Mitigação de Ataques de Canal Lateral Temporal em Sistemas de *Serving* de LLMs por Meio da Equalização de Latência *

Bruno Fonseca[1], **Uelinton Brezolin**[2], **Michele Nogueira**[1,2]

[1]Departamento de Ciência da Computação, Universidade Federal de Minas Gerais

[2]Departamento de Informática, Universidade Federal do Paraná

brunolopes@dcc.ufmg.br, uqbrezolin@inf.ufpr.br, michele@dcc.ufmg.br

Abstract. *LLMs introduce new security risks in serving systems, especially through temporal side channels exploited from response latency. Recent work shows that differences between cache hits and misses enable prompt-stealing attacks that can infer sensitive information processed by the system. Although mitigation proposals exist, many depend on reducing cache sharing or introducing additional communication and performance costs. This work presents a mitigation strategy based on equalizing the latency observed by the client, reducing the temporal distinction exploited by the attacker without disabling the caching mechanisms. The results show that the proposal reduced the attack recovery rate from up to 78.0% to 0.0%.*

Resumo. *Os LLMs introduzem novos riscos de segurança em sistemas de serving, especialmente por meio de canais laterais temporais explorados a partir da latência das respostas. Trabalhos recentes mostram que diferenças entre cache hits e cache misses permitem ataques de prompt stealing, capazes de inferir informações sensíveis processadas pelo sistema. Embora existam propostas de mitigação, muitas dependem da redução do compartilhamento de cache ou introduzem custos adicionais de comunicação e desempenho. Este trabalho apresenta uma estratégia de mitigação baseada na equalização da latência observada pelo cliente, reduzindo a distinção temporal explorada pelo atacante sem desabilitar os mecanismos de cache. Os resultados mostram que a proposta reduziu a taxa de recuperação do ataque de até 78,0% para 0,0%.*

1. Introdução

Os Grandes Modelos de Linguagem (do inglês, *Large Language Models* – LLMs) têm se consolidado como uma das tecnologias mais relevantes da computação contemporânea [Hagos et al. 2024]. Sua capacidade de compreender, gerar e transformar linguagem natural possibilita sua aplicação em tarefas relacionadas à cibersegurança, como análise de *logs*, triagem de alertas, identificação de padrões maliciosos, apoio à resposta a incidentes, detecção de vulnerabilidades e análise de códigos potencialmente inseguros [Zhang et al. 2025]. À medida que esses modelos são incorporados a sistemas reais e disponibilizados para múltiplos usuários, surgem novos desafios relacionados à segurança, à privacidade e à proteção de informações sensíveis. Esses sistemas ampliam a superfície

*Este trabalho foi apoiado pelo CNPq, pela CAPES – Código de Financiamento 001, pela FAPESP #2018/23098-0; e pelo Instituto Kunumi.

de ataque e introduzem riscos concretos de segurança e privacidade para usuários e provedores da tecnologia [Yao et al. 2024, Kibriya et al. 2024].

Os modelos LLMs são treinados com grandes volumes de dados provenientes de fontes amplas e heterogêneas, frequentemente coletados da Internet [Yan et al. 2025]. Esses conjuntos de dados contêm informações pessoais, documentos, credenciais e outros dados sensíveis que, ainda que não intencionalmente, acabam sendo incorporados ao processo de treinamento. Esses modelos memorizam e revelam trechos de seus dados de treinamento, evidenciando riscos concretos de vazamento de informações pelo próprio modelo [Carlini et al. 2021]. Um dos riscos é a abertura de canais laterais temporais (do inglês, *timing side-channels*), nos quais diferenças observáveis no tempo de resposta do sistema permitem inferir informações sensíveis sobre seu funcionamento interno. Esses sinais incluem padrões de execução, comportamento operacional e variações observáveis durante o processamento das requisições [Das et al. 2025]. Com a integração dos LLMs às arquiteturas cada vez mais complexas, os pontos de exposição deixam de se limitar ao modelo e passam a abranger também a infraestrutura de inferência, os mecanismos de *cache*, as filas de requisições, os servidores de aplicação e demais componentes auxiliares, ampliando os riscos relacionados a vazamentos de informações e canais laterais em sistemas de *serving* de LLMs [Ding et al. 2025].

Na literatura, trabalhos recentes têm demonstrado que otimizações empregadas em sistemas de *serving* de LLMs introduzem canais laterais capazes de comprometer a confidencialidade de *prompts* e consultas de usuários. Em [Song et al. 2025], os autores investigaram o impacto do compartilhamento de KV *cache* e de *caches* semânticos em serviços comerciais e *frameworks* de código aberto, demonstrando que diferenças de latência entre *cache hits* e *cache misses* são exploradas para inferir prefixos de *prompts* e consultas semanticamente relacionadas. De forma semelhante, em [Wu et al. 2025], os autores apresentaram o ataque PROMPTPEEK, que utiliza variações no tempo de atendimento e na ordem de escalonamento das requisições para reconstruir *prompts* de outros usuários em ambientes multi-inquilino. Em [Soleimani et al. 2025], os autores mostraram que padrões de tempo e tamanho de pacotes em APIs de *streaming* permitem inferir atributos sensíveis de *prompts* e respostas, mesmo quando o tráfego é protegido por *Transport Layer Security* (TLS). Na mesma linha, em [McDonald and Bar Or 2025], os autores apresentaram o ataque Whisper Leak, capaz de identificar tópicos sensíveis discutidos em interações com LLMs por meio da análise de metadados de rede. Por fim, em [Chu et al. 2025] mostrou mecanismo de compartilhamento seletivo de KV *cache* voltado à redução de vazamentos temporais em ambientes multi-inquilino. Embora esses trabalhos proponham mecanismos de mitigação, as soluções existentes geralmente reduzem a eficiência do *cache*, aumentam o *overhead* de comunicação ou comprometem a interatividade do sistema.

Este trabalho propõe um método para mitigação para ataques de canal lateral temporal em sistemas de *serving* de LLMs. A proposta busca impedir que um atacante diferencie *cache hits* e *cache misses* a partir da latência observada nas respostas do servidor. Para isso, o método introduz atrasos controlados em respostas do servidor, aproximando seus tempos observados em requisições que resultam em *cache misses* e em *cache hits*. Dessa forma, o método reduz a distinção temporal entre os dois casos, eliminando o canal e preservando os benefícios computacionais do *cache*, uma vez que a resposta continua

sendo gerada com menor custo de processamento. O método consiste em três etapas principais: *(i)* medição dos tempos de resposta associados aos *cache hits* e *cache misses*; *(ii)* definição de uma política de quantização dos tempos de resposta; e *(iii)* aplicação de atrasos artificiais em respostas produzidas mais rapidamente pelo servidor. A partir das medições realizadas, o método estabelece intervalos de referência para a latência observável e retarda respostas cuja geração ocorre abaixo desses intervalos. Com isso, respostas originadas de *cache hits* tornam-se temporalmente indistinguíveis de respostas originadas de *cache misses*, reduzindo a correlação entre o estado interno do *cache* e a latência observada externamente.

A avaliação do método proposto seguiu uma abordagem empírica composta por dois experimentos principais. No primeiro experimento, foi reproduzido o ataque de *prompt stealing* descrito na literatura, sem a aplicação de mecanismos de mitigação, com o objetivo de estabelecer uma linha de base para a efetividade do ataque em ambientes vulneráveis. No segundo experimento, o mesmo ataque foi executado novamente em um ambiente com a mitigação proposta ativada, permitindo avaliar o impacto da equalização de latência sobre a capacidade do atacante de inferir informações a partir do tempo de resposta do servidor. Os experimentos foram realizados em um ambiente de *serving* de LLMs com compartilhamento de KV *cache*, utilizando modelos da família Llama e um conjunto de dados composto por *system prompts* sintéticos voltados à avaliação de vazamento de informações em aplicações baseadas em LLMs [Chu et al. 2025]. A avaliação considerou métricas relacionadas à recuperação de *tokens*, taxa de falsos positivos, número médio de consultas realizadas pelo atacante, e quantidade de *cache hits* mitigados. Os resultados obtidos demonstram que a política de atrasos sintéticos reduziu significativamente a diferença temporal observável entre *cache hits* e *cache misses*, impedindo a inferência de informações sensíveis a partir da latência das respostas.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve o método proposto. A Seção 4 detalha a metodologia de avaliação adotada e discute os resultados preliminares obtidos. Por fim, a Seção 5 apresenta as conclusões do trabalho.

2. Trabalhos Relacionados

Diversos trabalhos investigam riscos de segurança e privacidade em sistemas baseados em LLMs, em especial os canais laterais introduzidos por otimizações empregadas durante o processo de *serving* [Zhao et al. 2026]. Esses estudos demonstram que mecanismos projetados para reduzir latência e aumentar o *throughput*, como compartilhamento de KV *cache*, *speculative decoding* e transmissão incremental de *tokens*, expõem informações sensíveis por meio de diferenças observáveis no tempo de resposta e nos padrões de comunicação. No contexto de canais laterais em *caches* compartilhados, em [Song et al. 2025], os autores investigaram como mecanismos de *cache* compartilhada em sistemas de *serving* de LLMs podem introduzir canais laterais temporais capazes de vazar informações sensíveis de *prompts* e consultas de usuários. Os autores demonstraram que diferenças de latência entre *cache hits* e *cache misses* permitem a um adversário, com acesso apenas em modo *black-box*, inferir prefixos de *prompts* e identificar consultas semanticamente relacionadas. O mecanismo foi avaliado em ataques contra

serviços comerciais e *frameworks* amplamente utilizados, reportando elevadas taxas de sucesso na reconstrução de informações privadas. Como mitigação, eles propuseram restringir o compartilhamento do *cache* em blocos maiores de *tokens* e adotar mecanismos de anonimização em *caches* semânticos, reduzindo a granularidade do reaproveitamento e, consequentemente, a capacidade do atacante de distinguir o estado interno do sistema.

Os autores em [Soleimani et al. 2025] analisaram canais laterais observáveis no tráfego de rede de serviços interativos de LLMs, mostrando que padrões de tempo e tamanho dos pacotes permitem inferir atributos sensíveis dos *prompts* e das respostas, mesmo sob proteção do protocolo TLS. Os autores formalizam a ideia de indistinguibilidade para transcrições de comunicação e demonstram ataques capazes de explorar otimizações como *speculative decoding*. As avaliações em sistemas como vLLM e ChatGPT revelam acurácia entre 50% e 92% na inferência de atributos privados. Os autores também analisam contramedidas baseadas em agregação e atraso no envio de *tokens*, destacando o compromisso entre proteção, interatividade e *overhead* de comunicação. Na mesma linha, [McDonald and Bar Or 2025] apresentaram o ataque *Whisper Leak*, que utiliza sequências de tamanhos de pacotes e intervalos entre transmissões para classificar tópicos sensíveis discutidos em interações com LLMs. Avaliando 28 modelos comerciais, os autores mostram que um observador passivo de rede identifica temas como lavagem de dinheiro, condições médicas ou assuntos políticos com elevada precisão, frequentemente superior a 98% em termos de Área sob a Curva de Precisão-Revocação (AUPRC). O estudo demonstrou que o vazamento independe do acesso ao conteúdo das mensagens e decorre de padrões temporais e estruturais inerentes ao processo de geração e transmissão das respostas. As estratégias de mitigação avaliadas, como *padding* aleatório e agrupamento de *tokens*, reduzem parcialmente o problema, mas não eliminam completamente o risco.

Os autores em [Wu et al. 2025] apresentaram o ataque PROMPTPEEK, que explora o compartilhamento de KV *cache* em ambientes multi-inquilino para reconstruir *prompts* submetidos por outros usuários. A técnica utiliza alterações no tempo de atendimento e na ordem de escalonamento das requisições para detectar *cache hits* e recuperar os *tokens* do *prompt* de forma incremental. Os experimentos, realizados sobre *frameworks* como SGLang e vLLM, demonstram taxas de sucesso superiores a 95% mesmo em cenários com pouco conhecimento prévio do adversário. O estudo evidenciou otimizações projetadas para reduzir uso de memória e latência, comprometendo a confidencialidade dos dados processados em ambientes compartilhados. Diante desses riscos, trabalhos recentes também passaram a investigar mecanismos de mitigação para canais laterais temporais em sistemas de inferência de LLMs. Em [Chu et al. 2025], os autores propuseram o SafeKV, um mecanismo de compartilhamento seletivo de KV *cache* voltado à redução de vazamentos temporais em ambientes multi-inquilino. A proposta combina isolamento parcial de entradas sensíveis, monitoramento de acessos e separação entre regiões públicas e privadas do *cache*, reduzindo a capacidade de inferência baseada em *cache hits*. Embora a abordagem reduza significativamente os ataques de canal lateral, ela modifica a política de gerenciamento do *cache* e introduz custos adicionais de controle e isolamento, impactando o desempenho e o reaproveitamento global do KV *cache*.

Em todos esses casos, o vazamento de informações decorre principalmente da existência de diferenças temporais observáveis externamente, caracterizando canais late-

rais temporais (*timing side-channels*) exploráveis por atacantes em um cenário *black-box*. Embora diferentes otimizações de *serving* e mecanismos de transmissão em sistemas baseados em LLMs introduzam esses canais laterais, as soluções propostas na literatura concentram-se na redução do compartilhamento de cache, no aumento do volume de dados transmitidos ou na modificação dos protocolos de comunicação, o que implica perda de eficiência e maior custo operacional. Em contraste, este trabalho propõe uma mitigação baseada na equalização controlada da latência, na qual atrasos artificiais são aplicados às respostas do servidor para aproximar a latência observável de *cache hits* e *cache misses*. Dessa forma, busca-se reduzir a correlação entre o estado interno do *cache* e a latência observada externamente, eliminando o canal temporal no cenário avaliado sem alterar a estrutura do *cache*, sem aumentar o consumo da Unidade de Processamento Gráfico (GPU) ou memória e preservando os benefícios computacionais do *serving*.

3. Proposta

Esta seção descreve o método proposto para mitigar ataques de canal lateral temporal em sistemas de *serving* de LLMs que utilizam mecanismos de *cache* compartilhada. A proposta atua sobre a latência observada pelo cliente, introduzindo atrasos sintéticos em respostas produzidas mais rapidamente pelo servidor. O objetivo é reduzir a diferença temporal entre requisições que resultam em *cache hits* e requisições que resultam em *cache misses*, dificultando que um atacante utilize o tempo de resposta como indicador do estado interno da memória *cache*. A ideia central do método é separar o ganho computacional interno da memória *cache* da latência externamente observável. Em um funcionamento tradicional, uma requisição beneficiada por *cache hit* tende a ser respondida mais rapidamente, pois parte do processamento foi previamente realizado. Essa redução de tempo, embora desejável do ponto de vista de desempenho, revela ao cliente informações sobre requisições processadas anteriormente. A proposta, portanto, mantém o uso do *cache* para reduzir o custo de processamento, mas retarda o envio de determinadas respostas para mascarar a vantagem temporal associada ao *cache hit*.

O método utiliza estimativas dos tempos de resposta associados a *cache hits* e *cache misses*. A partir dessas estimativas, define-se uma política de normalização da latência, responsável por determinar o intervalo de tempo no qual as respostas devem ser observadas pelo cliente. Durante a execução, o servidor aplica atrasos sintéticos às respostas de modo que tanto respostas de requisições associadas a *cache hits* quanto respostas de requisições associadas a *cache misses* sejam retornadas dentro de uma mesma faixa temporal. Como os *cache misses* naturalmente demandam maior tempo de processamento, o atraso aplicado a essas respostas tende a ser menor, já que respostas associadas a *cache hits*, por serem produzidas mais rapidamente, recebem atrasos maiores. Dessa forma, as respostas internamente rápidas deixam de apresentar uma diferença temporal facilmente distinguível em relação às respostas que não se beneficiam do *cache*. A Figura 1 apresenta uma visão geral do método proposto, ilustrando como a mitigação atua sobre a latência observada externamente para reduzir a capacidade de inferência do atacante.

O método compreende três etapas principais: *(i)* estimativa da latência associada a *cache hits* e *cache misses*, utilizada para caracterizar a diferença temporal explorável pelo atacante; *(ii)* definição da política de quantização, responsável por agrupar as respostas em

faixas temporais de referência; e *(iii)* aplicação dos atrasos sintéticos, que ajusta a latência observada pelo cliente de acordo com a política definida. As próximas subseções detalham cada uma dessas etapas e discutem como elas contribuem para reduzir a correlação entre o estado interno do *cache* e a latência observada pelo cliente.

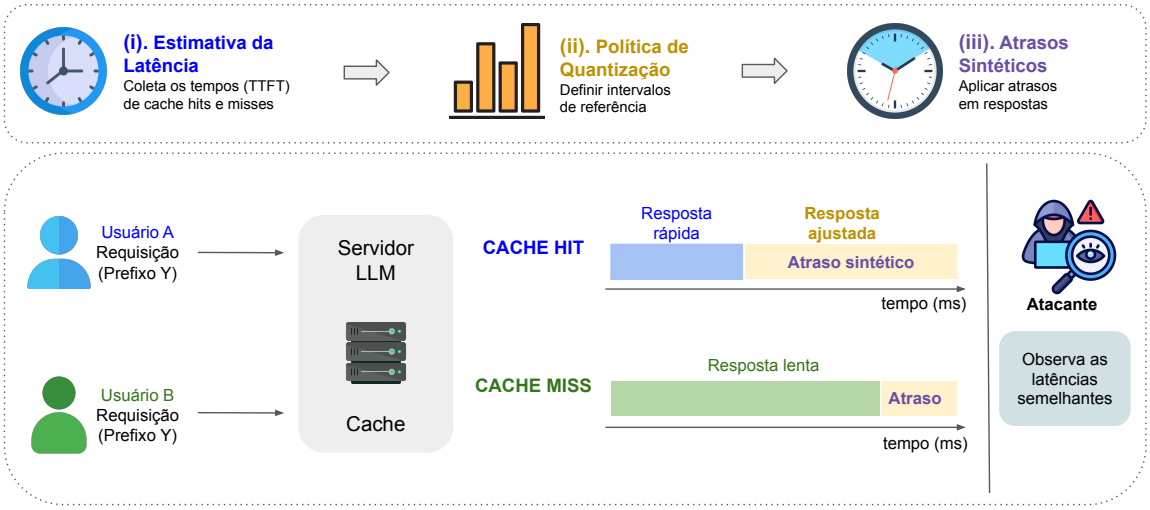

Figura 1. Visão Geral do Método Proposto

3.1. Estimativa da Latência

Essa etapa tem como objetivo caracterizar a diferença temporal observável entre respostas associadas a *cache hits* e *cache misses*. Para isso, o método realiza medições de latência sobre requisições de controle e requisições candidatas, permitindo estimar a separação temporal entre os dois comportamentos e identificar o grau de distinção explorável pelo atacante. A principal métrica utilizada é o TTFT (*Time To First Token*), correspondente ao tempo transcorrido entre o envio da requisição e o recebimento do primeiro *token* da resposta. Essa métrica representa o custo da etapa de *prefill*, na qual o modelo processa os *tokens* do *prompt* de entrada antes do início da geração da resposta.

Durante a fase de estimativa, o método coleta múltiplas medições de TTFT para diferentes pares de requisições. A requisição de controle corresponde a uma entrada sem reaproveitamento esperado de *cache*, enquanto a requisição candidata representa a entrada avaliada quanto à ocorrência de *cache hit*. Para cada par de medições, calcula-se a diferença temporal entre as respostas observadas, definida na Eq. 1.

$$\Delta = TTFT_{controle} - TTFT_{candidata} \tag{1}$$

Valores positivos indicam que a requisição candidata foi respondida mais rapidamente, sugerindo a ocorrência de um *cache hit*. A distribuição das medições obtidas permite estimar a separação temporal entre os dois comportamentos, caracterizando a intensidade do canal lateral observável externamente. As distribuições estimadas são utilizadas posteriormente para definir os parâmetros da política de quantização, especialmente a largura dos intervalos temporais empregados para normalizar as respostas do sistema.

3.2. Política de Quantização

Após a estimativa das latências associadas a *cache hits* e *cache misses*, o método define uma política de quantização para reduzir a precisão temporal disponível ao atacante. Essa política substitui a latência exata observada pelo cliente por faixas discretas de tempo. Assim, em vez de retornar cada resposta imediatamente após sua geração, o servidor libera as respostas apenas em instantes pertencentes a intervalos previamente definidos. A definição desses intervalos é feita na inicialização do servidor, a partir das distribuições de latência estimadas para *cache hits* e *cache misses*. Seja t o tempo interno necessário para gerar uma resposta, e seja W a largura do intervalo de quantização. A política divide a linha do tempo em faixas consecutivas de tamanho W, dadas pela Eq. 2.

$$I_k = [T_0 + kW, T_0 + (k+1)W) \tag{2}$$

T_0 representa o tempo inicial de referência e k indica o índice do intervalo. Para cada resposta, o método identifica a faixa à qual sua latência interna pertence e retarda seu envio até o limite superior dessa faixa, definindo uma faixa de liberação comum. Dessa forma, a latência observada pelo cliente deixa de ser o tempo exato de processamento e passa a ser uma versão quantizada desse tempo. A escolha de W utiliza como referência a separação temporal observada entre as distribuições de *cache hits* e *cache misses*. Como os *cache misses* naturalmente apresentam maior tempo de processamento, por exigirem a execução completa do *prefill*, seus tempos de resposta servem como referência para definir uma faixa que também englobe as respostas associadas a *cache hits*. Assim, respostas originadas de *cache hits* e *cache misses* são liberadas dentro de um mesmo intervalo temporal curto, reduzindo a distinção entre elas do ponto de vista do cliente.

A Figura 1 ilustra esse funcionamento. Sem a política de quantização, respostas associadas a *cache hits* são observadas antes das respostas associadas a *cache misses*, criando uma diferença temporal explorável. Com a política proposta, ambas as respostas são liberadas dentro de uma mesma faixa temporal, fazendo com que o atacante observe tempos de resposta próximos, mesmo quando o tempo interno de processamento é diferente. A escolha dos intervalos de quantização representa um compromisso entre segurança e desempenho. Os intervalos muito pequenos preservam melhor a latência original do sistema, mas mantêm diferenças temporais exploráveis por medições repetidas. Por outro lado, os intervalos amplos aumentam a semelhança temporal entre *cache hits* e *cache misses*, mas introduzem maior atraso percebido pelo usuário. Assim, a política de quantização deve reduzir a separação temporal explorável pelo ataque sem impor atrasos excessivos às respostas do servidor.

Como saída dessa etapa, são definidos os parâmetros que orientam a mitigação em tempo de execução: o tempo de referência, a largura dos intervalos de quantização e a regra que determina latência-alvo de liberação de cada resposta. Esses parâmetros são utilizados na etapa seguinte para calcular os atrasos sintéticos aplicados antes do envio das respostas ao cliente.

3.3. Aplicação dos Atrasos Sintéticos

Esta etapa recebe, como entrada, os parâmetros definidos pela política de quantização: o tempo de referência, a largura dos intervalos e a regra de liberação das respostas. A partir

desses valores, o servidor calcula, para cada requisição, se a resposta deve ser enviada imediatamente ou se deve ser retardada para se enquadrar na faixa temporal definida pela política de mitigação. Após a definição da política de quantização, o servidor passa a aplicar atrasos sintéticos às respostas antes de enviá-las ao cliente. Esse procedimento é configurado na inicialização do servidor, quando são carregados os parâmetros da política de mitigação, como o tempo de referência, a largura dos intervalos de quantização e os limites de latência definidos a partir das medições de *cache hits* e *cache misses*. A partir desses parâmetros, o servidor determina a faixa temporal na qual cada resposta deve ser observada pelo cliente.

Durante a execução, para cada requisição recebida, o servidor registra o instante de chegada da requisição e executa normalmente o processamento interno do modelo. Quando a resposta está pronta, o servidor calcula a latência interna de processamento, isto é, o tempo decorrido entre o recebimento da requisição e a geração da resposta. Em seguida, essa latência é comparada com o instante de liberação definido pela política de quantização. Caso a resposta tenha sido produzida antes desse instante, o servidor retarda seu envio por meio de um atraso sintético. Formalmente, seja t_{int} a latência interna medida pelo servidor para uma determinada requisição, e seja t_{lib} o instante de liberação definido pela política de quantização. A Eq. 3 define o atraso sintético aplicado à resposta.

$$d = \max(0, t_{lib} - t_{int}) \tag{3}$$

Assim, quando a resposta já se encontra dentro ou acima do tempo de liberação definido pela política, nenhum atraso adicional é aplicado, conforme definido pela Eq. 4.

$$t_{int} \geq t_{lib} \tag{4}$$

Quando a resposta é gerada antes desse limite, o servidor retarda seu envio por d unidades de tempo. Dessa forma, respostas com tempos internos diferentes passam a ser observadas pelo cliente dentro de uma mesma faixa temporal de referência. Assim, o método aplica atrasos tanto em respostas originadas de *cache hits* quanto em respostas originadas de *cache misses*. A diferença está na magnitude do atraso aplicado. Como respostas associadas a *cache misses* naturalmente demandam maior tempo de processamento, o atraso necessário para enquadrá-las na faixa temporal definida tende a ser menor. Por outro lado, respostas associadas a *cache hits*, por serem produzidas mais rapidamente, recebem atrasos maiores para que sua latência observável se aproxime da mesma faixa de resposta dos *cache misses*.

A aplicação de atraso também em respostas associadas a *cache misses* é necessária para evitar que o próprio mecanismo de mitigação introduza um novo padrão temporal observável. Caso apenas respostas associadas a *cache hits* fossem retardadas, essas respostas tenderiam a se concentrar em um comportamento artificial específico, enquanto respostas associadas a *cache misses* continuariam apresentando sua distribuição natural de latência. Essa diferença ainda poderia ser explorada por um atacante por meio de medições repetidas. Ao ajustar ambos os casos para uma mesma faixa temporal, a política reduz a distinção disponível ao atacante sem desabilitar o uso da *cache*. A Figura 1 ilustra esse procedimento. Internamente, uma resposta associada a *cache hit* é produzida mais rapidamente do que uma resposta associada a *cache miss*. Entretanto, após a aplicação dos

atrasos sintéticos, ambas são liberadas dentro da mesma faixa temporal observável. Com isso, o *cache* continua sendo utilizada para reduzir o custo computacional do servidor, mas sua vantagem temporal deixa de ser diretamente percebida pelo cliente.

4. Avaliação

A avaliação do método proposto foi realizada empiricamente por meio de dois experimentos principais. No primeiro experimento, foi reproduzido o ataque de *prompt stealing* baseado no trabalho de [Song et al. 2025], sem a aplicação de mecanismos de mitigação, com o objetivo de estabelecer uma linha de base para a taxa de sucesso do ataque em um ambiente vulnerável. No segundo experimento, o mesmo ataque foi executado novamente em um ambiente com o método proposto implementado, permitindo avaliar o impacto da mitigação sobre a capacidade do atacante de inferir informações a partir da latência das respostas. Em ambos os experimentos, foi utilizado um conjunto de dados composto por *system prompts* sintéticos voltados à avaliação de vazamento de informações em aplicações baseadas em LLMs [Chu et al. 2025]. Os experimentos foram realizados com dois modelos da família Llama: TinyLlama-1.1B e Llama-3.2-3B. Em ambos os casos, os modelos foram executados no servidor de *serving* e utilizados para geração dos *tokens* de saída, permitindo medir o TTFT e avaliar a diferença temporal entre requisições associadas ao *cache hits* e *cache misses*.

A avaliação foi conduzida a partir da perspectiva de um cliente atacante que interage com um servidor de *serving* de LLMs por meio de requisições comuns. Em sistemas que utilizam compartilhamento de *cache*, múltiplos usuários podem se beneficiar de estados internos previamente aquecidos por outras requisições. Embora esse mecanismo reduza o custo de inferência e melhore o desempenho do servidor, ele também cria um canal lateral temporal, no qual é difícil diferenciar um cliente legítimo de um cliente mal-intencionado tentando explorar o estado interno do *cache*. Requisições que resultam em *cache hits* tendem a apresentar menor latência do que requisições que resultam em *cache misses*, diferença explorada pelo atacante. Nesse cenário, a avaliação reproduz o fluxo do ataque de *prompt stealing*. Inicialmente, são enviadas requisições que simulam o comportamento de um cliente legítimo, aquecendo o estado interno do *cache* do servidor. Em seguida, um cliente atacante realiza consultas candidatas e mede a latência das respostas obtidas. A partir dessas medições, o atacante busca distinguir quais consultas se beneficiaram do *cache*, inferindo informações sobre *prompts* previamente processados pelo sistema. A comparação entre os dois experimentos permite verificar se a política de atrasos sintéticos reduz a diferença temporal explorável entre *cache hits* e *cache misses* e, consequentemente, diminui a efetividade do ataque.

4.1. Conjunto de Dados

Para a avaliação, foi utilizado o conjunto de dados `system-prompt-leakage` [Chua 2024], composto por *system prompts* sintéticos desenvolvidos para estudos de vazamento de informações em aplicações baseadas em LLMs. O conjunto de dados contém 283.353 amostras para treinamento e 71.351 amostras para teste, incluindo exemplos de vazamento direto e indireto de *prompts* do sistema. Nos casos de vazamento direto, as respostas reproduzem total ou parcialmente o conteúdo do *prompt* original, frequentemente por meio de pequenas substituições lexicais ou paráfrases simples. Já os casos de vazamento indireto envolvem reformulações mais

complexas, nas quais a resposta preserva o significado ou as instruções contidas no *prompt* sem reproduzi-lo. O conjunto de dados foi originalmente projetado para avaliação de mecanismos de detecção de *prompt leakage*, sendo utilizado neste trabalho para reproduzir cenários de compartilhamento de KV *cache* e analisar diferenças temporais entre *cache hits* e *cache misses* em sistemas de *serving* de LLMs.

4.2. Métricas de Avaliação

Para comparar os dois experimentos, foram utilizadas métricas relacionadas tanto à efetividade do ataque quanto ao custo necessário para executá-lo. A métrica *tokens* avaliados representa o número total de posições do *prompt* consideradas no experimento. Os *tokens* recuperados, ou verdadeiros positivos (TP), correspondem aos casos em que o atacante identifica corretamente o próximo *token* a partir da diferença de latência entre a requisição candidata e a requisição de controle. As falsas identificações, ou falsos positivos (FP), representam os casos em que o atacante classifica incorretamente um candidato como o próximo *token* correto. Já as falhas de detecção, ou falsos negativos (FN), correspondem aos casos em que o *token* correto não é recuperado pelo ataque.

A taxa de recuperação do ataque é calculada pela razão entre o número de *tokens* recuperados e o total de *tokens* avaliados. A taxa de falsos positivos corresponde à razão entre o número de falsas identificações e o total de *tokens* avaliados. Além dessas métricas, também foram medidas as tentativas médias por *token* recuperado, que indicam quantos candidatos são testados, em média, até a identificação correta de um *token*, e as consultas médias por *token*, que representam o número médio de requisições realizadas ao servidor durante o processo de ataque. Nos experimentos com mitigação, também foi considerada a quantidade de *cache hits* mascarados pela mitigação, isto é, os casos em que uma requisição candidata produziria reaproveitamento interno do *cache*, mas a política de equalização de latência impediu que essa informação fosse inferida pelo atacante.

4.3. Detalhamento do Processo de Mitigação

A primeira etapa da avaliação consistiu na reprodução do ataque de *prompt stealing* proposto por [Song et al. 2025]. A lógica central do ataque consiste em explorar diferenças no *TTFT* entre requisições que resultam em *cache hits* e requisições que resultam em *cache misses*. Como o reaproveitamento do KV *cache* reduz o custo da etapa de *prefill*, uma requisição cujo prefixo já foi previamente processado pelo servidor tende a receber o primeiro *token* da resposta mais rapidamente do que uma requisição equivalente que não se beneficia do *cache*. Essa diferença permite ao atacante inferir se determinada sequência de *tokens* já está presente no estado compartilhado do *cache*. Assim, o ataque foi realizado a partir de dois papéis simulados: um cliente legítimo e um cliente atacante. Inicialmente, o cliente legítimo envia requisições ao servidor, fazendo com que determinados prefixos ou trechos de *prompts* sejam armazenados no KV *cache*. Em seguida, o atacante tenta recuperar o conteúdo previamente processado de forma incremental. Para cada posição do *prompt*, o atacante gera um conjunto de *tokens* candidatos e constrói requisições contendo o prefixo já recuperado acrescido de cada candidato. Cada requisição candidata é enviada ao servidor e sua latência é medida.

A principal métrica utilizada foi o *TTFT* (*Time To First Token*), que representa o intervalo entre o envio da requisição e o recebimento do primeiro *token* da resposta. Essa

métrica foi extraída em modo *streaming*, pois captura diretamente o custo inicial de processamento do *prompt*, antes da geração completa da saída. Dessa forma, o *TTFT* evidencia diferenças entre requisições que reutilizam prefixos presentes no *cache* e requisições que exigem o processamento completo do *prefill*. Antes da recuperação dos *tokens*, foi realizada uma fase de calibração do canal temporal. Nessa fase, foram coletadas medições de latência para requisições conhecidamente associadas a *cache hits* e *cache misses*. Essas medições permitiram estimar a separação temporal entre os dois casos e definir limiares de decisão para classificar novas requisições. Como o *TTFT* varia de acordo com o tamanho do *prompt* e com ruídos do ambiente de execução, a classificação não foi baseada apenas em uma latência absoluta. Comparou-se a latência de cada requisição candidata com a latência de uma requisição de controle, construída para produzir *cache miss*.

A classificação de cada *token* candidato foi feita a partir da diferença entre o *TTFT* da requisição de controle e o *TTFT* da requisição candidata. Valores positivos e acima do limiar definido na fase de calibração indicam que a requisição candidata foi respondida mais rapidamente que a requisição de controle, sugerindo a ocorrência de *cache hit*. Nesse caso, o *token* testado é considerado uma provável continuação correta do *prompt*. Caso contrário, o candidato é descartado e o próximo *token* é testado. Para reduzir o impacto de ruídos, as medições foram agregadas utilizando a mediana do *TTFT* de 10 requisições iguais. No procedimento realizado, a busca incremental foi limitada a um número máximo de 80 tentativas por posição do *prompt*, interrompendo a busca quando o *token* correto não era identificado dentro desse limite. A efetividade do ataque foi avaliada por métricas como taxa de recuperação dos *tokens*, número médio de consultas por *token* recuperado, taxa de falsos positivos, e taxa de falsos negativos.

4.4. Resultados Preliminares

Esta subseção apresenta os resultados dos experimentos descritos anteriormente. Inicialmente, são apresentados os resultados do ataque de *prompt stealing* sem mitigação, utilizados como linha de base para avaliar a efetividade do ataque. Em seguida, são apresentados os resultados obtidos com a mitigação ativada no modelo de 3 bilhões de parâmetros, que apresentou a maior taxa de recuperação no cenário sem defesa. A Tabela 1 apresenta os resultados obtidos sem mitigação. No modelo de 1,1 bilhão de parâmetros, o ataque recuperou 255 dos 500 *tokens* avaliados, resultando em uma taxa de recuperação de 51,0%. Nesse cenário, também foram observadas 89 identificações falsas, correspondentes a uma taxa de falsos positivos de 17,8%. O ataque exigiu, em média, 19,7 tentativas por *token* e 1156 consultas por *token* recuperado.

No modelo de 3 bilhões de parâmetros, o ataque apresentou melhor desempenho. Foram recuperados 390 dos 500 *tokens* avaliados, resultando em uma taxa de recuperação de 78,0%. Além disso, não foram observadas identificações falsas, produzindo uma taxa de falsos positivos de 0,0%. O custo médio do ataque também foi menor, com 10,3 tentativas por *token* e 852,7 consultas por *token*. Esses resultados indicam que, nesse cenário experimental, o modelo maior apresentou uma separação temporal mais explorável entre *cache hits* e *cache misses*, permitindo maior recuperação de *tokens* com menor número médio de tentativas.

Após a caracterização do ataque sem defesa, a mitigação proposta foi aplicada ao modelo de 3 bilhões de parâmetros, por ter sido o cenário em que o ataque apresentou maior efetividade. A Tabela 2 apresenta os resultados após a aplicação dos atrasos

Tabela 1. Resultados do ataque de *prompt stealing* sem mitigação.

Métrica	LLM 1,1B	LLM 3B
Tokens avaliados	500	500
Tokens recuperados (TP)	255	390
Falsas identificações (FP)	89	0
Falhas de detecção (FN)	156	110
Taxa de recuperação	51,0%	78,0%
Taxa de falsos positivos	17,8%	0,0%
Tentativas médias por token recuperado	19,7	10,3
Consultas médias por token	1156,0	852,7

sintéticos. Com a mitigação ativada, o ataque não recuperou nenhum dos 500 *tokens* avaliados, resultando em taxa de recuperação de 0,0%. Além disso, não foram observadas falsas identificações, mantendo a taxa de falsos positivos em 0,0%. O mecanismo de defesa bloqueou 9872 ocorrências de *cache hits* que poderiam ser exploradas pelo atacante, resultando em uma taxa de mitigação de 100,0% no cenário avaliado.

Esse resultado ocorre porque, mesmo nos casos em que a requisição candidata corresponderia ao próximo *token* correto e produziria um *cache hit* internamente, a política de equalização de latência impede que essa diferença seja observada externamente. Assim, a latência da requisição candidata deixa de se diferenciar de forma confiável da latência da requisição de controle. Como consequência, o atacante não consegue classificar o candidato como *cache hit* e passa a tratá-lo como erro ou *cache miss*. Dessa forma, a mitigação não impede necessariamente a ocorrência interna do reaproveitamento de *cache*, mas remove o sinal temporal utilizado pelo ataque para identificar o *token* correto. Esses resultados indicam que, no cenário avaliado, a política de equalização de latência eliminou a diferença temporal explorável entre *cache hits* e *cache misses*, impedindo que o atacante use o tempo de resposta como indicador confiável do estado interno do *cache*.

Tabela 2. Resultado do ataque com a mitigação aplicada ao modelo LLM 3B.

Métrica	LLM 3B com mitigação
Tokens avaliados	500
Tokens recuperados (TP)	0
Cache hits mitigados/bloqueados	9872
Falsas identificações (FP)	0
Falhas de detecção (FN)	500
Taxa de recuperação do ataque	0,0%
Taxa de mitigação da defesa	100,0%
Taxa de falsos positivos	0,0%
Tentativas médias por token recuperado	0,0
Consultas médias por token	2400,0

Discussão dos resultados

Os resultados obtidos mostram que o ataque de *prompt stealing* é efetivo quando não há mecanismos de mitigação sobre a latência observável pelo cliente. Nos dois modelos avaliados, o atacante conseguiu recuperar parte dos *tokens*-alvo a partir da distinção

temporal entre requisições associadas a *cache hits* e requisições associadas a *cache misses*. Esse comportamento confirma que o tempo de resposta funciona como um canal lateral capaz de revelar informações sobre o estado interno do *cache*. Nesse contexto, a diferença de desempenho entre os modelos também é relevante. No modelo de 1,1 bilhão de parâmetros, o ataque recuperou 51,0% dos *tokens* avaliados, enquanto no modelo de 3 bilhões de parâmetros a taxa de recuperação chegou a 78,0%. Além disso, o modelo de 3 bilhões de parâmetros apresentou menor número médio de tentativas por *token* e menor número médio de consultas por *token* recuperado. No cenário avaliado, o modelo maior apresentou condições mais favoráveis à exploração do canal temporal.

Uma explicação é que modelos maiores tendem a apresentar maior custo de processamento na etapa de *prefill*, tornando mais evidente a diferença entre requisições que reutilizam dados do *cache* e requisições que exigem processamento completo. No entanto, esse não é o único fator que pode influenciar o resultado. A maior taxa de recuperação também está associada à capacidade preditiva do modelo de 3 bilhões de parâmetros, que tende a produzir distribuições de probabilidade mais úteis para ranquear candidatos ao próximo *token*. Assim, o ataque se beneficia tanto de uma separação temporal mais clara entre *cache hits* e *cache misses* quanto de uma melhor priorização dos *tokens* testados. Aspectos como implementação do *cache*, política de escalonamento, gerenciamento de memória, tamanho de lote e variações no tempo de *prefill* afetam diretamente o *TTFT* observado pelo atacante. Portanto, embora os resultados indiquem maior efetividade do ataque no modelo de 3 bilhões de parâmetros, essa diferença não deve ser atribuída exclusivamente ao tamanho do modelo. Para isolar esse efeito, seriam necessários experimentos adicionais controlando separadamente o modelo utilizado, o motor de inferência e os parâmetros de execução.

Com a mitigação ativada no modelo de 3 bilhões de parâmetros, a taxa de recuperação do ataque foi reduzida de 78,0% para 0,0%. Esse resultado indica que a política de equalização de latência removeu o sinal temporal utilizado pelo atacante para classificar uma requisição candidata como *cache hit*. Mesmo nos casos em que a requisição candidata correspondia ao próximo *token* correto e produzia reaproveitamento interno do *cache*, a diferença de tempo em relação à requisição de controle deixou de ser observável de forma confiável. Assim, o atacante passou a classificar esses casos como erro, impedindo a recuperação dos *tokens*-alvo. Esse comportamento também explica os valores observados para o número de consultas e tentativas no cenário com mitigação. Mesmo sem recuperar *tokens*, o ataque continua executando consultas candidatas para cada posição do *prompt*, respeitando o limite máximo de tentativas definido no experimento. No cenário avaliado, foram consideradas até 80 tentativas por *token*, repetidas 30 vezes para reduzir o efeito de ruídos nas medições de latência. Dessa forma, o número de consultas por *token* avaliado é dado por $80 \times 30 = 2400$. Esse valor representa o esforço realizado pelo atacante para cada *token*-alvo, mesmo quando nenhuma tentativa resulta em recuperação bem-sucedida.

Por outro lado, o valor de tentativas médias por *token* recuperado torna-se igual a 0,0, pois nenhum *token* foi efetivamente recuperado após a aplicação da mitigação. Em outras palavras, a métrica não indica ausência de esforço do atacante, mas sim ausência de sucessos a partir dos quais calcular a quantidade média de tentativas até a recuperação. A política de equalização de latência faz com que, mesmo quando uma requisição candidata

corresponde ao *token* correto e produz reaproveitamento interno do *cache*, a diferença temporal em relação à requisição de controle não seja suficiente para classificá-la como *cache hit*. Esse resultado também evidencia uma característica importante da proposta: a mitigação não desabilita o uso do *cache*. O reaproveitamento interno continua ocorrendo e preserva os ganhos computacionais do servidor, mas a vantagem temporal associada ao *cache hit* é mascarada antes que a resposta seja observada pelo cliente. Dessa forma, o método atua sobre o canal lateral explorado pelo ataque, e não sobre o mecanismo em si.

Esse aspecto é relevante em ambientes reais de inferência compartilhada, em que múltiplos usuários utilizam ao mesmo tempo o mesmo servidor de *serving*. Para reduzir custos computacionais e melhorar o desempenho das respostas, o compartilhamento de *KV cache* é empregado em plataformas comerciais de acesso público, agentes conversacionais e aplicações corporativas que são integradas a LLMs. No entanto, esse mesmo mecanismo amplia a superfície de ataque ao permitir que informações relacionadas ao estado interno do *cache* sejam inferidas por clientes externos. Nesse contexto, a proposta apresentada mostra que é possível reduzir a exposição ao canal lateral temporal sem desabilitar os mecanismos de otimização responsáveis pelos ganhos de desempenho do servidor. Dessa forma, o método busca equilibrar requisitos de segurança e eficiência computacional, preservando os benefícios do reaproveitamento de *cache* enquanto reduz a capacidade de exploração do canal temporal por usuários mal-intencionados.

Apesar dos resultados obtidos demonstrarem a efetividade da proposta no cenário avaliado, a avaliação ainda apresenta algumas limitações. Os experimentos foram conduzidos com dois modelos e com um conjunto específico de dados e parâmetros de ataque. Além disso, a mitigação foi avaliada de forma mais detalhada no modelo de 3 bilhões de parâmetros por ter apresentado maior taxa de sucesso no cenário sem defesa. Trabalhos futuros devem avaliar a proposta em outros modelos, diferentes tamanhos de *prompt*, cargas concorrentes no servidor, diferentes valores de intervalo de quantização e cenários com maior variabilidade de latência. Também é necessário medir de forma mais detalhada o impacto da mitigação sobre a latência percebida pelo usuário, uma vez que a segurança adicional é obtida por meio da introdução de atrasos sintéticos nas respostas.

5. Conclusão

Este trabalho apresentou um método de mitigação para ataques de canal lateral temporal em sistemas de *serving* de LLMs com compartilhamento de *KV cache*. A proposta reduz a distinção temporal entre *cache hits* e *cache misses* por meio da equalização da latência observada pelo cliente, utilizando atrasos sintéticos sem desabilitar os mecanismos de *cache*. Diferentemente de abordagens da literatura que reduzem o compartilhamento da *cache* ou introduzem custos adicionais de comunicação e processamento, o método proposto atua diretamente sobre o sinal temporal explorado pelo atacante. Os resultados experimentais mostraram que, sem mitigação, o ataque de *prompt stealing* recuperou até 78,0% dos *tokens* avaliados. Com a mitigação ativada, a taxa de recuperação foi reduzida para 0,0%, eliminando a diferença temporal explorável entre *cache hits* e *cache misses* no cenário avaliado. Como trabalhos futuros, pretende-se avaliar a proposta em diferentes modelos, cargas concorrentes e políticas de quantização, além de investigar com maior detalhe o impacto da mitigação sobre a latência percebida.

Referências

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, Ú., Oprea, A., and Raffel, C. (2021). Extracting Training Data from Large Language Models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association.

Chu, K., Lin, Z., Xiang, D., Shen, Z., Su, J., Chu, C., Yang, Y., Zhang, W., Wu, W., and Zhang, W. (2025). Selective KV-Cache Sharing to Mitigate Timing Side-Channels in LLM Inference. *arXiv e-prints*, page arXiv:2508.08438.

Chua, G. (2024). system-prompt-leakage: Synthetic system prompt dataset for llm prompt leakage research. `https://huggingface.co/datasets/gabrielchua/system-prompt-leakage`. Acessado: 22-05-2026.

Das, B. C., Amini, M. H., and Wu, Y. (2025). Security and Privacy Challenges of Large Language models: A Survey. *ACM Computing Surveys*, 57:1–39.

Ding, R., Xu, T., Shen, X., Ding, A. A., and Fei, Y. (2025). Moecho: Exploiting Side-channel Attacks to Compromise User Privacy in Mixture-of-experts LLMs. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, page 2159–2173. Association for Computing Machinery.

Hagos, D. H., Battle, R., and Rawat, D. B. (2024). Recent Advances in Generative AI and Large Language Models: Current Status, Challenges, and Perspectives. *IEEE Transactions on Artificial Intelligence*, pages 5873–5893.

Kibriya, H., Khan, W. Z., Siddiqa, A., and Khan, M. K. (2024). Privacy issues in Large Language Models: A Survey. *Computers and Electrical Engineering*, page 109698.

McDonald, G. and Bar Or, J. (2025). Whisper Leak: A Side-Channel Attack on Large Language Models. *arXiv e-prints*, page arXiv:2511.03675.

Soleimani, M., Jia, G., Gim, I., seob Lee, S., and Khandelwal, A. (2025). Wiretapping LLMs: Network side-channel attacks on interactive LLM services. Cryptology ePrint Archive, Paper 2025/167.

Song, L., Pang, Z., Wang, W., Wang, Z., Wang, X., Chen, H., Song, W., Jin, Y., Meng, D., and Hou, R. (2025). The Early Bird Catches the Leak: Unveiling Timing Side Channels in LLM Serving Systems. *IEEE Transactions on Information Forensics and Security*, pages 11431–11446.

Wu, G., Zhang, Z., Zhang, Y., Wang, W., Niu, J., Wu, Y., and Zhang, Y. (2025). I Know What You Asked: Prompt Leakage via KV-Cache Sharing in Multi-tenant LLM Serving. In *Network and Distributed System Security (NDSS) Symposium*. Internet Society.

Yan, B., Li, K., Xu, M., Dong, Y., Zhang, Y., Ren, Z., and Cheng, X. (2025). On Protecting the Data Privacy of Large Language Models (llms) and LLM Agents: A Literature Review. *High-Confidence Computing*, page 100300.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., and Zhang, Y. (2024). A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing*, page 100211.

Zhang, J., Bu, H., Wen, H., Liu, Y., Fei, H., Xi, R., Li, L., Yang, Y., Zhu, H., and Meng, D. (2025). When LLMs Meet Cybersecurity: A Systematic Literature Review. *Cybersecurity*, page 55.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Dong, Z., Hou, Y., Zhang, B., Min, Y., Zhang, J., Liu, P., Wang, X., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Hu, Y., Nie, J.-Y., and Wen, J.-R. (2026). A survey of large language models. *Frontiers of Computer Science*, page 2012627.