



ARTIGO

NOVOS DESAFIOS PARA A IA: A REDUÇÃO DA ENERGIA REQUERIDA NA INFERÊNCIA

POR

Luigi Carro
carro@inf.ufrgs.br

Avanços recentes em Grandes Modelos de Linguagem (LLMs) revolucionaram o desenvolvimento de software por meio de recursos de automação e raciocínio. O uso destas novas máquinas tem tido enorme impacto na economia, nas relações sociais e, obviamente, na Computação, pela automação da produção de software, como discutido na palestra que apresentei durante o SEMISH 2026. No entanto, a maioria dos modelos de ponta depende de processamento em nuvem e

é otimizada para computação de uso geral [1]. Isto tem impacto evidente no tamanho destas LLMs e consequente gasto energético para produção de resultados.

Para resolver o problema do consumo energético em nível de hardware, muito esforço foi feito no desenvolvimento de arquiteturas dedicadas, aceleradores e aritmética de baixa precisão. Embora eficazes, tais estratégias introduzem complexidade, limitam a acessibilidade e são dispendiosas para adoção em múltiplas plataformas hoje existentes. Um estudo

recente [2] confirma esta tendência, onde a vasta maioria das pesquisas em eficiência energética prioriza o hardware personalizado. Na outra ponta, alternativas que recorrem ao software, como quantização e uso de modelos menores (SLM), frequentemente implicam em queda de desempenho.

Neste trabalho, apresentamos um conceito para geração de software embarcado projetado ao redor de SLMs locais, executando sem depender de APIs em nuvem. A arquitetura proposta divide o processo de geração de software em etapas modulares, preservando a qualidade da saída e reduzindo a demanda por recursos. A validação desta proposta deu-se através de medições práticas do consumo de energia e memória em diferentes modelos de hardware disponíveis.

Estratégia de implementação

A estratégia utilizada para desenvolvimento foi baseada em agentes, inspirada em *Chain of Thoughts* [3] e *Tree of Thoughts* [4], na qual o processo de desenvolvimento de software é dividido em múltiplas etapas especializadas dentro de um fluxo iterativo, permitindo contornar problemas conhecidos, como janelas de contexto reduzidas e menor capacidade de geração do modelo.

Para estabelecer uma base de comparação, utilizamos inicialmente o ChatGPT-4 completo para gerar código, empregando o prompt da Figura 1.

“Using these 2 pdf documents with APIs for GUI and running tracker logic, in python. I want to make one system that measures how many meters I ran. There must be a button to start the race, and one to finish. The race must be stored together with the date, and how many minutes were spent, and at what speed. When I want, I should be able to see my performance, whether in graphical mode, either in a table, sorted by most recent days, or by months. The system should also alert me if I’m doing too little or if I’m being too aggressive.”



FIG. 01 | PROMPT UTILIZADO

A decomposição de problemas sempre foi uma estratégia clara na área da ciência da computação. Além disso, conforme demonstrado em abordagens como *chain-of-thoughts* e *tree-of-thoughts*, o uso de prompts menores e mais específicos pode elevar a qualidade da resposta de um LLM ao encadear as perguntas a serem respondidas. Aplicamos essa estratégia ao problema em questão, dividindo a tarefa de geração de software com base em nossas APIs e distribuindo o processo de desenvolvimento de software entre agentes.

Esse fluxo de trabalho multiagente permite que o sistema utilize modelos locais de forma mais eficaz, superando suas limitações individuais por meio da divisão de responsabilidades e garan-

tindo um processo de geração de software mais confiável.

A arquitetura proposta consiste em três agentes principais:

- Agente de Engenharia de Prompt (LLaMA 3.1 8B)

Recebe a solicitação do usuário; reformata a entrada em um prompt com mais detalhes de implementação, que permitem que a SLM tenha uma maior taxa de sucesso; encaminha o prompt estruturado para o Agente de Geração de Código.

- Agente de Geração de Código (LLaMA 3.1 8B Instruct)

Recebe o *prompt* estruturado do Agente de Engenharia de *prompt*; utiliza RAG para recuperar funções baseadas em APIs; gera soluções combinando diferentes funções das APIs; encaminha o código gerado para o Validador.

- Agente Validador (LLaMA 3.1 8B Instruct)

Executa o código gerado; registra erros de compilação ou execução; se erros forem detectados, devolve-os ao Agente de Geração de Código para refinamento; se o código estiver livre de erros, ele é entregue ao usuário.

Esse ciclo iterativo entre os agentes Gerador de Código e Validador continua até que o Agente Validador confirme que o software gerado está livre de erros.

Os resultados obtidos na prática foram muito similares, em comparação com máquinas muito maiores. A mesma janela foi criada, e a mesma biblioteca de APIs foi usada por ambas, ChatGPT-4 e as SLMs usadas neste trabalho. Com a qualidade da saída assegurada, focamos em avaliar o consumo energético.

Hardware escolhido e resultados

Nossa metodologia foi projetada para avaliar a eficiência energética do sistema utilizando hardware representativo de nível consumidor, refletindo cenários de implementação realistas e sem o uso de aceleradores especializados. Para viabilizar uma análise comparativa robusta, selecionamos um conjunto diversificado de GPUs cujas arquiteturas, processos de fabricação e sistemas de memória evoluíram significativamente. Os modelos NVIDIA GTX 1080 Ti, RTX 2060 Super e RTX 4070 Ti foram escolhidos para traçar uma clara progressão geracional, permitindo-nos mensurar como os avanços tanto no processamento central quanto na eficiência de memória impactam o desempenho por watt. Para garantir que nossa análise não se limitasse a um único fabricante, incluímos a AMD RX 7700 XT como um contraponto arquitetônico moderno. Esse conjunto de hardware selecionado oferece uma base abrangente para analisar as compensações críticas entre poder computacional e consumo de energia na execução local de LLMs. A Tabela 1 resume os resultados alcançados.

Tipo de máquina	Tempo (s)	Energia (J)	Energia (Wh)
Nvidia GTX 1080Ti	150,14	27.207,58	7,55
Nvidia RTX 2060	140,38	15.438,28	4,28
Nvidia RTX 4070Ti	82,05	14.501,81	4,02
AMD RX 7700XT	109,65	20.865,41	5,79



TAB. 01 | **RESULTADOS MEDIDOS**

A ordem utilizada na tabela inicia-se pela máquina mais antiga. Máquinas mais modernas têm melhor custo energético, porém o ganho é mais limitado. Essa aparente contradição explica-se pelo fato de que a RTX 4070 Ti, para alcançar seu desempenho superior, opera com um consumo de potência instantânea significativamente maior — uma estratégia que prioriza a velocidade em detrimento do consumo mínimo de energia total para a tarefa. Isso demonstra que depender exclusivamente da evolução do hardware para alcançar eficiência energética é uma estratégia de retornos decrescentes. É precisamente nesse ponto que nossa abordagem baseada em software se torna fundamental. Ao utilizar uma orquestração baseada em agentes para aplicar a estratégia de “dividir para conquistar”, o sistema decompõe a carga de trabalho em etapas menores e mais gerenciáveis. Essa estratégia não apenas mitiga as limitações de modelos de linguagem menores, mas também otimiza a utiliza-

ção dos recursos de hardware, permitindo que até mesmo GPUs mais antigas executem tarefas complexas de forma eficiente e com menor consumo de energia. É importante também observar o impacto do conjunto de instruções no consumo. A AMD disponibiliza apenas floating-point 64, o que para a aplicação alvo implica em desperdício. Assim, mesmo uma máquina mais moderna pode perder eficiência energética se seu conjunto de instruções não atende à aplicação adequadamente.

É fundamental contextualizar nossos resultados em relação aos valores publicados por grandes empresas, como a OpenAI, que reportam ~2,45 J/token. Na máquina com melhor resultado energético em nossos experimentos, foram consumidos 7,5 J/token, valor muito mais elevado. Contudo, nossas medições representam um custo energético de ponta a ponta, abrangendo todas as etapas do processo: desde o carregamento do modelo na VRAM e a inicialização do pipeline até a geração do token final. Trata-se de um cenário de “partida a frio”, representativo do uso sob demanda em condições reais em um ambiente local. Em contrapartida, os dados de referência publicados carecem de uma metodologia clara. Não se especifica se esses valores incluem apenas o tempo de inferência (diluindo o custo de carregamento do modelo ao longo de milhares de requisições) ou se desconsideram outras variáveis críticas, como o consumo de energia da rede, da refrigeração e da infraestrutura de servidores, o que torna imprecisa uma comparação direta. Nossos resultados, embora numericamente mais elevados que os

mencionados para o ChatGPT-4, estão em consonância com outros trabalhos acadêmicos que realizam medições energéticas transparentes em hardware de consumo [5].

Conclusão e trabalhos futuros

Este trabalho apresenta uma prova de conceito que demonstra a viabilidade de utilizar estratégias baseadas em software para aplicar a abordagem de “dividir para conquistar” em tarefas complexas. Ao decompor a carga de trabalho em agentes especializados, mitigamos algumas das ineficiências inerentes a modelos menores e prolongamos a vida útil do hardware

existente. Essa perspectiva alinha-se a um caminho alternativo na busca por eficiência energética em IA. Além da questão energética, o uso de software para alcançar o objetivo de redução de consumo permite uma enorme redução do custo de desenvolvimento, que também se alinha com outras vantagens no uso de SLMs locais, como desempenho competitivo, independência operacional e a privacidade dos dados.

Como trabalhos futuros, pretende-se desenvolver agentes baseados na sincronização das SLMs aqui demonstradas, onde a realimentação permitirá raciocínios complexos, a um custo reduzido.

Referências

1. ZHAO W.X. et al., “A Survey of Large Language Models,” arXiv preprint arXiv:2303.18223, 2023. [Online].
2. M. D. GOMONY, M.D. et al., “LLMs at the Edge: Hype or Hope?,” in Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2025.
3. WEI J. et al., “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” in Advances in Neural Information Processing Systems 35 (NeurIPS 2022), 2022, pp. 24824–24837. [Online].
4. YAO, S. et al., “Tree of Thoughts: Deliberate Problem Solving with Large Language Models,” in Advances in Neural Information Processing Systems 36 (NeurIPS 2023), 2023. [Online].
5. N. ALIZADEH, et al. “Analyzing the Energy and Accuracy of LLMs in Software Development” Available: <https://arxiv.org/html/2412.00329v1>



LUIGI CARRO é Professor Titular no Departamento de Informática Aplicada da Universidade Federal do Rio Grande do Sul. Atua na área de Sistemas Embarcados, Arquiteturas não Convencionais e Processamento de Sinais. Publicou centenas de artigos e seu índice H é 49. Sua pesquisa nos últimos 10 anos tem se focado em sistemas de baixa energia em hardware e software

