OPTIMIZING COMPUTATIONAL EFFICIENCY AND SECURITY-BY-DESIGN CODE SECURITY IN LARGE LANGUAGE MODEL PIPELINES: COMPARATIVE ANALYSIS OF INFERENCE COSTS

OTIMIZAÇÃO DA EFICIÊNCIA COMPUTACIONAL E SEGURANÇA DE CÓDIGO POR PROJETO EM PIPELINES DE MODELOS DE LINGUAGEM DE GRANDE PORTE: ANÁLISE COMPARATIVA DE CUSTOS DE INFERÊNCIA

OPTIMIZACIÓN DE LA EFICIENCIA COMPUTACIONAL Y SEGURIDAD DEL CÓDIGO POR DISEÑO EN GRANDES CANALIZACIONES DE MODELOS DE LENGUAJE: ANÁLISIS COMPARATIVO DE LOS COSTOS DE INFERENCIA

https://doi.org/10.56238/sevened2025.036-117

Tagleorge Silveira[1], Pedro Pinheiro[2], Hélder Rodrigo Pinto[3], Salviano Pinto Soares[4], José Baptista[5]

ABSTRACT

Large Language Models (LLMs) have revolutionized software engineering by generating code rapidly and accurately across a wide range of programming tasks. However, the growing reliance on these models raises concerns regarding their energy consumption, runtime overhead, and the efficiency of achieving successful, security-by-design, and maintainable code. This article presents an analytical comparison of several leading LLMs—such as OpenAI's GPT series, Anthropic's Claude, Meta's Llama, and Google's Gemini—by evaluating their success rates in producing secure and optimized code, the number of prompts required for successful output, and the corresponding computational and energy costs. The findings emphasize strategies to balance accuracy, performance, and sustainability in LLM-assisted programming.

Keywords: LLMs. Energy Efficiency. Cybersecurity. Security-by-design. Optimization.

RESUMO

Os Grandes Modelos de Linguagem (LLMs) revolucionaram a engenharia de software ao gerar código de forma rápida e precisa em uma ampla gama de tarefas de programação. No

[1] Master and Doctoral Student Electrical and Computer Engineering (ECE). Universidade de Trás-os-Montes e Alto Douro (UTAD). Professor at Instituto Superior de Tecnologias Avançadas. ISTEC Porto. Researcher at CITECA . ISTEC Porto, and IEEE (S'17–M'21). E-mail: tagleorge@ieee.org,
[2] Multimedia Engineering Student. Instituto Superior de Tecnologias Avançadas (ISTEC Porto). CITECA - ISTEC Porto.E-mail: pedro.pinheiro8912@gmail.com Orcid: https://orcid.org/0009-0009-5866-0582
[3] Dr. in Information Sciences. MSc in Computer Engineering. PG in Artificial Intelligence & Machine Learning. MBA in Business Management. Professor at Instituto Superior de Tecnologias Avançadas (ISTEC Porto).Researcher at CITECA - ISTEC Porto, Professor at ISEP and Researcher at GECAD - ISEP. E-mail: helder.pinto@my.istec.pt, Orcid: https://orcid.org/0009-0003-5810-9383.
[4] Dr. in Electrical Engineering. Professor School of Sciences and Technology-Engineering Department (UTAD), 5000-801 Vila Real. Portugal; Researcher at Institute of Electronics and Informatics Engineering of Aveiro (IEETA). University of Aveiro, 3810-193 Aveiro, Portugal, and Intelligent Systems Associate Laboratory (LASI). E-mail: salblues@utad.pt, Orcid: https://orcid.org/0000-0001-5862-5706.
[5] Dr. in Electrical and Computer Engineering. Professor School of Sciences and Technology-Engineering Department (UTAD), 5000-801 Vila Real, Portugal, Researcher in Institute of Systems and Computing Engineering (INESC-TEC) Porto.E-mail: baptista@utad.pt, Orcid: https://orcid.org/0000-0003-0297-4709.

entanto, a crescente dependência desses modelos levanta preocupações quanto ao seu consumo de energia, sobrecarga de tempo de execução e à eficiência na obtenção de um código bem-sucedido, seguro desde a concepção e de fácil manutenção. Este artigo apresenta uma comparação analítica de vários LLMs líderes — como a série GPT da OpenAI, Claude da Anthropic, Llama da Meta e Gemini do Google — avaliando as suas taxas de sucesso na produção de código seguro e otimizado, o número de prompts necessários para uma saída bem-sucedida e os custos computacionais e energéticos correspondentes. As conclusões enfatizam estratégias para equilibrar precisão, desempenho e sustentabilidade na programação assistida por LLM.

Palavras-chave: LLMs. Eficiência Energética. Cibersegurança. Segurança por Design. Otimização.

RESUMEN

Los Grandes Modelos de Lenguaje (LLM) han revolucionado la ingeniería de software al generar código de forma rápida y precisa en una amplia gama de tareas de programación. Sin embargo, la creciente dependencia de estos modelos genera inquietudes respecto de su consumo de energía, la sobrecarga en tiempo de ejecución y la eficiencia para lograr un código exitoso, seguro por diseño y mantenible. Este artículo presenta una comparación analítica de varios LLM líderes, como la serie GPT de OpenAI, Claude de Anthropic, Llama de Meta y Gemini de Google, evaluando sus tasas de éxito en la producción de código seguro y optimizado, el número de indicaciones necesarias para obtener un resultado satisfactorio y los correspondientes costes computacionales y energéticos. Los resultados ponen de relieve las estrategias para equilibrar la precisión, el rendimiento y la sostenibilidad en la programación asistida por LLM.

Palabras clave: Llms. Eficiencia Energética. Ciberseguridad. Seguridad por Diseño. Optimización.

1 INTRODUCTION

The need to reduce costs and improve efficiency is a necessity of modern times. Code security and efficiency is critical for the best usage of required systems and functionalities that require less runtime and less energy expenditure. The focus of this analysis is conducted mainly on the parameters mentioned above, in order to determine if both of them can be lowered without sacrificing models resourcefulness and quality, therefore making sure that the desired output of a correct, error free code and the development of secure-by-design systems, can be achieved without having many losses in time, energy, code efficiency, and security. The effective need of having the safety and efficiency of the code generated by an LLM are the priority in this matter, however, the analysis of the cost-benefit ratio is also important for large scale programs that work with large data sets, making sure that no error occurs, and if it does, that the code refactoring is done appropriately and quickly, saving energy and time.

The motivation for the realization of this work is centered around the need for better management of good coding practices, as well as the need to analyze the resulting costs of the used energy and time spent working on providing possible code-refactoring solutions. This theme is relevant today as the demand for reliable, emission-free energy is increasingly urgent. Without one, the need to reduce costs and augment efficiency is critical for the well management of society nowadays. Therefore, the analysis of the objectives mentioned above, as well as the possible comparison between some LLMs, is highly needed to understand and make a full comprehensive use of the capabilities of the tools at hand in order to achieve the best possible result with low-cost performance (Bhatt, 2025).

2 METHOD

In this investigation, an initial analysis of recent and relevant literature was conducted, which sought both to identify the articles to be studied in technology and the usage of AI for the best possible code output with as few errors as possible, whilst maintaining energy efficiency and low runtime overhead. The process of evaluation and interpretation of the reflections placed on the selected biblioFigureic material was made over selected articles that made comparisons between different LLM models, as well as the study of spent energy resources when refactoring or generating specific code.

(I) Efficient, faster coding to increase success rates

The need to enhance and improve the coding methods and speed has always been a much regarded need in tech. Nowadays, LLMs combined with agentic AIs can reduce time strains and make apps, programs and websites in a matter of minutes, in comparison to a human developer. However, since the models are not perfect, the code can contain some noticeable errors that a good developer would not make, even if it takes them longer to develop the software in question. Another impactful parameter when generating good and efficient code is a well written prompt that the model will be able to process with a more degree of precision, without the need to deviate from the objective at hand. Different prompt strategies may be implemented when using LLMs to determine the success rate that one can output, as an example, the comparison of the success rates of six different prompt strategies was made on GPT-4, as well as other LLMs, showing the difference in the number of attempts, as well as the change in the degree of difficulty of the tasks themselves (Hou, 2024).

Another control parameter that should be considered is the self-planning code generation that we can implement using LLMs. The self-planning phase can be done by matching the intent with the plan therefore not only giving the initial prompt of what the user wants the LLM to do but also indicating step by step specifically what the LLM needs to do therefore the output should be a lot cleaner and a lot more accurate than if it was only with the initial prompt without any guidance (Jiang, 2023; Ye, 2025). Another technique involving a Large language model Aware selection approach for In-context-Learning based code generation named LAIL, is also to be considered for the remarkable improvements in generating desired programs, outperforming state-of-the-art baselines by considerable amount (Li, 2023). This new approach uses other LLMs To select a positive or negative example for requirement, this means that other LLMs are responsible for the ultimate outcome of LAIL.

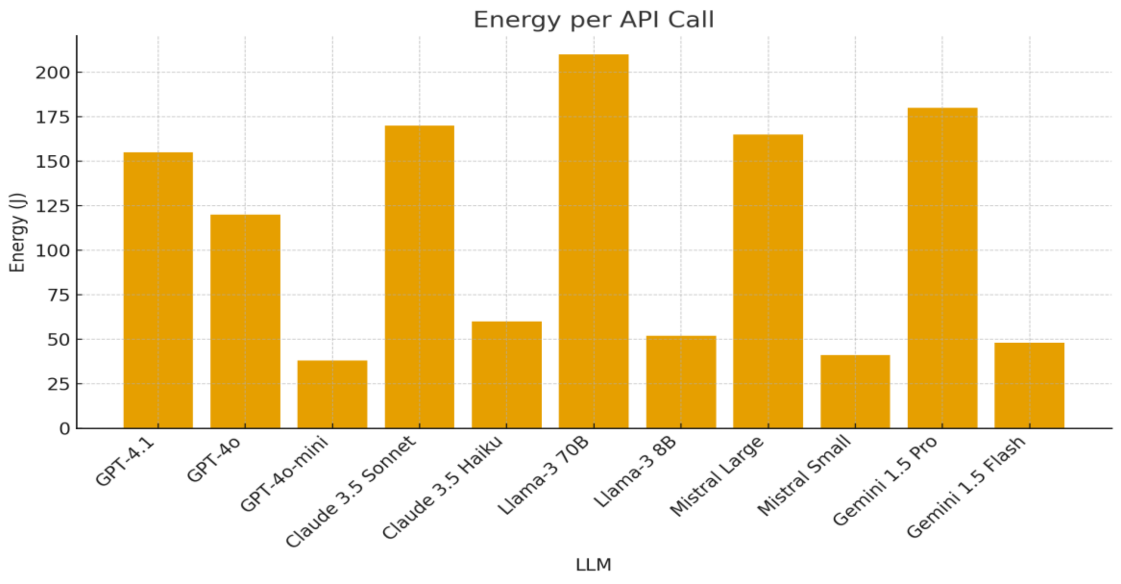
(II) Measuring energy and emissions in ML

The environmental costs that training an LLM have are significant. Dozens of hundreds of tons of CO2 are just a simple fact of the cost of training and LLM, not to mention the expenditure and energy. An example of that is the LLM Bloom which generated 24,7 and 50,5 tons of CO2 (Luccioni, 2022; Jiang, 2024; Singh, 2024). The training costs of LLMs are indeed significant, however inference, the process of responding to user queries is where costs ultimately increase exponentially and the environment is more affected. One query on ChatGPT is estimated to cause an emission of 4 grams of CO2 more than 20 times more than one web search. The reduction of carbon footprint when training/using LLMs has been largely overlooked, therefore the need to make responsible advancements to reduce carbon emissions prioritizing the environment without decreasing efficiency when using LLMs is a needed directive for the future (Wu, 2025; Ding, 2024; Ozcan, 2025).

To show an initial example of a small sample of models, in terms of energy per API call that the models represented in Figure 1. This shows us that there is a certain amount of energy that is used whenever we prompt the model to obtain a relative output. We can also infer from this that the largest models use the most amount of energy, whereas the smaller models use the least amount.

Figure 1

Energy per API Call Model Comparison

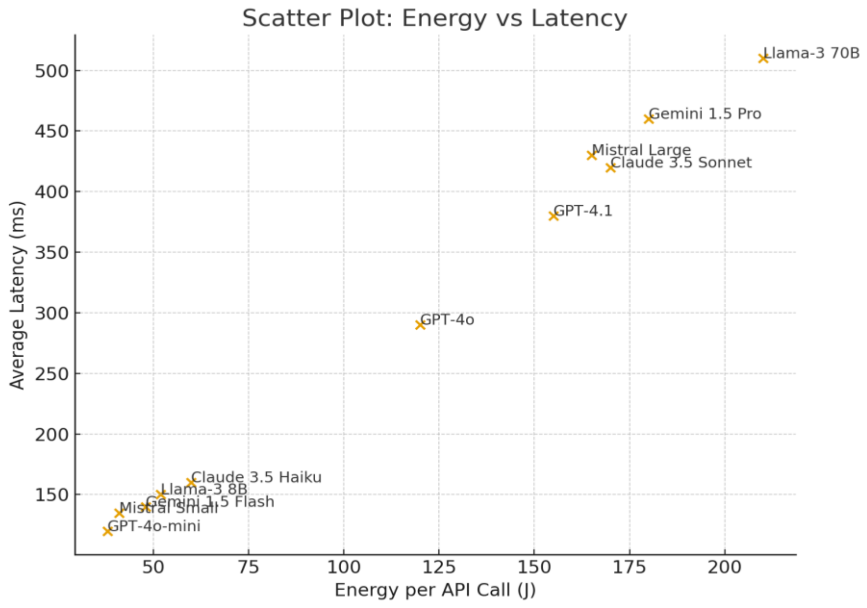


Source: Produced by the author, 2025.

As can be observed on Figure 2 again, the largest models consume the highest amount of energy per API call and have the largest response time, whereas the low parameter models are the opposite. If the objective is to be truthful to energy efficient models, then we cannot use LLMs. However compressed LLMs can provide results with near equal success rate in comparison to uncompressed LLMs while spending less energy per API call and taking less time.

Figure 2

Energy vs Latency Model Comparison

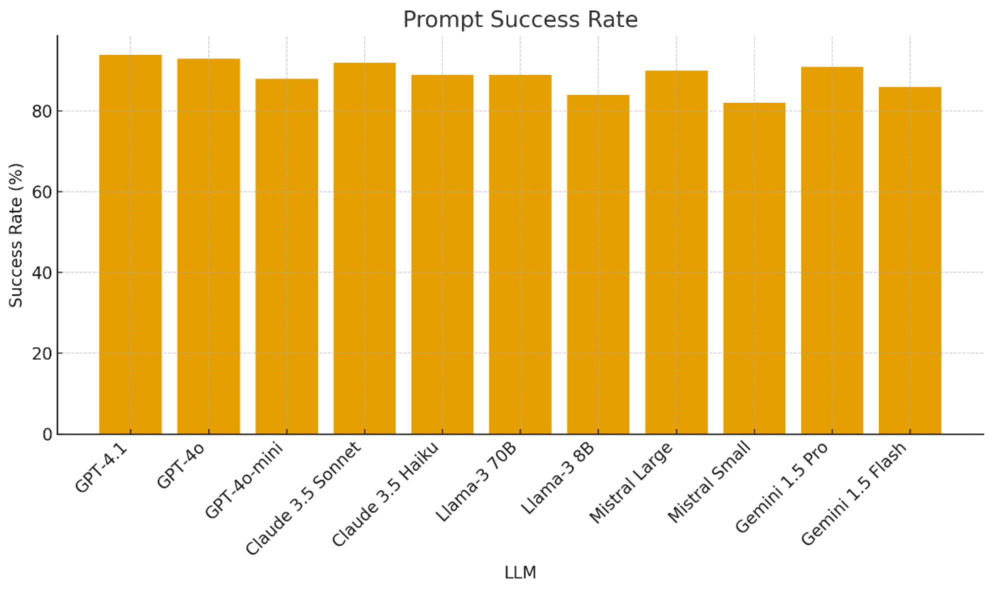


Source: Produced by the author, 2025.

As can be seen, Figure 3, the prompt success rate doesn't differ much from model to model, only on more complex prompts can relevantly differences from model to model.

Figure 3

Prompt Success Rate Model Comparison

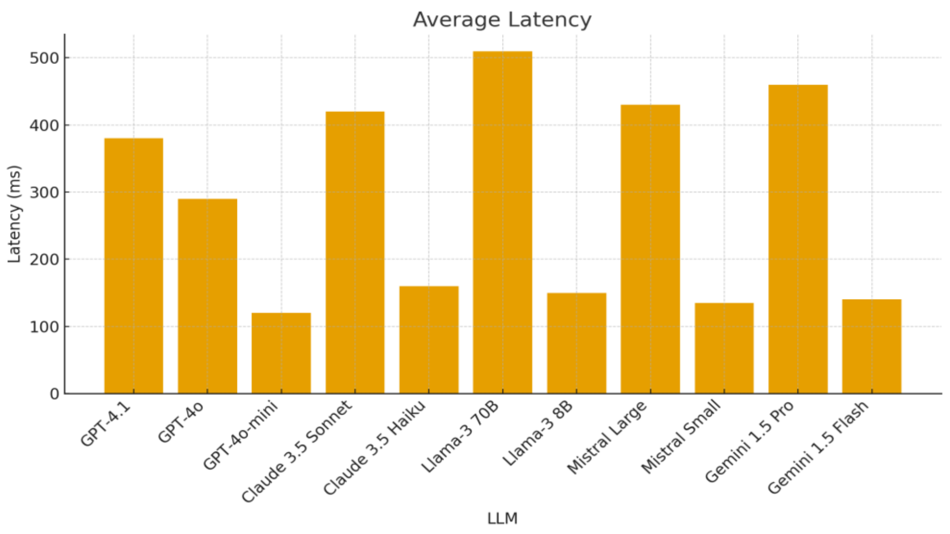


Source: Produced by the author, 2025.

Prompt success rate is considered the capability of an LLM to complete the task given by the used in a single prompt with no additional guidelines/instructions on any other prompts. However, in Figure 4 it can be observed, in a much more comprehensive manner, that latency differs from models with more parameters in comparison to models with less parameters.

Figure 4

Average Latency Model Comparison



Source: Produced by the author, 2025.

(III) Model selection & architecture choices

Model selection and Architectural choices are important for achieving a more accurate LLM. Studies indicate that 7B and 13B models can achieve high accuracy rates despite having lower parameters and in token usage. The more tokens are used in inference processes the more energy will be consumed for the expected output. Not only this but there's also a need to reduce the latency of said inference. The idea is that the usage of less tokens per second over a large share batch size as well as over time becomes less and less, therefore reducing the associated energy costs and carbon footprint (Bian, 2025; Yuan, 2025).

There's also the possibility to use a mixture of experts' architecture for better and faster reasoning from LLMs. This consists of a router that redirects tokens to the experts, specialized sub-networks, that handle different parts of the problem, weighing them to give a more accurate and precise output (Yuan, 2025). This approach is one of the most energy saving options that we must consider, since it disperses energy for more selected and unique parts of processing and comprehension whereas beforehand one would need to access an entire LLM for just a simple query. With this method only parts of an LLM need to be accessed to generate a viable and accurate response. The more complex the mending process, more parts will need to be accessed in order to be accurate and therefore achieve a proper output.

The consideration of smaller language models to achieve low latency while maintaining the same level of accuracy as larger models is noticeable, inference and token utilization are also kept low therefore the need to use less memory is also noticeable. For problem specific questions/objectives the 7B parameters LLMs are an option to be considered, they can even be used to train larger LLMs with 70B or 150B parameters. This takes into consideration the rate of accuracy and success when responding to user inputs. However, the degree of accuracy and detail on a response needs to be followed with reduction and energy costs.

(IV) Runtime Optimizations

Runtime optimizations are important to consider reducing time spent and energy spent with reasoning. Quantization is an important technique to consider for improving model memory footprint and keeping model accuracy. Post training quantization reduces inference compute by up to 3x-4x, with no noticeable performance loss. The knowledge distillation technique relies on smaller models teaching larger ones, therefore reducing costs associated with processing information and reducing runtime overhead, as well as cutting costs by 70% (Xiao, 2023; Ganesh, 2024).

The early exit technique allows the reduction of execution time, therefore being an interesting aspect to consider when wanting to lower energy costs. The technique consists in reducing the time of reasoning by verifying the threshold of said reasoning and when the threshold achieves a desired percentage (40-70%), the reasoning will stop, and an answer will be given. This method benefits simpler, less complex input prompts from the user, reducing overall run time overhead and lowering energy costs by 23-50% but also lowering response time (Ilager, 2025).

(V) Hardware & Infrastructure Choices

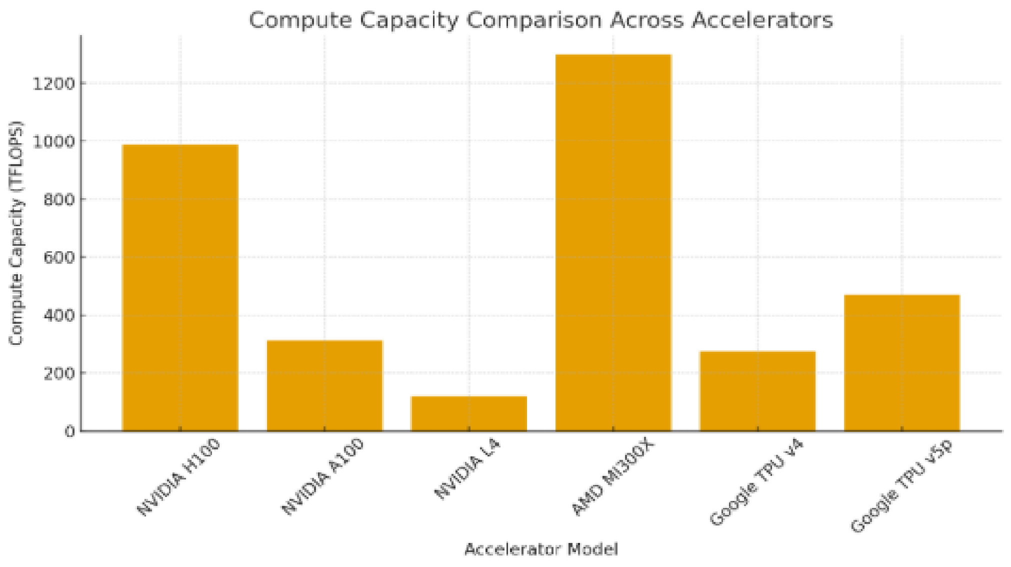
Hardware selection is a fundamental prospect that needs to be considered when establishing new LLM training routines. GPUs are the standard for LLM inference, however special accelerators such as TPUs, AWS Inferentia, and other emerging AI specific Asics, can also achieve a high energy efficiency per token processed. The need to develop new customized accelerators has been a modern advancement made by companies like Google, to improve speed and energetic efficiency on LLMs (Li, 2024; Chowdhery, 2022). Tensor model parallelism is another method to improve overall processing on the LLM by creating different partitions of the weights and then concatenating them to achieve the desired output. This is done so that there is less strain on memory. The most commonly used format is the Megatron LM, which is used on LLM's with 50-70b parameters, however, the latency increases with this technique, however the precision and scalability increase as well. (Narayanan, 2021).

An important note to take into account is that energy efficiency per token is referring to long-term usage since we are discussing LLMs that is the main crucial point that we need to take into account when we speak about energy efficiency we need to consider a long-term options instead of short-term ones, reliability needs to be achieved**,** for that to consideration of accuracy and precision as the core and primary motivators for such research needs to be invoked. After that, the need for energy efficient solutions must be pursued. the next Figures will show the difference in energy as well as cost efficiency between certain GPUs and TPUs, as well as inference latency comparison, so that inadequate study of such values can be done.

In Figure 5 shows the difference from several GPUs and TPUs in compute capacity. This is meant to illuminate regarding the Tensor Processing Units which ultimately cost more energy to run, as well as working with a different architecture than a GPU. The biggest bars in this Figure are of the GPUs.

Figure 5

Compute Capacity Comparison Across Accelerators

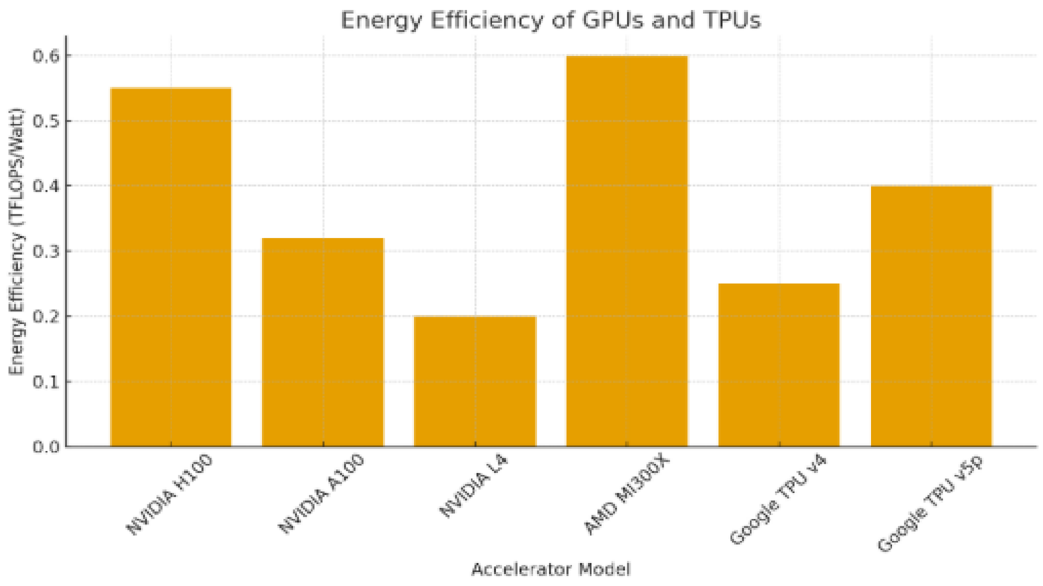


Source: Produced by the author, 2025.

The Figure 6 shows the difference between GPUs and TPUs regarding Energy Efficiency. Due to the specific design of a TPU, the usage of energy is more controlled since only some parts of the architecture are chosen to run the models themselves, this makes it a much more energetically efficient strategy to reduce the carbon footprint associated with LLMs

Figure 6

Energy Efficiency of GPUs and TPUs

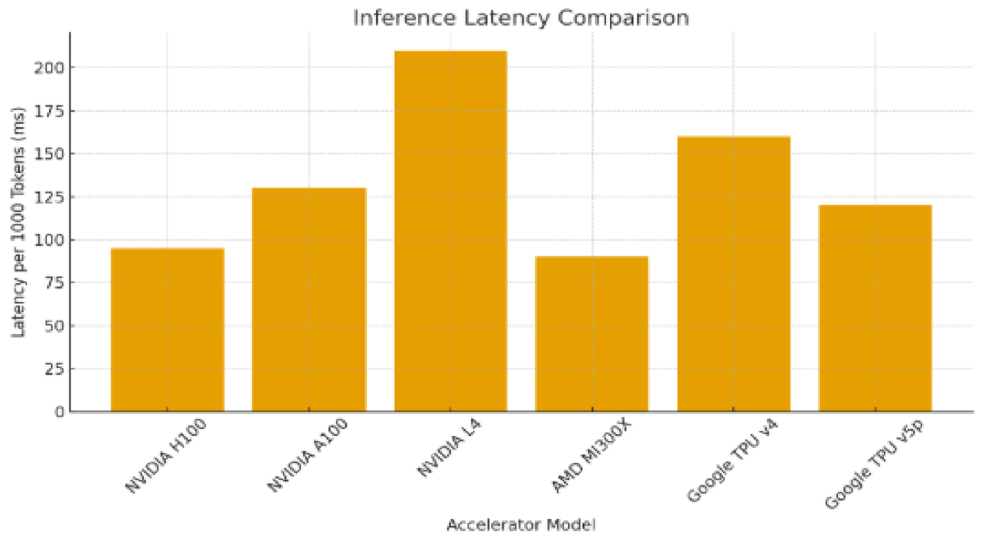


Source: Produced by the author, 2025.

In Figure 7 is meant to show the inference to latency comparison between GPUs and TPUs. It can be observed that since the customized architecture of a TPU is more energy friendly the latency will therefore increase. The division of weights for this architecture is the main concern when it comes to latency issues, for the concatenation of the weights needs to be done so that the overall output can be accurate and precise. On the other hand the GPUs have less latency and more power expenditure.

Figure 7

Inference Latency Comparison



Source: Produced by the author, 2025.

(VI) Secure and Clean Code Practices to Reduce Overhead

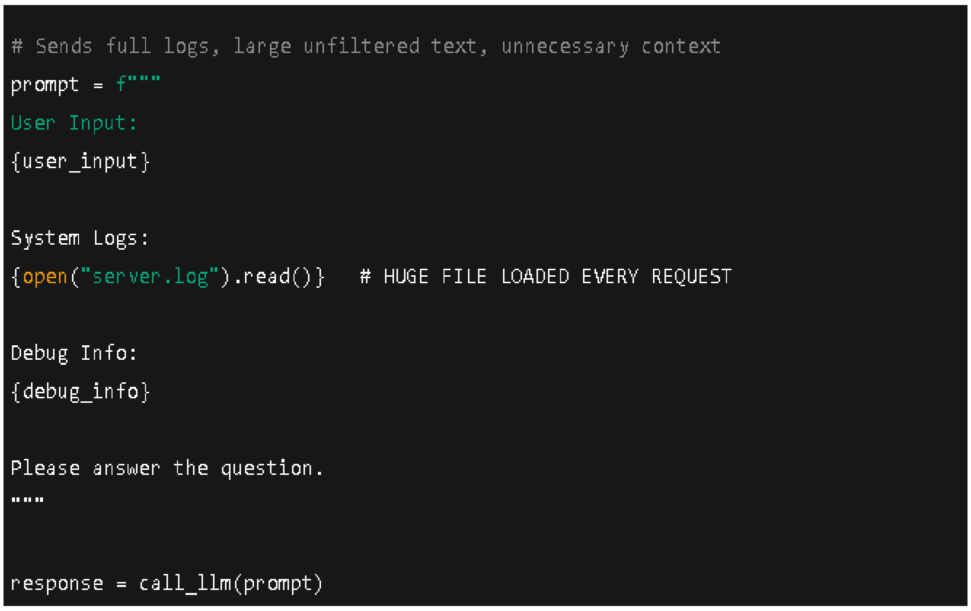
There's now more than ever an urgent need to write good, polished, clean and secure code, not only to avoid critical errors but also to maintain good security measures on websites, but also for the development of security-by-design systems, as well as to avoid possible bugs/errors in the future of a critical piece of software. Code reusability as well as revision and maintenance, are all important and for that, code needs to be written in a manner that generally can be well perceived by anyone that works with writing or reading code (Hunter-Zinck, 2021; Digkas, 2020).

Azure guidelines provide a set of best practices to build secure, scalable, clean and maintainable as well as cost-efficient applications. These guidelines are important because they are also applied in AI pipelines as well as LLMs. The figures below are meant to illustrate the difference between good and bad coding.

The Figure 8 shows code that loads and sends entire logs with a massive token usage, allowing for the user to input information that can be out of context or just not need it therefore making latency time higher as well as energy costs. Figure 9 shows a clean code that determines the length of the input therefore giving it a smaller token window lowering energy cost and allowing for a faster runtime.

Figure 8

Bad coding example #1

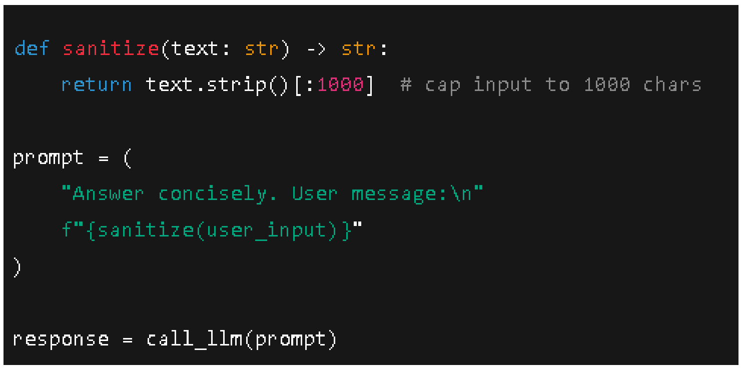


```
# Sends full logs, large unfiltered text, unnecessary context
prompt = f"""
User Input:
{user_input}

System Logs:
{open("server.log").read()}   # HUGE FILE LOADED EVERY REQUEST

Debug Info:
{debug_info}

Please answer the question.
"""

response = call_llm(prompt)
```

Source: Produced by the author, 2025.

Figure 9

Good Coding Example #1



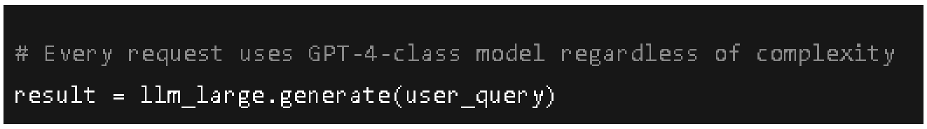
```
def sanitize(text: str) -> str:
    return text.strip()[:1000]  # cap input to 1000 chars

prompt = (
    "Answer concisely. User message:\n"
    f"{sanitize(user_input)}"
)

response = call_llm(prompt)
```

Source: Produced by the author, 2025.

In Figure 10 shows code that just attaches the results meaning every single request made from a user into an LLM, whereas Figure 11 routes input query from the user adding a length to the input and then rerouting it to a small LLM if it meets that criteria or to a large LLM if it meets other criteria this allows for faster queries so lower latency reducing overhead runtime from 50 to 80%.

Figure 10

Bad Coding Example #2

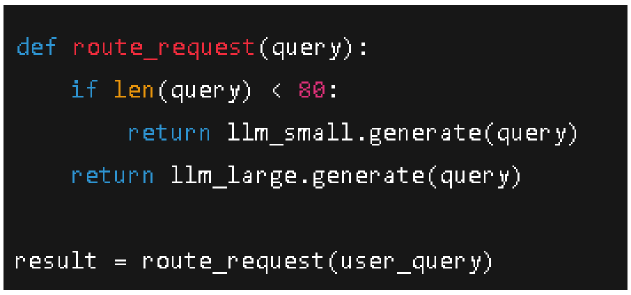


```
# Every request uses GPT-4-class model regardless of complexity
result = llm_large.generate(user_query)
```

Source: Produced by the author, 2025.

Figure 11

Good Coding Example #4



```
def route_request(query):
    if len(query) < 80:
        return llm_small.generate(query)
    return llm_large.generate(query)

result = route_request(user_query)
```

Source: Produced by the author, 2025.

(VII) Case Studies & Worked Examples

Real world deployments of LLMs, like Llama-3 series models, provide estimates of inference energy per 1k tokens. Meta's Llama-3 Shows that a 70B-parameter model consumes several times more energy than a fine-tuned 8B model, while the smaller one still performs adequately for many domain-specific applications. Another example, is a noticeable energy reduction of almost 100 Joules per API call is OpenAI's GPT-4o Mini, compared with GPT-4o (OpenAI, 2024; Meta AI, 2024).

3 CONCLUSION

The initial analysis presented throughout this chapter demonstrates that reducing energy consumption and runtime overhead in LLM-assisted software development is both technically feasible and essential for sustainable computational practices. Modern Large Language Models, such as GPT-4o, Claude 3, Llama-3, and Gemini, offer extraordinary capabilities for generating secure, clean, and efficient code, yet their performance must be weighed against the significant environmental and computational costs associated with large-scale inference. As shown through recent scientific studies, smaller or optimized models (7B–13B parameters), quantized architectures, mixture-of-experts routing, early-exit reasoning, and hardware-aware deployment strategies can collectively reduce energy consumption by 50–80% without compromising accuracy or security.

The review also reinforces that prompt engineering, self-planning strategies, and LLM-aware selection frameworks such as LAIL significantly increase code-generation success rates while minimizing the number of model calls required—directly reducing energy expenditure. Likewise, secure and clean code practices, backed by Azure's engineering guidelines, show that well-structured, modular, and token-efficient code pipelines contribute meaningfully to reducing computational overhead. In parallel, empirical case studies confirm that the shift from monolithic 70B-parameter systems to distilled or specialized smaller models results in dramatic gains in efficiency, lower emissions, and improved maintainability, especially when caching, batching, and routing mechanisms are included.

Ultimately, this chapter highlights that the path toward sustainable AI-assisted programming relies on a holistic approach: combining model-level optimizations, careful choice of architecture, infrastructure-aware deployment, low-overhead prompt strategies, and strict adherence to secure coding principles, such as security-by-design software development. By integrating these practices, developers can aim to achieve high-quality, error-free code while simultaneously lowering carbon footprints, operational costs, development of security-by-design systems and avoiding runtime inefficiencies. As the technology continues to evolve, responsible model usage—focused on accuracy, energy efficiency, security and long-term sustainability—must become a central pillar of modern software engineering.

REFERENCES

1. Bian, S., Yan, M., & Venkataraman, S. (2025). *Scaling Inference-Efficient Language Models*. ArXiv, abs/2501.18107. https://doi.org/10.48550/arxiv.2501.18107.

2. Bhatt, P., & Kashiyani, P. (2025). *Generative AI for Code Synthesis: A Comparative Study of Large Language Models in Software Engineering*. International Journal of Scientific Research in Engineering and Management. https://doi.org/10.55041/ijsrem44215.

3. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., García, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A., Pillai, T., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Díaz, M., Firat, O., Catasta,

M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., & Fiedel, N. (2022). *PaLM: Scaling Language Modeling with Pathways*. ArXiv, abs/2204.02311.

4. Ding, Y., & Shi, T. (2024). *Sustainable LLM Serving: Environmental Implications, Challenges, and Opportunities: Invited Paper*. 2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC), 37–38. https://doi.org/10.1109/igsc64514.2024.00016

5. Digkas, G., Chatzigeorgiou, A., Ampatzoglou, A., & Avgeriou, P. (2020). *Can Clean New Code Reduce Technical Debt Density?* IEEE Transactions on Software Engineering, 48, 1705–1721. https://doi.org/10.1109/tse.2020.3032557

6. Ganesh, P., et al. (2024). *DeciLM: Distilled Models for Efficient LLM Serving*. ACL 2024. https://aclanthology.org/2024.acl-long.122/

7. Hou, W., & Ji, Z. (2024). *Comparing Large Language Models and Human Programmers for Generating Programming Code*. Advanced Science, 12. https://doi.org/10.1002/advs.202412279.

8. Hunter-Zinck, H., De Siqueira, A., Vasquez, V., Barnes, R., & Martinez, C. (2021). *Ten simple rules on writing clean and reliable open-source scientific software*. PLoS Computational Biology, 17. https://doi.org/10.1371/journal.pcbi.1009481.

9. Ilager, S., Briem, L., & Brandić, I. (2025). *Green-Code: Learning to Optimize Energy Efficiency in LLM-Based Code Generation*. 2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 559–569. https://doi.org/10.1109/ccgrid64434.2025.00068.

10. Jiang, P., Sonne, C., Li, W., You, F., & You, S. (2024). *Preventing the Immense Increase in the Life-Cycle Energy and Carbon Footprints of LLM-Powered Intelligent Chatbots*. Engineering. https://doi.org/10.1016/j.eng.2024.04.002.

11. Jiang, X., Dong, Y., Wang, L., Shang, Q., & Li, G. (2023). *Self-Planning Code Generation with Large Language Models*. ACM Transactions on Software Engineering and Methodology, 33, 1–30. https://doi.org/10.1145/3672456.

12. Li, J., Tao, C., Li, J., Li, G., Jin, Z., Zhang, H., Fang, Z., & Liu, F. (2023). *Large Language Model-Aware In-Context Learning for Code Generation*. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3715908.

13. Li, J., Xu, J., Huang, S., Chen, Y., Li, W., Liu, J., Lian, Y., Pan, J., Ding, L., Zhou, H., Wang, Y., & Dai, G. (2024). *Large Language Model Inference Acceleration: A Comprehensive Hardware Perspective*. ArXiv, abs/2410.04466. https://doi.org/10.48550/arxiv.2410.04466.

14. Luccioni, A., Viguier, S., & Ligozat, A. (2022). *Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model*. J. Mach. Learn. Res., 24, 253:1–253:15. https://doi.org/10.48550/arxiv.2211.02001.

15. Meta AI. (2024). *The Llama 3 Herd of Models: Technical Report*. Meta AI Research. Available in https://arxiv.org/pdf/2407.21783.

16. Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., & Zaharia, M. (2021). *Efficient Large-Scale Language Model Training on GPU Clusters Using*

Megatron-LM. SC21: International Conference for High Performance Computing, Networking, Storage and Analysis, 1–14. https://doi.org/10.1145/3458817.3476209

17. OpenAI. (2024). *OpenAI o-series models*. OpenAI Technical Report. Available in https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf.

18. Ozcan, M., Wiesner, P., Weiss, P., & Kao, O. (2025). *Quantifying the Energy Consumption and Carbon Emissions of LLM Inference via Simulations*. ArXiv, abs/2507.11417. https://doi.org/10.48550/arxiv.2507.11417.

19. Singh, A., Patel, N., Ehtesham, A., Kumar, S., & Khoei, T. (2024). *A Survey of Sustainability in Large Language Models: Applications, Economics, and Challenges*. 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC), 00008–00014. https://doi.org/10.1109/ccwc62904.2025.10903774.

20. Wu, Y., Hua, I., & Ding, Y. (2025). *Unveiling Environmental Impacts of Large Language Model Serving: A Functional Unit View*. ArXiv, abs/2502.11256. https://doi.org/10.48550/arxiv.2502.11256.

21. Xiao, G., et al. (2023). *SmoothQuant: Accurate and Efficient Post-Training Quantization for LLMs*. arXiv preprint. https://arxiv.org/abs/2211.10438

22. Ye, T., Huang, W., Zhang, X., T., Liu, P., Yin, J., & Wang, W. (2025). *LLM4EFFI: Leveraging Large Language Models to Enhance Code Efficiency and Correctness*. ArXiv, abs/2502.18489. https://doi.org/10.48550/arxiv.2502.18489.

23. Yuan, Z., Sun, W., Liu, Y., Zhou, H., Zhou, R., Li, Y., Zhang, Z., Song, W., Huang, Y., Jia, H., Murugesan, K., Wang, Y., He, L., Gao, J., Sun, L., & Ye, Y. (2025). *EfficientLLM: Efficiency in Large Language Models*. ArXiv, abs/2505.13840. https://doi.org/10.48550/arxiv.2505.13840.