greatly reducing computational and memory costs. Researchers have applied PEFT to LLMs for various software engineering tasks [43, 44, 62], such as code generation [81, 95], code review [48], and clone detection [5]. These studies show that PEFT can significantly reduce the training time and memory consumption of LLMs.

Others. In addition to the above techniques, several studies do not provide a specific technique but focus on the efficiency and greenness of the code generated by LLM4SE solutions. The earliest work is by Liu et al. [45], which investigates the code quality of ChatGPT-generated programs and uncovers efficiency issues with them. Recent studies by Huang et al. [30] and Vartziotis et al. [74] analyze the efficiency and energy consumption of LLM-generated code, respectively. They find that LLM-generated code often lags behind human-written code in efficiency, while its energy consumption can significantly exceed that of human-written code.

3 VISION

This section presents a vision of the future of efficient and green LLM4SE. We outline the key facets of this vision from the perspectives of *industry*, *individual practitioners*, and *society*, highlighting the transformative potential of efficient and green LLM4SE.

For Industry. Advances in efficient and green LLM4SE will facilitate the development of _Low-cost and Low-latency Software Engineering Tools_. The existing software engineering tool landscape is dominated by heavyweight solutions that are computationally expensive and energy-intensive. Companies must not only invest in large amounts of computing resources such as GPUs to develop and run these tools, but also have to bear the high cost of the associated energy consumption (electricity, cooling, etc.) and carbon emissions, requiring the purchase of many carbon credits to compensate for the environmental impact [37]. These costs limit the existing tools' profitability. For example, Microsoft's GitHub Copilot currently loses $20 per user per month instead of making a profit [86]. Moreover, these tools often have high latency, often resulting in a poor user experience [21, 78].

By adopting efficient and green LLM4SE solutions, companies can replace existing imperfect tools with upgraded ones that are faster, greener, and require fewer resources to develop and operate. Not only can these upgraded tools be used internally by companies to streamline their software development processes, but they can also be offered as Software as a Service (SaaS) solutions to other companies and developers, giving them access to cutting-edge software engineering tools at reduced costs and faster response times, all without requiring a large investment in computing resources. In addition, efficient and green LLM4SE solutions can diminish the entry barriers for software engineering tools, presenting significant opportunities for Small and Medium Enterprises (SMEs) and startups, which typically face resource constraints while striving to improve their development workflows. This fosters a more equitable environment in the software industry, allowing SMEs and startups to compete on par with larger companies. Overall, for the software industry, adopting efficient and green LLM4SE will revolutionize the software engineering tool landscape, making it more affordable and accessible to all stakeholders within the industry.

For Individual Software Practitioners. The advancement of efficient and green LLM4SE can pave the way for the emergence and widespread adoption of _Private, Personalized, Trusted, and Collaborative Software Engineering Assistants_. These new assistants will transcend current tools such as GitHub Copilot [22], which are confined to one single task like code suggestion, rely on cloud infrastructure, pose potential privacy risks [54, 85], and are challenging or costly to customize [12]. With minimal computational resources and energy consumption, these new assistants can be seamlessly integrated into software practitioners' local development environments. They remain entirely private to the practitioners, securing their intellectual property and sensitive information from various threats [16, 32, 54, 85]. This integration facilitates real-time interaction between software practitioners and their assistants with minimal latency, enabling them to provide instant feedback and suggestions to practitioners as they write code, test software, or engage in other software engineering activities. Such real-time interactions can occur without an Internet connection, which is particularly crucial in regions of developing countries with limited access to high-speed Internet. Furthermore, this interaction can consume less computational resources and energy than today, making it financially viable for individual practitioners to utilize these assistants on personal devices like laptops.

Moreover, the assistants would deliver personalized assistance for various software engineering tasks, tailored to individual preferences and project requirements. Such personalization benefits from the efficient and green nature of the assistants, allowing it to be accomplished with minimal computational resources and energy consumption. Following such personalization, the assistants will furnish more precise and pertinent assistance to software practitioners, thereby catalyzing trust between software practitioners and their assistants. This trust will enhance the collaboration between software practitioners and assistants, enabling them to work together more effectively to improve productivity, software quality, and the overall development experience [47].

For Society. Efficient and green LLM4SE will play a critical role in fostering _Better Environmental Sustainability in Software Engineering_. The software industry is a significant contributor to global carbon emissions and energy consumption, with data centers and cloud computing facilities, often used for LLM development and operations, being major responsible parties [6, 80]. Embracing efficient and green LLM4SE solutions can mitigate these environmental impacts. These new solutions promise to reduce the energy consumption and carbon emissions associated with LLM development and operation, thereby reducing reliance on data centers and cloud computing facilities and promoting greener practices in the software industry. In addition, the code produced by these solutions will be more resource-efficient and environmentally sustainable, consuming less energy during execution and incurring fewer carbon emissions than current software products.

Transitioning to efficient and green LLM4SE will not only benefit the environment, but will also align with the growing societal emphasis on environmental sustainability. Large technology companies can achieve sustainability goals such as carbon neutrality [69, 75], while smaller companies and individual developers can share more carbon credits, facilitating their rapid growth [37]. By promoting efficient and green LLM4SE, the software industry can demonstrate its commitment to environmental responsibility and contribute to a more sustainable future for all.

4 ROADMAP

This section outlines a roadmap to achieve efficient and green LLM4SE. We identify several specific research paths and potential solutions that the research community can pursue.

A Comprehensive Benchmark. Prior to embarking on the development of LLM4SE solutions, it is essential to establish a comprehensive benchmark dataset to evaluate their efficiency and greenness. While many existing benchmarks, such as CodeXGLUE [49] and CoderEval [88], evaluate the effectiveness (e.g., accuracy) of LLMs in certain software engineering tasks, very few specifically target the efficiency and greenness of LLM4SE. A few studies have collected datasets to evaluate the efficiency or energy consumption of the code generated by LLMs, such as EffiBench [30], and the datasets constructed by Liu et al. [46] and Vartziotis [74]. Nevertheless, these datasets primarily focus on evaluating the generated code rather than the LLMs, and thus are not comprehensive enough. Consequently, researchers and practitioners lack a comprehensive understanding of the current landscape of LLM4SE development in terms of efficiency and greenness, making it difficult for them to

compare different LLM4SE solutions and make informed decisions about which one to adopt. We suggest that the research community prioritize curating a diverse set of tasks and datasets representative of real-world software engineering applications to benchmark the efficiency and greenness of LLM4SE solutions. These efforts will lay the foundation for future advances in the area.

More Efficient Training Methods. Training LLMs is computationally intensive and time-consuming, which is a major bottleneck in the development of LLM4SE. Some studies have used PEFT-based methods to speed up the training of LLMs, as introduced in section 2. However, there are more efficient training methods that can be applied to LLM4SE but have not been well studied, such as data and model parallelism [92], pipeline parallelism [36], using better optimizers [87], etc. So far, no study has systematically and comprehensively evaluated the effectiveness of these methods on LLM4SE solutions. Therefore, we suggest that researchers try to apply these methods and assess their effectiveness in the context of LLM4SE. Then, we can identify the most effective methods and the unique challenges in developing LLM4SE solutions, and thereby propose new training acceleration methods tailored to LLM4SE.

Better Compression Techniques. Popular model compression techniques, including knowledge distillation and quantization, have shown promise in compressing LLMs for efficient and green LLM4SE, as introduced in Section 2. However, further exploration is still needed to optimize the efficiency of LLMs to a satisfactory extent, aiming at response times in the range of a few tens of milliseconds [4] and memory consumption of less than 50 MB [70]. Current compression techniques may fall short of these requirements, especially for very large LLMs like CodeLlama [51], which can range from tens to hundreds of gigabytes in size. Therefore, there is a pressing need to propose novel compression techniques tailored to these very large LLMs in order to achieve the desired efficiency. Furthermore, while knowledge distillation has been the primary focus of current compression efforts for LLM4SE solutions [63, 64, 66], other techniques like quantization and pruning have received relatively less attention, with fewer studies available. Hence, we advocate for increased research efforts to explore the potential of quantization and pruning techniques alongside the innovation of new knowledge distillation methods. Moreover, combining these techniques can also be investigated to achieve better efficiency and greenness improvements for LLM4SE solutions.

Improved Inference Acceleration. Similar to training, the inference process of LLMs is computationally demanding and time-consuming, presenting another bottleneck in adopting LLM4SE. While some existing studies have proposed dynamic inference methods to accelerate LLM inference [67, 68], there remains a lack of comprehensive exploration in this area. For instance, cascade inference strategies, which organize small and large LLMs in a cascading manner and adaptively select suitable ones instead of relying solely on slow and massive LLMs for each query, could be investigated for LLM4SE. Moreover, certain studies have suggested the utilization of non-autoregressive decoding [20, 25] and speculative decoding [8, 39] to accelerate LLM inference, which are potentially applicable to LLM4SE solutions [52]. We suggest that researchers systematically evaluate the effectiveness of these methods in the context of LLM4SE and devise more efficient inference acceleration

techniques tailored to LLM4SE solutions. Additionally, a recent study by Wei et al. [79] leverages code completion engines to filter out infeasible tokens generated by LLMs, thereby enhancing efficiency in utilizing LLMs for program repair. This study underscores the potential of integrating LLMs with existing program analysis tools to enhance efficiency and sustainability in LLM4SE solutions, representing another promising avenue for exploration.

Program Optimization. One area not touched in LLM4SE is optimizing the programs built for LLM inference. The inference pipeline of LLMs is typically constructed using human-written code, which may not fully exploit specific hardware features such as CPU/GPU instructions for parallel computing [42]. To address this, researchers can develop program transformation tools that automatically transform LLM inference code to take advantage of specific backend libraries or hardware features [52]. In addition, direct optimization of the LLM inference code by eliminating dead code [72] and tuning compiler optimization flags [10] can also improve the efficiency and energy consumption of LLM4SE solutions. We recommend that the software engineering community further explore these techniques to help LLM inference achieve better efficiency and lower environmental impact in LLM4SE solutions.

Furthermore, with respect to the code generated by LLMs, there is no specific method in the literature for improving its efficiency and greenness. However, some existing techniques can be used to achieve this goal. We suggest that researchers collect the efficient code snippets of human developers and use them as training data to fine-tune LLMs, which can equip them with the ability to generate efficient and green code that matches human coding practices. In addition, integrating code optimization techniques into the code generation process of LLMs holds promise for refining the generated code to improve its efficiency and environmental sustainability during execution. These techniques include strategies such as dead code elimination [72], loop unrolling [31], and constant folding [56]. By integrating these strategies, the efficiency and greenness of the generated code can be improved, which is reflected in the reduced execution time, memory usage, and energy consumption of the generated code in real-world software products, which can further contribute to achieving efficient and green LLM4SE solutions.

5 CONCLUSION AND FUTURE WORK

As LLMs gain traction in the software engineering community, concerns about their efficiency (i.e., time and memory resources consumed) and greenness (i.e., energy consumption and carbon emissions) have escalated. This paper provides a brief overview of the current landscape of LLM4SE solutions, focusing on their efficiency and greenness while highlighting the associated challenges and opportunities in this emerging field. We then sketch a future vision to provide insights into the potential benefits of adopting efficient and green LLM4SE solutions for industry, individual practitioners, and society. To realize this vision, we propose a roadmap for future research, outlining specific directions and potential solutions that the research community can pursue. Through these efforts, we aim to motivate more people to join and contribute to the LLM4SE research journey, with the goal of fostering a new era of software engineering where LLM4SE solutions are not only effective, but also efficient and environmentally sustainable.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Toufique Ahmed and Premkumar Devanbu. 2022. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–5.

[3] Toufique Ahmed, Kunal Suresh Pai, Premkumar Devanbu, and Earl T Barr. 2024. Automatic semantic augmentation of language model prompts (for code summarization). In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 1004–1004.

[4] Gareth Ari Aye and Gail E. Kaiser. 2020. Sequence Model Design for Code Completion in the Modern IDE. arXiv:2004.05249 [cs.SE]

[5] Shamil Ayupov and Nadezhda Chirkova. 2022. Parameter-Efficient Finetuning of Transformers for Source Code. arXiv:2212.05901 [cs.CL]

[6] Lotfi Belkhir and Ahmed Elmeligi. 2018. Assessing ICT global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production* 177 (2018), 448–463.

[7] Tom Brown, Benjamin Mann, and Nick et al. Ryder. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[8] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318* (2023).

[9] J. Chen, X. Hu, Z. Li, C. Gao, X. Xia, and D. Lo. 2024. Code Search Is All You Need? Improving Code Suggestions With Code Search. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 857–857. https://doi.ieeecomputersociety.org/

[10] Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. 2021. Efficient compiler autotuning via bayesian optimization. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1198–1209.

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] Nicole Choi. 2024. Customizing and fine-tuning LLMs: What you need to know — github.blog. https://github.blog/2024-02-28-customizing-and-fine-tuning-llms-what-you-need-to-know/. [Accessed 28-03-2024].

[13] Google Cloud. 2023. Sec-PaLM2. https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/sec-palm. [Accessed 26-03-2024].

[14] Carlos Dantas, Adriano Rocha, and Marcelo Maia. 2023. Assessing the Readability of ChatGPT Code Snippet Recommendations: A Comparative Study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 283–292. https://doi.org/10.1145/3613372.3613413

[15] Alex de Vries. 2023. The growing energy footprint of artificial intelligence. *Joule* 7, 10 (2023), 2191–2194.

[16] Edoardo Debenedetti, Giorgio Severi, Nicholas Carlini, Christopher A Choquette-Choo, Matthew Jagielski, Milad Nasr, Eric Wallace, and Florian Tramèr. 2023. Privacy side channels in machine learning systems. *arXiv preprint arXiv:2309.05610* (2023).

[17] Shihan Dou, Junjie Shan, Haoxiang Jia, Wenhao Deng, Zhiheng Xi, Wei He, Yueming Wu, Tao Gui, Yang Liu, and Xuanjing Huang. 2023. Towards understanding the capability of large language models on code clone detection: a survey. *arXiv preprint arXiv:2308.01191* (2023).

[18] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. arXiv:2310.03533 [cs.SE]

[19] Sidong Feng and Chunyang Chen. 2024. Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[20] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324* (2019).

[21] GitHub. 2023. GitHub Copilot Community. https://github.com/orgs/community/discussions/categories/copilot?discussions_q=category%3ACopilot+network. [Accessed 03-10-2023].

[22] GitHub. 2023. GitHub Copilot · Your AI pair programmer. https://github.com/features/copilot/. [Accessed 22-03-2024].

[23] Green.earth. 2024. Google's $35 million venture into the carbon credit market. https://www.green.earth/news/googles-35-million-venture-into-the-carbon-credit-market. [Accessed 27-03-2024].

[24] Anastasiia Grishina, Max Hort, and Leon Moonen. 2023. The EarlyBIRD Catches the Bug: On Exploiting Early Layers of Encoder Models for More Efficient Code Classification. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 895–907. https://doi.org/10.1145/3611643.3616304

[25] J Gu, J Bradbury, C Xiong, VOK Li, and R Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*.

[26] Matt Hamblen. 2023. Update: ChatGPT runs 10K Nvidia training GPUs with potential for thousands more. https://www.fierceelectronics.com/sensors/chatgpt-runs-10k-nvidia-training-gpus-potential-thousands-more. [Accessed 02-04-2024].

[27] Dávid Hidvégi, Khashayar Etemadi, Sofia Bobadilla, and Martin Monperrus. 2024. CigaR: Cost-efficient Program Repair with LLMs.

[28] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems* 35 (2022), 30016–30030.

[29] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv preprint arXiv:2308.10620* (2023).

[30] Dong Huang, Jie M. Zhang, Yuhao Qing, and Heming Cui. 2024. EffiBench: Benchmarking the Efficiency of Automatically Generated Code.

[31] Jung-Chang Huang and Tau Leng. 1999. Generalized loop-unrolling: a method for program speedup. In *Proceedings 1999 IEEE International Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No. PR00122)*. IEEE, 244–248.

[32] Yizhan Huang, Yichen Li, Weibin Wu, Jianping Zhang, and Michael R Lyu. 2023. Do Not Give Away My Secrets: Uncovering the Privacy Issue of Neural Code Completion Tools. *arXiv preprint arXiv:2309.07639* (2023).

[33] Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of Code Language Models on Automated Program Repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 1430–1442.

[34] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. 2023. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1646–1656.

[35] Ayush Kaushal, Tejas Vaidhya, and Irina Rish. 2023. LORD: Low Rank Decomposition Of Monolingual Code LLMs For One-Shot Compression. arXiv:2309.14021 [cs.CL]

[36] Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. 2023. Bpipe: Memory-balanced pipeline parallelism for training large language models. In *International Conference on Machine Learning*. PMLR, 16639–16653.

[37] Jennifer L. 2023. Meta to Buy Almost 7 Million Carbon Credits from Aspiration — carboncredits.com. https://carboncredits.com/meta-to-buy-almost-7-million-carbon-credits-from-aspiration/. [Accessed 28-03-2024].

[38] Kif Leswing. 2023. ChatGPT and generative AI are booming, but the costs can be extraordinary. https://www.cnbc.com/2023/03/13/chatgpt-and-generative-ai-are-booming-but-at-a-very-expensive-price.html. [Accessed 02-04-2024].

[39] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.

[40] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024. Toward Sustainable GenAI using Generation Directives for Carbon-Friendly Large Language Model Inference. *arXiv preprint arXiv:2403.12900* (2024).

[41] Jiliang Li, Yifan Zhang, Zachary Karas, Collin McMillan, Kevin Leach, and Yu Huang. 2024. Do Machines and Humans Focus on Similar Code? Exploring Explainability of Large Language Models in Code Summarization. *arXiv preprint arXiv:2402.14182* (2024).

[42] Yijin Li, Jiacheng Zhao, Sun Qianqi, Haohui Mai, Lei Chen, Wanlu Cao, Yanfan Chen, YING LIU, Xinyuan Zhang, Xiyu Shi, et al. 2023. SIRIUS: Harvesting Whole-Program Optimization Opportunities for DNNs. *Proceedings of Machine Learning and Systems* 5 (2023).

[43] Jiaxing Liu, Chaofeng Sha, and Xin Peng. 2023. An Empirical Study of Parameter-Efficient Fine-Tuning Methods for Pre-Trained Code Models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, Los Alamitos, CA, USA, 397–408. https://doi.org/10.1109/ASE56229.2023.00125

[44] Shuo Liu, Jacky Keung, Zhen Yang, Fang Liu, Qilin Zhou, and Yihan Liao. 2024. Delving into Parameter-Efficient Fine-Tuning in Code Change Learning: An Empirical Study. arXiv:2402.06247 [cs.SE]

[45] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *ACM Transactions on Software Engineering and Methodology* (2023).

[46] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D. Le, and David Lo. 2024. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ACM Trans. Softw. Eng.*

Methodol. (jan 2024). https://doi.org/10.1145/3643674

[47] David Lo. 2023. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps. arXiv:2309.04142 [cs.SE]

[48] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, Los Alamitos, CA, USA, 647–658. https://doi.org/10.1109/ISSRE59848.2023.00026

[49] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *CoRR* (2021).

[50] Kasper Groes Albin Ludvigsen. 2023. Facebook disclose the carbon footprint of their new LLaMA models — kaspergroesludvigsen.medium.com. https://kaspergroesludvigsen.medium.com/facebook-disclose-the-carbon-footprint-of-their-new-llama-models-9629a3c5c28b. [Accessed 27-03-2024].

[51] Meta. 2023. Code Llama. https://ai.meta.com/blog/code-llama-large-language-model-coding/

[52] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Hongyi Jin, Tianqi Chen, and Zhihao Jia. 2023. Towards efficient generative large language model serving: A survey from algorithms to systems. *arXiv preprint arXiv:2312.15234* (2023).

[53] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *The Eleventh International Conference on Learning Representations*.

[54] Liang Niu, Shujaat Mirza, Zayd Maradni, and Christina Pöpper. 2023. {CodexLeaks}: Privacy Leaks from Code Generation Language Models in {GitHub} Copilot. In *32nd USENIX Security Symposium (USENIX Security 23)*. 2133–2150.

[55] Moumita Das Purba, Arpita Ghosh, Benjamin J Radford, and Bill Chu. 2023. Software vulnerability detection using large language models. In *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 112–119.

[56] Marcelino Rodriguez-Cancio, Benoit Combemale, and Benoit Baudry. 2016. Automatic microbenchmark generation to prevent dead code elimination and constant folding. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 132–143.

[57] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[58] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. 2023. From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–9.

[59] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2023. An empirical evaluation of using large language models for automated unit test generation. *IEEE Transactions on Software Engineering* (2023).

[60] Google Cloud Security. 2023. Introducing VirusTotal Code Insight: Empowering threat analysis with generative AI. https://blog.virustotal.com/2023/04/introducing-virustotal-code-insight.html. [Accessed 26-03-2024].

[61] Chaoxuan Shi, Tingwei Zhu, Tian Zhang, Jun Pang, and Minxue Pan. 2023. Structural-semantics Guided Program Simplification for Understanding Neural Code Intelligence Models. In *Proceedings of the 14th Asia-Pacific Symposium on Internetware (Internetware '23)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3609437.3609438

[62] Ensheng Shi, Yanlin Wang, Hongyu Zhang, Lun Du, Shi Han, Dongmei Zhang, and Hongbin Sun. 2023. Towards Efficient Fine-Tuning of Pre-trained Code Models: An Experimental Study and Beyond. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2023)*. Association for Computing Machinery, New York, NY, USA, 39–51. https://doi.org/10.1145/3597926.3598036

[63] Jieke Shi, Zhou Yang, Hong Jin Kang, Bowen Xu, Junda He, and David Lo. 2023. Greening Large Language Models of Code. *arXiv preprint arXiv:2309.04076* (2023).

[64] Jieke Shi, Zhou Yang, Bowen Xu, Hong Jin Kang, and David Lo. 2023. Compressing Pre-Trained Models of Code into 3 MB *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 24, 12 pages. https://doi.org/10.1145/3551349.3556964

[65] Jasvinder Singh. 2023. What is the Energy Consumption of AI enabled Large Language Modelling (LLM's) types of searches? — linkedin.com. https://www.linkedin.com/pulse/what-energy-consumption-ai-enabled-large-language-modelling-singh. [Accessed 27-03-2024].

[66] Chia-Yi Su and Collin McMillan. 2024. Distilled GPT for source code summarization. *Automated Software Engineering* 31, 1 (2024), 22. https://doi.org/10.1007/s10515-024-00421-4

[67] Zhensu Sun, Xiaoning Du, Fu Song, Shangwen Wang, and Li Li. 2024. When Neural Code Completion Models Size up the Situation: Attaining Cheaper and Faster Completion through Dynamic Model Inference. arXiv:2401.09964 [cs.SE]

[68] Zhensu Sun, Xiaoning Du, Fu Song, Shangwen Wang, Mingze Ni, and Li Li. 2023. Don't Complete It! Preventing Unhelpful Code Completion for Productive and Sustainable Neural Code Completion Systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 324–325. https://doi.org/10.1109/ICSE-Companion58688.2023.00089

[69] Google Sustainability. 2023. Aiming to Achieve Net-Zero Emissions - Google Sustainability — sustainability.google. https://sustainability.google/operating-sustainably/net-zero-carbon/. [Accessed 28-03-2024].

[70] Alexey Svyatkovskiy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, Juliana Vicente Franco, and Miltiadis Allamanis. 2021. Fast and memory-efficient neural code completion. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 329–340.

[71] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. 2022. Transformer-based language models for software vulnerability detection. In *Proceedings of the 38th Annual Computer Security Applications Conference*. 481–496.

[72] Theodoros Theodoridis, Manuel Rigger, and Zhendong Su. 2022. Finding missed optimizations through the lens of dead code elimination. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 697–709.

[73] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[74] Tina Vartziotis, Ippolyti Dellatolas, George Dasoulas, Maximilian Schmidt, Florian Schneider, Tim Hoffmann, Sotirios Kotsopoulos, and Michael Keckeisen. 2024. Learn to Code Sustainably: An Empirical Study on LLM-based Green Code Generation. *arXiv preprint arXiv:2403.03344* (2024).

[75] Fang Wang, Jean Damascene Harindintwali, Zhizhang Yuan, Min Wang, Faming Wang, Sheng Li, Zhigang Yin, Lei Huang, Yuhao Fu, Lei Li, et al. 2021. Technologies and perspectives for achieving carbon neutrality. *The Innovation* 2, 4 (2021).

[76] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* (2024).

[77] Frank Watson. 2021. Microsoft buys 1.3 million carbon offsets in 2021 portfolio. https://www.spglobal.com/commodityinsights/en/market-insights/latest-news/coal/012921-microsoft-buys-13-million-carbon-offsets-in-2021-portfolio. [Accessed 27-03-2024].

[78] Xiaokai Wei, Sujan Kumar Gonugondla, Shiqi Wang, Wasi Ahmad, Baishakhi Ray, Haifeng Qian, Xiaopeng Li, Varun Kumar, Zijian Wang, Yuchen Tian, Qing Sun, Ben Athiwaratkun, Mingyue Shang, Murali Krishna Ramanathan, Parminder Bhatia, and Bing Xiang. 2023. Towards Greener Yet Powerful Code Generation via Quantization: An Empirical Study. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (San Francisco, CA, USA) (ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 224–236. https://doi.org/10.1145/3611643.3616302

[79] Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the copilots: Fusing large language models with completion engines for automated program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 172–184.

[80] Sarah Wells. 2023. Generative AI's Energy Problem Today is Foundational. https://spectrum.ieee.org/ai-energy-consumption. [Accessed 28-03-2024].

[81] Martin Weyssow, Xin Zhou, Kisub Kim, David Lo, and Houari Sahraoui. 2024. Exploring Parameter-Efficient Fine-Tuning Techniques for Code Generation with Large Language Models. arXiv:2308.10462 [cs.SE]

[82] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1482–1494.

[83] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[84] Zhou Yang, Zhensu Sun, Terry Zhuo Yue, Premkumar Devanbu, and David Lo. 2024. Robustness, security, privacy, explainability, efficiency, and usability of large language models for code. *arXiv preprint arXiv:2403.07506* (2024).

[85] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, DongGyun Han, and David Lo. 2023. Unveiling Memorization in Code Models.

[86] Deborah Yao. 2023. Microsoft's GitHub Copilot Loses $20 a Month Per User — aibusiness.com. https://aibusiness.com/nlp/github-copilot-loses-20-a-month-per-user. [Accessed 28-03-2024].

[87] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv*

preprint arXiv:1904.00962 (2019).

[88] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.

[89] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2023. A Survey on Large Language Models for Software Engineering. *arXiv preprint arXiv:2312.15223* (2023).

[90] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2023. Cupid: Leveraging chatgpt for more accurate duplicate bug report detection. *arXiv preprint arXiv:2308.10022* (2023).

[91] Zhaowei Zhang, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Diet Code is Healthy: Simplifying Programs for Pre-Trained Models of Code. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1073–1084. https://doi.org/10.1145/3540250.3549094

[92] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. *arXiv preprint arXiv:2403.03507* (2024).

[93] Zibin Zheng, Kaiwen Ning, Yanlin Wang, Jingwen Zhang, Dewu Zheng, Mingxi Ye, and Jiachi Chen. 2023. A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends. arXiv:2311.10372 [cs.SE]

[94] Xin Zhou, Ting Zhang, and David Lo. 2024. Large Language Model for Vulnerability Detection: Emerging Results and Future Directions. *arXiv preprint arXiv:2401.15468* (2024).

[95] Andrei Zlotchevski, Dawn Drain, Alexey Svyatkovskiy, Colin B. Clement, Neel Sundaresan, and Michele Tufano. 2022. Exploring and Evaluating Personalized Models for Code Generation *(ESEC/FSE 2022)*. 1500–1508. https://doi.org/10.1145/3540250.3558959