

Obsidian como Memória Externa em Grafo para Agentes de Codificação baseados em LLM

Matheus F. Queiroz Pinto¹, Giovani Apolinário Ocan¹, Jefferson Rodrigo Speck¹

¹ Faculdade Donaduzzi - Toledo, PR - Brazil
matheus.pinto@alunos.bpkedu.com.br
giovani.ocan@alunos.bpkedu.com.br
jeffersonspeck@msn.com

Abstract. *This study analyzes the use of Obsidian as a structured external memory layer for LLM-based coding agents. The investigated problem is the linear growth of conversational history, which increases token consumption and may degrade response quality in long-context scenarios. Methodologically, a prototype was built using interconnected Markdown files, organized into semantic types and accessed through the selective injection of relevant subgraphs. The preliminary results did not confirm the token-reduction scenario. Nevertheless, the approach demonstrated operational feasibility as a persistent and searchable repository of project knowledge, indicating paths for refinement based on filtering, summarization, and context control.*

Resumo. *Este estudo analisa o uso do Obsidian como uma camada de memória externa estruturada para agentes de codificação baseados em Modelos de Linguagem. O problema investigado é o crescimento linear do histórico conversacional, que aumenta o consumo de tokens e pode degradar a qualidade das respostas em contextos longos. Metodologicamente, foi construído um protótipo baseado em arquivos Markdown interconectados, organizados em tipos semânticos e acessados por injeção seletiva de subgrafos relevantes. Os resultados preliminares não confirmaram o cenário de redução de tokens. Ainda assim, a abordagem demonstrou viabilidade operacional como repositório persistente e consultável de conhecimento de projeto, indicando caminhos para refinamentos baseados em filtragem, sumarização e controle de contexto.*

1. Introdução

Agentes de codificação baseados em Modelos de Linguagem de Grande Escala (LLM) não possuem memória persistente entre sessões. A solução comum é manter uma conversa única que cresce ao longo do projeto, produzindo contexto que aumenta linearmente, elevando o custo por sessão e degradando a qualidade das respostas. [Liu et al. 2024] demonstraram queda superior a 30% na acurácia quando a informação relevante está no meio de contextos longos, fenômeno chamado *lost-in-the-middle*. Esse efeito é amplificado pela acumulação de erros, explorações descartadas e código irrelevante ao longo das sessões [Morph 2026].

Abordagens de Geração Ampliada por Recuperação (RAG) vetorial endereçam parcialmente o problema, mas operam sobre *chunks* sem relação estrutural, desperdiçando

*tokens*¹ com fragmentos isolados, sem capturar relações entre decisões arquiteturais, entidades e módulos do projeto [Packer et al. 2023]. Sistemas como MemGPT oferecem persistência, mas dependem de infraestrutura adicional e não são nativamente integráveis aos agentes mais utilizados [Packer et al. 2023].

O padrão LLM Wiki [Karpathy 2026] demonstrou que arquivos *Markdown* locais bastam como base consultável, sem bancos de vetores ou pipelines complexos. O Obsidian, por sua vez, implementa nativamente um grafo de documentos *Markdown* com *wikilinks*, estrutura análoga a um *knowledge graph* legível por humanos e LLMs.

Diante desses achados, coloca-se a seguinte pergunta de pesquisa: *a injeção seletiva de subgrafos relevantes, extraídos de uma memória externa estruturada em Obsidian, reduz o consumo de tokens por sessão em agentes de codificação baseados em LLM quando comparada a uma abordagem sem memória estruturada?*

Para além da redução quantitativa, a investigação explora três dimensões interdependentes, sendo elas custo computacional por sessão, persistência de contexto entre sessões e qualidade arquitetural das respostas. Essas dimensões podem não se mover na mesma direção, visto que um aumento no custo de *tokens* pode ser justificado por ganhos em persistência ou qualidade, e uma redução de *tokens* pode ocorrer às custas dessas outras dimensões.

2. Estudos Relacionados

Abordagens de recuperação estruturada indicam que grafos podem melhorar a seleção de contexto para LLM. O GraphRAG organiza documentos-fonte em índice de entidades e relações, gerando resumos de comunidades semanticamente associadas com ganhos sobre RAG convencional em bases extensas [Edge et al. 2024]. Complementarmente, *graph-walk compression* e KG-RAG mostram que a travessia estrutural pode reduzir o contexto e o consumo de *tokens* sem comprometer a acurácia [Zarrinkia et al. 2026, Soman et al. 2024].

Outra linha de pesquisa explora memórias externas desacopladas da sessão individual como alternativa ao crescimento contínuo do histórico conversacional. O A-MEM organiza memórias como notas atômicas interligadas, em estrutura próxima ao método Zettelkasten [Xu et al. 2025]. Em agentes de codificação, memórias específicas de projeto combinam resumos arquiteturais pré-gerados e aprendizados acumulados em sessões anteriores para reduzir custos operacionais mantendo a acurácia [Yeo et al. 2025]. Esses trabalhos evidenciam a relevância de grafos e memória persistente, mas ainda deixam espaço para arquiteturas leves, locais e integráveis ao fluxo real de desenvolvimento de software.

3. Metodologia e Execução

A proposta foi avaliada por meio da construção de um protótipo de memória externa para agentes de codificação baseados em LLM. A arquitetura foi organizada em três camadas: (C1) um *vault* do Obsidian, usado como repositório local de conhecimento em grafo;

¹Unidades mínimas de processamento de texto em modelos de linguagem, correspondentes aproximadamente a palavras ou subpalavras; modelos possuem um limite máximo de tokens que podem processar por vez.

(C2) um arquivo `AGENTS.md`, responsável por definir o protocolo de acesso à memória; e (C3) um agente de codificação baseado em LLM, que consulta seletivamente os arquivos do *vault* durante a execução das tarefas. Essa decomposição segue a compreensão clássica de arquitetura de software como organização de elementos, relações e decisões estruturais [Bass et al. 2021].²

O *vault* foi estruturado em arquivos *Markdown* conectados por *wikilinks* bidirecionais. Cada arquivo corresponde a um nó do grafo e representa decisões arquiteturais, entidades de domínio, padrões recorrentes, restrições do projeto ou resumos de sessões anteriores. Os nós possuem metadados em *frontmatter YAML*³, incluindo tipo semântico, data de atualização e relações com outros nós.

O protocolo de acesso tem três níveis: o arquivo `_meta/hot.md`, sempre disponibilizado ao agente, reúne os nós mais frequentes ou transversais; `_meta/index.md` permite localizar nós por módulo, tema ou tipo semântico; e os arquivos específicos são acessados apenas quando indicados pelo índice. Os nós foram organizados em cinco tipos semânticos: *architecture*, para decisões estruturais; *entity*, para entidades centrais do domínio; *pattern*, para padrões recorrentes de implementação; *constraint*, para restrições e regras do projeto; e *session*, para resumos compactados de sessões anteriores.

Para a avaliação preliminar, foi definido um experimento controlado com duas condições, seguindo a lógica de comparação entre tratamentos em Engenharia de Software [Wohlin et al. 2012]. Na primeira, denominada *baseline sem memória estruturada*, o agente recebeu o enunciado da tarefa sem consulta ao *vault*. Na segunda, denominada *proposta com memória estruturada*, o agente recebeu o mesmo enunciado, porém com injeção seletiva de contexto proveniente do subgrafo relevante do *vault*, conforme definido no `AGENTS.md`.

As condições foram aplicadas a tarefas equivalentes em um projeto de API REST, mantendo os requisitos e variando apenas a presença da memória. A métrica principal foi o total de *tokens* por condição (soma de `prompt_tokens` e `completion_tokens`). Complementarmente, registrou-se a aderência funcional da resposta ao requisito, evitando que uma redução de *tokens* fosse interpretada como ganho diante de respostas insuficientes.

A avaliação assume caráter preliminar e exploratório. O estudo restringe-se a duas condições principais, em vez de comparar contexto longo, contexto curto, RAG e *LLM-as-judge*, concentrando-se na hipótese central de que uma memória externa em grafo, implementada com Obsidian e *Markdown*, pode reduzir o consumo de *tokens* ao substituir o crescimento acumulativo do contexto pela recuperação seletiva de subgrafos.

4. Considerações Finais

Este estudo teve como objetivo investigar se a injeção seletiva de subgrafos relevantes, extraídos de uma memória externa estruturada em Obsidian, reduz o consumo de *tokens* por sessão em agentes de codificação baseados em LLM quando comparada a uma abordagem

²O repositório com os artefatos construídos para o experimento está disponível em: <https://github.com/MatheusFQueiroz/llm-graph-memory-experiment>.

³*Frontmatter* é um bloco inicial de metadados de um arquivo. Em *YAML*, esses metadados são descritos por pares chave-valor, como `type`, `last_updated` e `linked_to`.

sem memória estruturada. A pergunta de pesquisa formulada foi: *a injeção seletiva de subgrafos relevantes, extraídos de uma memória externa estruturada em Obsidian, reduz o consumo de tokens por sessão em agentes de codificação baseados em LLM quando comparada a uma abordagem sem memória estruturada?*

Com base nos dados desta avaliação, a resposta é negativa. O cenário C3 (memória externa via MCP) consumiu em média 276.614 *tokens* por sessão e C2 (sem memória estruturada) 128.882, acréscimo de cerca de 114%. Mesmo considerando apenas os *tokens* não servidos por cache, C3 foi cerca de 22% mais custoso que C2.

Esses resultados indicam que o custo de consulta e injeção da memória externa superou os possíveis benefícios da recuperação seletiva de contexto. Dois fatores principais explicam o comportamento: o *overhead* do protocolo de consulta ao *vault*, com inclusão de `hot.md` e múltiplos nós relacionados em cada sessão; e o crescimento progressivo dos próprios nós da memória, especialmente arquivos como `task.md`, que passaram a acumular decisões, implementações e registros anteriores. Assim, a memória externa reproduziu parcialmente o crescimento acumulativo de contexto que buscava mitigar.

Do ponto de vista qualitativo, a memória externa também não garantiu melhor orientação arquitetural. No cenário C2, o agente organizou o projeto desde a primeira sessão em camadas modulares (`routers/`, `schemas/`, `services/`), enquanto em C3 a implementação permaneceu concentrada em `main.py` até a última tarefa. O maior custo em C3 não veio acompanhado de ganho arquitetural, pelo contrário, veio acompanhado de aderência ao estado previamente registrado no *vault*, reduzindo o espaço para refatorações que o agente executou espontaneamente em C2. Esse resultado materializa o *trade-off* entre custo, persistência e qualidade, uma vez que a memória externa, na configuração atual, operou também como restrição implícita, fixando decisões iniciais e dificultando reorganizações posteriores. Menor consumo de *tokens* não implica necessariamente melhor desempenho, assim como maior consumo não implica melhor qualidade arquitetural.

A principal contribuição não consiste em evidência quantitativa de redução de *tokens*, mas em uma análise exploratória da viabilidade operacional de memórias externas em grafo para agentes de codificação baseados em LLM. Os achados caracterizam três dimensões interdependentes (custo, persistência e qualidade arquitetural) e demonstram que, em sua forma atual, a abordagem não otimiza nenhuma delas, uma vez que o consumo cresceu, a qualidade não melhorou e a persistência atuou mais como restrição do que como ganho. O Obsidian mostrou-se tecnicamente integrável ao fluxo, mas sua efetividade depende de mecanismos de filtragem, sumarização e ranqueamento que garantam ganho em persistência sem prejudicar a qualidade.

São limitações: o caráter preliminar do experimento, a ausência de replicação estatística, o uso de um domínio de API REST, a falha do servidor MCP na primeira sessão de C3 e a inexistência de uma métrica formal para avaliação da qualidade do código. Para trabalhos futuros, propõe-se avaliar a abordagem em projetos mais longos, investigar sumarização automática dos nós, limitar e ranquear injeções por tarefa e incorporar métricas de qualidade. Embora a hipótese inicial não tenha sido confirmada, os resultados delimitam condições necessárias para que memórias externas em grafo se tornem efetivas em agentes de codificação baseados em LLM.

Referências

- Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley Professional, 4 edition.
- Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., and Larson, J. (2024). From local to global: A graph RAG approach to query-focused summarization.
- Karpathy, A. (2026). Llm wiki.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Morph (2026). Context rot: The complete guide to why llms degrade as context grows.
- Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. (2023). Memgpt: Towards llms as operating systems.
- Soman, K., Rose, P. W., Morris, J. H., Akbas, R. E., Smith, B., Peetoom, B., Villouta-Reyes, C., Ceronio, G., Shi, Y., Rizk-Jackson, A., Israni, S., Nelson, C. A., Huang, S., and Baranzini, S. E. (2024). Biomedical knowledge graph-optimized prompt generation for large language models. *Bioinformatics*, 40(9):btac560.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer.
- Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. (2025). A-mem: Agentic memory for llm agents.
- Yeo, I., Ryu, D., and Baik, J. (2025). Improving LLM-based fault localization with external memory and project context.
- Zarrinkia, Y., Srinivasan, V., and Thomo, A. (2026). The reasoning bottleneck in Graph-RAG: Structured prompting and context compression for multi-hop QA.