Fine-Tuning Eficiente de Modelos de Linguagem Para Detectar Anomalias em *Logs* Privados usando Aprendizado Federado

Gabriel U. Talasso[1], **Allan M. de Souza**[1], **Daniel Guidoni**[2]
Eduardo Cerqueira[3], **Leandro A. Villas**[1]

[1]Universidade Estadual de Campinas (UNICAMP), Brasil,
[2]Universidade Federal de Ouro Preto (UFOP), Brasil
[3]Universidade Federal do Pará (UFPA), Brasil

g235078@dac.unicamp.br, cerqueira@ufpa.br, guidoni@ufop.edu.br
{allanms, lvillas}@unicamp.br

Resumo. *O crescimento acelerado de sistemas distribuídos em redes de computadores tem ampliado as preocupações com vulnerabilidades, falhas e ataques maliciosos, tornando a detecção de anomalias uma tarefa crucial para garantir a confiabilidade e segurança desses sistemas. A análise de logs de sistema desponta como uma abordagem promissora para identificar comportamentos anômalos, mas enfrenta desafios significativos, como a falta de flexibilidade, eficiência computacional e adaptabilidade para cenários distribuídos e restritos em recursos, além de questões relacionadas à privacidade dos dados. Este trabalho investiga o uso de modelos de linguagem aliados a técnicas de eficiência de treinamento e comunicação no contexto do aprendizado federado, com o objetivo de aprimorar a detecção de anomalias em cenários desafiadores. A abordagem proposta viabiliza o treinamento colaborativo e privado entre múltiplos clientes, preservando a privacidade dos dados enquanto otimiza a eficiência em ambientes de recursos limitados. Resultados demonstram o desenvolvimento bem-sucedido de um fluxo de trabalho para ajuste de modelos de linguagem na detecção de anomalias, com análises detalhadas de desempenho atingindo performance superior a 98% de F1 além de uma redução de até 4000x no tamanho das mensagens transmitidas delineando assim diretrizes promissoras para futuros avanços na área.*

Abstract. *The rapid growth of distributed systems in computer networks has heightened concerns regarding vulnerabilities, failures, and malicious attacks, making anomaly detection a crucial task to ensure the reliability and security of these systems. System log analysis emerges as a promising approach to identify anomalous behaviors, but it faces significant challenges, such as lack of flexibility, computational efficiency, and adaptability for distributed scenarios with resource constraints, as well as issues related to data privacy. This work investigates the use of language models combined with training and communication efficiency techniques within the context of federated learning, aiming to enhance anomaly detection in challenging scenarios. The proposed approach enables collaborative and private training across multiple clients, preserving data privacy while optimizing efficiency in resource-constrained environments. Results demonstrate the successful development of a workflow for fine-tuning language models in anomaly detection, with detailed performance analysis achieving an F1 score greater than 98%, along with up to a 4000x reduction in the size of transmitted messages, outlining promising guidelines for future advancements in the field.*

1. Introdução

Sistemas distribuídos desempenham um papel essencial e em constante expansão em diversas aplicações contemporâneas, abrangendo redes veiculares, redes de *smartphones* ou até

mesmo em dispositivos de Internet das Coisas (IoT). No entanto, o uso crescente desses sistemas também aumenta sua vulnerabilidade a falhas ou ataques maliciosos, comumente denominados anomalias, que podem comprometer o desempenho das aplicações, provocar violações de segurança e privacidade, e ocasionar prejuízos financeiros significativos aos sistemas e seus operadores [Pang et al. 2021]. Nesse cenário, métodos para detecção de anomalias tornaram-se indispensáveis, pois garantem não apenas a confiabilidade e integridade, mas também o correto e eficaz funcionamento desses sistemas [Li et al. 2022, Pang et al. 2021].

Uma abordagem promissora no contexto de sistemas de computação para detecção de anomalias se baseia na análise de *logs* do sistema. Esses registros contém eventos críticos e informações de execução e dos processos correntes nas máquinas, sendo ferramentas valiosas para a avaliação de desempenho e segurança dos processos envolvidos. Contudo, a análise manual desses dados é inviável dada a enorme quantidade de *logs* gerados e a complexidade intrínseca de identificar padrões de comportamento anômalos [Xu et al. 2009].

Métodos e ferramentas baseados em aprendizado de máquina têm sido aplicados com sucesso, mas enfrentam desafios significativos em capturar informações temporais, contextuais e semânticas nos dados de *log* que possuem uma natureza predominantemente textual. Além disso, essas abordagens frequentemente carecem de flexibilidade e adaptabilidade para atender às aplicações necessárias [Guo et al. 2021, Almodovar et al. 2024]. Outro obstáculo importante ocorre em cenários reais onde regulações e regras de *compliance* restringem o acesso aos dados privados, inviabilizando treinamentos em larga escala e reduzindo a eficiência das soluções desenvolvidas [Li et al. 2022].

Modelos de Linguagem de Grande Escala (*Large Language Modelos - LLMs*), devido à sua flexibilidade para lidar com diferentes tipos de textos e à capacidade de interpretar dados sequenciais, trouxeram avanços significativos na compreensão e análise contextual, superando limitações de métodos tradicionais em muitas aplicações. Na detecção de anomalias em *logs*, por exemplo, esses modelos têm apresentado desempenho expressivo [Guan et al. 2024]. No entanto, a implementação e o uso de LLMs para a detecção de anomalias em sistemas distribuídos enfrenta desafios não triviais de serem superados, incluindo o elevado custo computacional associado ao treinamento de modelos que podem possuir bilhões de parâmetros, limitações de recursos em dispositivos IoT ou *smartphones*, e problemas relacionados à privacidade e segurança no compartilhamento de *logs* sensíveis [Li et al. 2022].

Diante da carência de soluções na literatura que sejam ao mesmo tempo flexíveis, robustas e adaptáveis a ambientes distribuídos com limitações de recursos, incluindo processamento, comunicação e privacidade dos dados, como por exemplo cenários veiculares e de mobilidade urbana, de *smartphones* ou de dispositivos e sensores IoT, este artigo apresenta uma abordagem inovadora e eficiente para a detecção de anomalias em *logs*[1]. A solução é baseada no Fine-Tuning de Modelos de Linguagem por meio de Aprendizado Federado (Federated Learning - FL). Além disso, apresenta-se uma análise do desempenho e dos custos associados ao uso da proposta em sistemas distribuídos.

A abordagem proposta utiliza modelos pré-treinados em grandes volumes de dados, porém de pequeno porte em comparação aos LLMs, denominados Modelos de Linguagem de Pequena Escala (*Small Language Modelos - SLMs*) [Wang et al. 2024]. Para otimizar recursos como memória, processamento e comunicação durante o treinamento e aprimoramento dos modelos, aplicamos técnicas de ajuste eficiente de parâmetros [Hu et al. 2021].

[1]Código disponível em: `https://github.com/GabrielTalasso/FL-LLM-AD`

Além disso, por se tratar de um ambiente distribuído e com dados privados, utilizamos o FL, tornando possível o treinamento colaborativo entre diferentes participantes da rede, conhecidos como clientes, sem a necessidade de compartilhar dados sensíveis com terceiros [McMahan et al. 2017]. As principais contribuições deste artigo são apresentadas a seguir:

- Proposta de um fluxo eficiente para ajuste de modelos de linguagem com baixo *overhead*, voltado à detecção de anomalias em *logs* de sistemas distribuídos.
- Aplicação do FL para possibilitar o treinamento colaborativo entre múltiplos clientes, reduzindo riscos de privacidade e otimizando a eficiência em comunicação.
- Análise extensiva de desempenho, custo e eficiência da abordagem proposta, explorando variações na qualidade e tamanho dos modelos, identificando caminhos promissores e direções para avanços futuros.

O restante do artigo está organizado da seguinte maneira. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 introduz a metodologia e conceitos chave. A Seção 4 apresenta a análise de desempenho, de custo e os respectivos resultados. A Seção 5 conclui o artigo a apresenta os trabalhos futuros.

2. Trabalhos Relacionados

A literatura apresenta diversas soluções baseadas em LLM para a detecção de anomalias em *logs* de sistemas. Contudo, muitas delas apresentam limitações significativas em termos de eficiência, devido ao uso de modelos excessivamente grandes e custosos para treinamento. Além disso, essas abordagens frequentemente não são aplicáveis em cenários distribuídos, pois desconsideram restrições relacionadas a recursos computacionais, custos de comunicação e privacidade dos dados. Assim, nesta seção, são apresentados métodos relevantes descritos na literatura, juntamente com uma análise das suas limitações no que tange à eficiência e à aplicabilidade.

O método LogBERT [Guo et al. 2021] utiliza a arquitetura do modelo BERT [Devlin 2018] para a detecção de anomalias. A abordagem nesse caso se baseia em criar *templates* das mensagens de *logs* e codificar cada uma deles em um *token* específico utilizando-os em sequência como entrada no modelo. Em seguida, é feito o treinamento da arquitetura do BERT do zero, ou seja, sem utilizar pesos pré-treinados, com a finalidade de capturar o padrão das sequências dessas mensagens. Nesse caso, além de obter um custo alto por necessitar treinar o modelo inteiro, essa solução utiliza um modelo clássico de linguagem que é menos flexível e pode ter desempenho limitado quando comparado com os modelos generativos modernos do tipo SLMs e LLMs. Por fim, esse método é pouco flexível uma vez que toda mensagem deve ser transformada em um *template* específico e a arquitetura inteira precisaria ser alterada no caso da inclusão de novos tipos de mensagens.

Já o LogFiT [Almodovar et al. 2024] utiliza uma arquitetura similar ao BERT, nomeado de RoBERTa [Liu 2019] e faz um ajuste dos pesos desse modelo pré treinado para aprender o comportamento dos *logs* normais. Assim como o LogBERT, LogFiT não necessita de rótulos para treinar e tem uma abordagem de detecção baseada no erro do modelo que assume-se já ter aprendido o comportamento padrão dos dados, dessa forma erros muito grandes na predição acusam que a amostra pode ser anômala. LogFiT tem a vantagem de utilizar o modelo pré treinado que aumenta a flexibilidade para adaptação em vários contextos e reduz custos de treinamento, no entanto, ainda necessita do ajuste de todos pesos do modelo o que torna o custo do ajuste bastante elevado, impossibilitando aplicações em cenários de dispositivos restritos.

A solução LogLLM [Guan et al. 2024] trata o problema de detecção de anomalias como um problema de classificação supervisionada, ou seja utilizam amostras rotuladas de dados normais e anômalos. Além disso, LogLLM utiliza o modelo BERT para criação de representações semânticas dos *logs* e um modelo Llama que, em linguagem natural é instruído a analisar a sequencia e responder se ela é ou não anômala. A união desses dois modelos com ajuste de treinamento faz com ele ele se torne capaz de analisar *logs* e classificá-los. Uma vantagem é que, ao contrario dos métodos anteriores, LogLLM não utiliza *parsers* nem *templates*, aumentando assim sua flexibilidade a diferentes contextos e diferentes tipos de *logs*. Contudo, esse método, além de necessitar da rotulação prévia dos dados normais e anômalos, apresenta limitações com respeito ao alto custo de treinamento desses modelos grandes que precisam ser ajustados para desempenhar a tarefa de detecção.

O LogGPT [Qi et al. 2023] também utiliza uma abordagem supervisionada, enviando os *logs* diretamente para o modelo GPT através de uma API, sem a necessidade de ajustes do modelo, utilizando apenas a inferência. O LogGPT passa, através de um *prompt*, um conjunto de exemplos de *logs* normais e anômalos com seu respectivo rótulo feito por um humano e em seguida uma série de instruções sobre como o modelo deve se comportar e sua tarefa de classificação de uma nova sequência. Apesar de ter vantagens quanto ao baixo custo computacional local uma vez que utiliza o modelo através de uma API sem treinamento adicional, o LogGPT sofre em performance, reportando valores de precisão e sensitividade baixos. Além disso, essa abordagem possui problemas ligados a privacidade no compartilhamento de *logs* no caso de aplicações em que esses dados não podem ser enviados a terceiros.

Por fim, ao contrário das abordagens anteriores que apenas consideram um cenário centralizado e sem limitações de recursos, o FedLog [Li et al. 2022] desenvolve um modelo próprio utilizando redes convolucionais temporais e técnicas de mascaramento em vez de arquiteturas *transformers* e modelagem por linguagem. Essa abordagem se baseia no uso de FL para criar um modelo colaborativamente de forma distribuída e privada, utilizando conjuntamente técnicas de poda de modelo para reduzir os custos de comunicação associados. Porém, além de necessitar do uso de *templates* e *parsers*, o que limita a flexibilidade e reutilização dos modelos e a adição de novos tipos de mensagens, o FedLog ainda propõe o treinamento do modelo completo em todos dispositivos participantes, o que é inviável em cenários limitados de recursos computacionais.

Um resumo de cada método e suas vantagens e desvantagens pode ser encontrado na Tabela 1. Nesse caso as colunas representam (i) Federado: se o método considera o cenário federado ou não, ou seja, considera dados privados e distribuídos entre diversos dispositivos (ii) Não-Superv.: se o método é não-supervisionado, ou seja, não necessita de rótulos atribuídos por um humano nas amostras anômalas para treinamento (iii) Sem Parser: se o método não utiliza *parsers* ou *templates* que limitam seu funcionamento em diferentes aplicações e que podem ocorrer em erros quando deparados com *logs* ainda não vistos durante o treinamento, (iv) Eficiente em Param.: se o método utiliza de alguma técnica ou abordagem eficiente em parâmetro para ajuste e não necessita do treinamento do modelo completo ou da comunicação de todos os parâmetros, o que implica em altos custos computacionais e (v) Pré-treinado: se a bordagem utiliza de modelos pré-treinados que podem ser úteis para encurtar o treinamento e reaproveitar o conhecimento anterior do modelo.

Em vista com as limitações e lacunas na literatura com relação a eficiência, desempenho e aplicabilidade dos modelos, neste trabalho propomos um processo de treinamento que lida com a eficiência computacional e de comunicação enquanto se preocupa com a flexibilidade,

Tabela 1. Comparação das características de diferentes métodos de detecção de anomalia em logs relatados na literatura e da nossa abordagem proposta.

Método	Federado	Não-Superv.	Sem Parser	Eficiente em Param.	Pré-Treinado
LogBERT		✓			
LogGPT				✓	✓
LogLLM			✓		✓
LogFiT		✓	✓		✓
FedLog	✓	✓		✓	
Nossa Proposta	✓	✓	✓	✓	✓

performance e adaptabilidade dos modelos para cenários distribuídos e com limitações de recursos e dados privados.

3. Metodologia

Esta seção apresenta os conceitos e a metodologia deste artigo para o ajuste fino (*Fine-Tuning* - FT) de modelos de linguagem, com o objetivo de detectar anomalias em *logs* de sistemas. Além de reduzir os custos computacionais e de comunicação, o processo proposto permite o treinamento colaborativo de modelos por meio do FL, aumentando a segurança dos dados enquanto aprimora o desempenho dos modelos.

3.1. Visão Geral

Uma visão geral da proposta está apresentada na Figura 1, que destaca o passo a passo do funcionamento da metodologia. Nesse contexto, os dispositivos clientes podem variar conforme a aplicação em que a detecção de anomalias será realizada. Exemplos incluem dispositivos de gerenciamento de redes na borda, *smartphones* de usuários, dispositivos e sensores em redes IoT e até mesmo instituições inteiras, com *logs* abrangendo vários sistemas de diversos tipos.

A primeira etapa do processo (***1***) consiste no pré-processamento local dos *logs*. Nesta etapa, são definidas variáveis como a janela de análise, a formatação da saída e procedimentos para remover informações sensíveis, como endereços IP, utilizando expressões regulares. Esse pré-processamento é padronizado entre todos os clientes. Na etapa seguinte (***2***), cada cliente recebe o modelo de linguagem pré-treinado em sua totalidade, ou seja, com todos os parâmetros. Esse envio ocorre apenas uma vez, no início da participação dos clientes, e os pesos do modelo são mantidos localmente para execução de treinamentos e inferências

Na etapa (***3***), cada cliente é responsável por ajustar, de forma paralela, o modelo recebido utilizando seus dados locais pré-processados. Essa é uma das etapas mais custosas do processo, razão pela qual aplicam-se técnicas de ajuste eficiente de parâmetros (*Parameter-Efficient Fine-Tuning - PEFT*), que permitem modificar uma quantidade reduzida de pesos do modelo. Este processo diminui os requisitos de computação, memória e processamento durante o treinamento. Em seguida, na etapa (***4***), os clientes enviam os parâmetros atualizados, que são significativamente menos numerosos em comparação aos parâmetros originais, para o servidor. O servidor, então, na etapa (***5***), agrega os pesos recebidos em um único modelo, chamado de modelo global, que incorpora o conhecimento adquirido localmente por cada cliente participante.

Por fim, na etapa (***6***), os parâmetros agregados são retornados aos clientes, que os utilizam para realizar inferências de forma privada e paralela em seus próprios dados locais. Na etapa (***7***), o modelo local de cada cliente analisa os resultados obtidos e, com base em um limiar de performance das predições, identifica *logs* possivelmente anômalos. O processo de (***3***) até (***6***) é repedido por diversas rodadas de comunicação, até se atingir estabilidade ou um valor pré-definido na performance de predições.

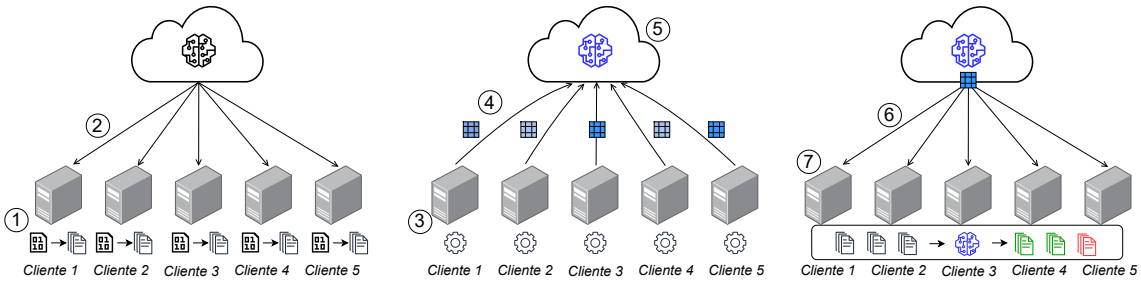

Figura 1. Ilustração do passo a passo de funcionamento da metodologia de ajuste de modelos de linguagem para detecção de anomalias

A seguir detalharemos cada uma dessas etapas e seu respectivo funcionamento e importância na detecção das anomalias.

3.2. Pré-processamento de Logs

A primeira etapa é o pre-processamento dos *logs* locais dos clientes. Nessa etapa, todos os clientes aplicam as mesmas regras de processamento ao conjunto completo de seus dados. Esse procedimento é importante para separar corretamente diferentes mensagens e remover informações que poderiam prejudicar o treinamento, como IDs e horários, ou representar riscos à privacidade dos usuários, como endereços IP. Apesar disso, não há padronização nos tipos de mensagens permitidas (e.g., por *templates* ou *parsers*), o que proporciona maior flexibilidade ao modelo para lidar com diferentes formatos de *logs*.

O primeiro passo do pré-processamento é a definição da janela que será utilizada durante o treinamento para ajuste. Nessa etapa, a abordagem varia conforme a aplicação e a necessidade, podendo incluir: (***i***) divisão por seção, em cenários onde os *logs* já estão naturalmente segmentados por processos ou tarefas específicas executadas no sistema; (***ii***) uso de uma janela fixa baseada na quantidade de mensagens recebidas; ou (***iii***) uso de uma janela fixa baseada no intervalo de tempo das mensagens [Guan et al. 2024]. A escolha do tipo de janela depende diretamente da forma como os *logs* são coletados.

A segunda etapa do processo consiste na separação das mensagens e na remoção de informações sensíveis e capazes de confundir o modelo. Para isso, utiliza-se uma sequência especial, "*;-;*", para delimitar cada mensagem dos *logs*. Essa abordagem auxilia o modelo a identificar com maior clareza o início e o fim de mensagens e alertas presentes nos *logs*. Em seguida, empregam-se expressões regulares para substituir automaticamente todos os números, que podem representar datas, horários, endereços IP ou IDs, por uma sequência especial, "<*>". Essa substituição permite que o modelo reconheça a presença de valores numéricos nessas posições sem gerar confusões durante a predição, uma vez que o modelo não será treinado e penalizado na predição dos números. Adicionalmente, esse processo contribui na privacidade dos dados uma vez que remove informações sensíveis como os IPs. Um exemplo detalhado desse processo de transformação é apresentado na Figura 2.

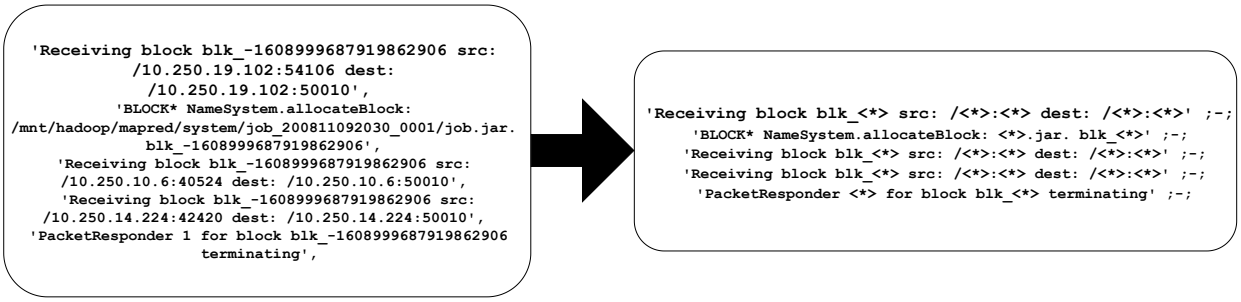

Figura 2. Exemplo do resultado do pré-processamento de um conjunto de *logs*. Nesse caso foi usado o *dataset* HDSF com um janelamento por seção.

Podemos ressaltar que esse método auxilia no treinamento, ao mesmo tempo em que mantém a flexibilidade ao contrário dos métodos de *parsing* e *templates*. Assim, qualquer palavra pode ser incluída e não existe limitações na quantidade dos tipos de mensagens diferentes que podem ser analisadas [Almodovar et al. 2024]. Além disso, é importante ressaltar que esse mesmo processo deve ser aplicado tanto no conjunto de dados de treino como também no momento em que o modelo for realizar a inferência em novos dados coletados para a efetiva detecção de anomalias.

3.3. Aprendizado Federado

Visando cenários em que os recursos computacionais e os dados estão distribuídos entre diversos clientes, nossa proposta adota o uso do FL. O FL possibilita o treinamento distribuído e paralelo de modelos de aprendizado de máquina de forma colaborativa, aumentando a quantidade de dados utilizados durante o treinamento e, consequentemente, melhorando o desempenho dos modelos nas tarefas. Além disso, o FL preserva os dados localmente e compartilha apenas os modelos, elevando os níveis de privacidade, uma vez que nenhuma informação sensível é diretamente transmitida. Essa abordagem é particularmente vantajosa no contexto de detecção de anomalias, pois permite o uso de *logs* de sistemas privados para o desenvolvimento de aplicações de segurança relevantes sem comprometer a privacidade dos dados e por é utilizada nessa nessa proposta em conjunto com as demais técnicas.

Na forma padrão do FL, conhecida como FedAvg [McMahan et al. 2017], um subconjunto aleatório de clientes é selecionado a cada rodada para realizar o treinamento local dos modelos utilizando seus dados privados. Após o treinamento, os clientes enviam os modelos atualizados ao servidor central, que os agrega por meio de uma média ponderada dos parâmetros, considerando o tamanho do conjunto de dados local de cada cliente. O servidor então retorna o modelo global atualizado a todos os clientes, concluindo um ciclo denominado rodada. Esse processo é repetido até que o modelo global convirja, ou seja, alcance um nível de desempenho satisfatório ou pare de melhorar. Dessa forma, os clientes obtêm um modelo colaborativamente treinado, preservando a privacidade de seus dados em todo o processo.

Matematicamente, o FedAvg pode ser formalizado da seguinte maneira. Considerando um cenário com C clientes e T rodadas de comunicação entre cliente e servidor. A cada rodada $t \in \{1, 2, \ldots, T\}$, um subconjunto c de clientes, onde $|c| \leq |C|$, é selecionado aleatoriamente para participar do treinamento. Cada cliente $i \in C$ possui seu conjunto de dados locais D_i, com $n_i = |D_i|$ exemplos, e o conjunto total de dados é $n = |\bigcup_{i \in C} D_i|$. Nesse caso, D_i é um conjunto de amostras normais pré-processadas de *logs* coletados no sistema de cada um dos clientes.

Inicialmente, quando $t = 0$, cada cliente recebe o modelo completo (pre-treinado) W_0. Contudo, como utilizamos técnicas de PEFT, cada cliente é responsável por ajustar um sub-conjunto menor de parâmetros w_t (para um treinamento realizado na rodada t). Após o treinamento, o cliente envia os novos parâmetros do modelo, w^i_{t+1}, para o servidor. O servidor, por sua vez, agrega os modelos recebidos da seguinte maneira:

$$w_{t+1} = \sum_{i \in C} \frac{n_i}{n} w^i_t, \tag{1}$$

Onde w_{t+1} representa o modelo global atualizado e w^i_{t+1} é o modelo treinado recebido do cliente i. O objetivo principal do FedAvg é minimizar a seguinte função:

$$\min f(w) \text{ onde } f(w) = \sum_{k=1}^{C} \frac{n_i}{n} \mathcal{L}(x_i, y_i; w), \tag{2}$$

Nesta equação, w é o modelo global, $\mathcal{L}$ representa a sua função de perda, x_i são os dados de entrada e y_i os rótulos associados. Como a abordagem utilizada é não supervisionada, o conjunto de entradas x_i é tratado como os rótulos y_i, e a perda é calculada com base na capacidade do modelo de prever corretamente as palavras em uma sentença.

Dessa forma é possível um treinamento colaborativo de distribuído e com maior privacidade, contudo o FL enfrenta alguns desafios significativos, sendo um dos principais o alto custo de comunicação, decorrente da necessidade de transmitir modelos completos entre os clientes e o servidor central em cada rodada de treinamento [Souza et al. 2023, Cho et al. 2020]. Como mencionado, para mitigar essa limitação nossa abordagem busca implementar um treinamento mais eficiente, utilizando técnicas de PEFT, o qual os detalhes são apresentados na seção seguinte.

3.4. Fine-Tuning Eficiente

Um dos principais desafios no treinamento distribuído de modelos de linguagem está relacionado aos altos custos envolvidos. Além do custo computacional associado às atualizações dos pesos, no contexto de FL, é necessário transmitir essas atualizações para o servidor central, o que aumenta significativamente o custo de comunicação e pode gerar problemas de rede.

Para mitigar o desafio acima, emprega-se uma técnica de *fine-tuning* eficiente em parâmetros (PEFT), a Low-Rank Adaptation (LoRA) [Hu et al. 2021], considerada o estado da arte para adaptar LLMs de forma eficiente. O LoRA modifica apenas um número reduzido de parâmetros (em comparação com o modelo completo), mantendo os pesos do modelo pré-treinado congelados, o que resulta em uma significativa redução do custo computacional.

Essa técnica insere matrizes de baixo posto (*ranks*) em determinadas camadas do modelo *transformers*, permitindo uma adaptação para execução de outras tarefas sem a necessidade de atualizar os pesos do modelo pre-treinado (W_0). Dessa forma, mantém-se o conhecimento original do modelo enquanto se ajusta seu comportamento para as necessidades específicas em treinamento. Ou seja, é possível aproveitar o conhecimento prévio do modelo em linguagem natural enquanto o ajusta para entender o comportamento normal de *logs*.

Essas matrizes são chamadas de adaptadores e, no contexto federado, apenas elas precisam ser comunicadas ao longo das rodadas, conforme ilustrado na Figura 1. A Equação 3 descreve como os parâmetros treinados com LoRA são combinados aos pesos congelados para adaptar o modelo:

$$W = W_0 + w = W_0 + AB \tag{3}$$

Onde $W_0 \in \mathbb{R}^{d\times k}$ representa a matriz de pesos original (congelada) de uma camada do LLM. O termo w representa o ajuste de pesos durante o *fine-tuning* para entendimento dos *logs*. As matrizes $A \in \mathbb{R}^{d\times r}$ e $B \in \mathbb{R}^{r\times k}$ são as matrizes de baixo rank, com r sendo o rank, onde $r \ll d$ e $r \ll k$. Além disso, como essas matrizes são incorporadas somando-as diretamente aos pesos do modelo durante o *fine-tuning*, formando o modelo atualizado $W \in \mathbb{R}^{d\times k}$, elas não introduzem latência adicional durante a inferência [Hu et al. 2021].

Assim, o LoRA é utilizado para ajustar o comportamento modelo, garantindo que ele represente corretamente os *logs* normais gerados localmente. Após a atualização desses adaptadores, apenas eles precisam ser transmitidos ao servidor, o que reduz significativamente a quantidade de dados enviados. Além disso, como há menos parâmetros a serem otimizados durante o treinamento, o custo computacional também é consideravelmente reduzido.

Dessa forma, com o comportamento padrão dos dados representados através das matrizes LoRA é necessário a realização do processo de tomada de decisão para a detecção, categorizando *logs* como anômalos ou normais de maneira eficiente e precisa.

3.5. Tomada de Decisão

Por fim, de forma paralela ao treinamento são necessárias regras de tomada de decisão para acusar ou rejeitar uma anomalia em novos dados. Com o modelo tendo aprendido a representar e gerar dados de *log* comuns entre os clientes, a abordagem que escolhemos para detectar anomalias se baseia em observar o aumento dos erros nas predições do modelo. Esse processo se alinha com outras abordagens da literatura [Almodovar et al. 2024, Li et al. 2022].

A ideia é utilizar a saída do modelo de linguagem para determinar as próximas palavras mais prováveis ao longo de uma sentença e, em seguida, com um hiperparâmetro K, medir a acurácia das top-K palavras mais prováveis. Ou seja, se a próxima palavra aparece entre as top-K preditas pelo modelo, é considerado um acerto. Com essas acurácias é possível definir um limiar (β) de "normalidade" e acusar anomalia para acurácias mais baixas que o esperado. A escolha dos hiperparâmetros K e β pode ser feita por uma busca prévia, como analisaremos nas seções seguintes.

4. Análise de Desempenho

Esta seção tem como objetivo avaliar a solução proposta considerando diversos cenários, verificando as vantagens, desvantagens e efeitos dos hiperparâmetros no desempenho do modelo e na eficiência de comunicação. Na Seção 4.1 detalhamos a implementação da solução proposta e o conjunto de dados, modelos e métricas utilizadas. A Seção 4.2 apresenta as medições e experimentos realizados.

4.1. Cenário de Avaliação

Esta seção detalha a implementação da abordagem, abrangendo os aspectos técnicos do treinamento e o ambiente utilizado nos experimentos. Para nossa análise de custo e desempenho, consideramos modelos pequenos de linguagem (SLMs), que possuem capacidade semântica e compreensão de sequências em linguagem natural. Esses modelos, pré-treinados, reduzem os custos associados ao treinamento do zero, sendo adequados para aplicações em borda em sistemas distribuídos. A família de modelos escolhida foi a SmolLMs [Allal et al. 2024], composta por versões com parâmetros variando entre 135 milhões, 360 milhões e 1,7 bilhões.

Os treinamentos utilizaram técnicas de ajuste eficiente com LoRA, variando o parâmetro *rank* entre 2 e 128. Além disso, todos os modelos foram treinados e armazenados utilizando quantização para 4 bits, o que reduziu significativamente a memória necessária durante o treinamento. Os experimentos foram realizados em um cluster de GPUs NVIDIA A100, cada uma com 80 GB de memória.

O conjunto de dados utilizado para os experimentos é o HDSF, que contém tanto *logs* normais quanto anômalos do Hadoop Distributed File System [Xu et al. 2009]. Este conjunto foi transformado em janelas e pré-processado conforme especificado na Seção 3.2, resultando em mais de 400 mil amostras normais para o treinamento. O conjunto de validação foi considerado centralizado no servidor, sendo assim é comum a todos os clientes, contendo 1000 amostras normais e 1000 amostras anômalas para avaliar a capacidade do modelo em distingui-las.

Todos os experimentos descritos a seguir, a menos que especificado de outra forma, foram conduzidos com 50 rodadas de comunicação no aprendizado federado e uma federação de 50 clientes, dos quais 10% são selecionados a cada rodada com uma divisão IID (Independente e Identicamente Distribuída) do *dataset* original. Além disso, para o treinamento dos modelos foi utilizada uma taxa de aprendizado com decaimento cossenoidal [Ye et al. 2024], variando de 10^{-3} até 10^{-5}, com um tamanho de batch de 32 e um total de 10 interações por rodada por cliente.

4.2. Resultados

Inicialmente, para avaliar o treinamento nos modelos de forma federada e sua respectiva melhora, a Figura 3 apresenta, em escala logarítmica, as perdas de treinamento para os modelos de 135M, 360M e 1.7B. Considerando o treinamento sem os adaptadores LoRa, representado por FFT (do inglês, *Full Fine Tuning*), é possível notar uma instabilidade na perda durante o ajuste do modelo que pode ser explicada pela dificuldade de treinar modelos completos e com muitos parâmetros pela instabilidade dos pesos e esquecimento na rede [Kirkpatrick et al. 2017, Hu et al. 2021]. Pode-se também observar Uma diferença visível na velocidade de aprendizado quando aumenta-se o tamanho dos adaptadores, principalmente nos casos de modelos menores. No caso do modelo maior, a variação no tamanho dos *ranks* (R) não impacta significativamente por conta do modelo já ser poderoso o suficiente e necessitar de menos adaptações para representar melhor os dados.

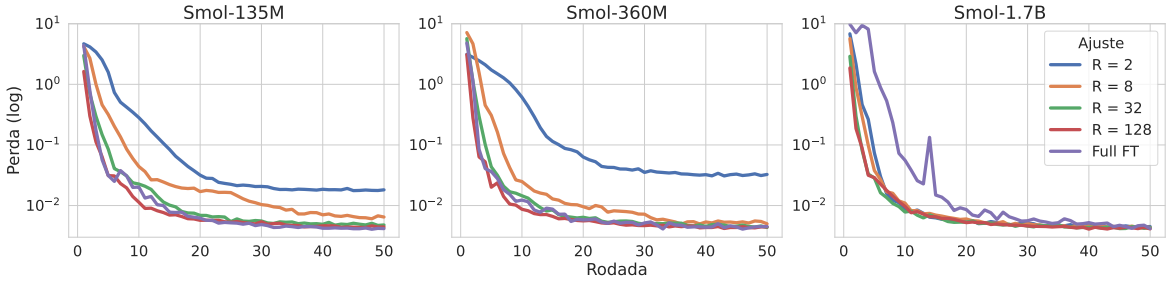

Figura 3. Perdas ao longo do treinamento federado para diferentes tamanhos de modelos e de LoRA *ranks* (R) utilizados.

Além de analisar a perda do modelo de treinamento é necessário verificar a performance na detecção de anomalias no conjunto de validação. Nesse caso utilizamos a partição criada anteriormente e não vista durante o treinamento e avaliamos o modelo global gerado a cada rodada. A métrica utilizada foi F1, que é uma combinação por média harmônica da precisão

e sensibilidade do modelo, representando tanto a capacidade de classificar corretamente um dado quanto a capacidade de detectar mais das anomalias existentes.

Dessa forma, a Figura 4 apresenta o máximo valor de F1 atingido ao longo do treinamento por cada uma das abordagens que consideram o modelo sem treinamento para detecção de anomalias, ou seja, o modelo pré-treinado (PT), o modelo com parâmetros LoRA com *ranks* variando de 2 a 128 e o ajuste do modelo completo (FFT). Além disso, variamos o valor K das top-K predições, relatado na Seção 3.5 sobre a quantidade de palavras preditas para calcular a qualidade do modelo. O limiar β foi escolhido a partir de uma busca exploratória em cada caso e definido como aquele que atinge melhor F1.

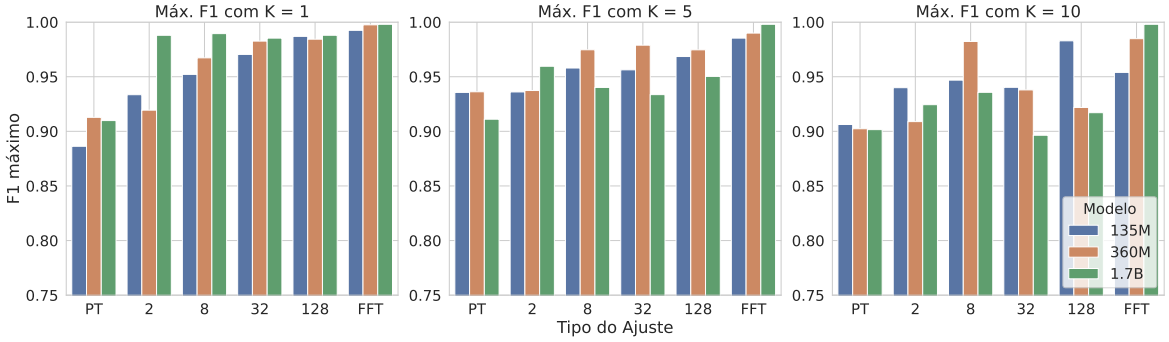

Figura 4. Máximo F1 atingido pelo modelo global ao longo das rodadas por tamanho do modelo e tipo do ajuste (PT: Pré-Treinado, FFT e *ranks* dos adaptadores) variando o hiperparâmetro top-K.

Com esse resultado, destaca-se a importância tanto do número de parâmetros do modelo quanto do número de parâmetros treinados nos adaptadores para a qualidade das detecções. Em muitos casos, a quantidade treinada influencia diretamente uma detecção mais eficaz, mostrando que à medida que o número de parâmetros aumenta, a precisão das detecções melhora. Além disso, o parâmetro K que se demonstrou mais consistente foi com o valor $K = 1$, alcançando melhores performances na maioria dos casos, e resultados mais instáveis em outros valores. Portanto, as próximas análises e comparações serão baseadas nesse valor.

Considerando o cenário proposto de forma distribuída, com restrições de recursos como poder de computação e custos de comunicação, e com base nos resultados de melhora na performance com o aumento do número de parâmetros treinados, analisamos a troca entre eficiência e performance na detecção. A Tabela 2 apresenta a diferença no tamanho das mensagens transmitidas por cliente, por rodada, para diferentes modelos e diferentes tamanhos dos *ranks* dos adaptadores LoRA. Fica evidente que, ao utilizar técnicas de ajuste eficiente (PEFT), a abordagem proposta reduz mais de 4300x a quantidade de dados transmitidos considerando R=2 e o modelo FFT no caso do modelo 1.7B. Além disso, mostramos o número de parâmetros treináveis em cada tipo de ajuste, o que impacta diretamente no número de FLOPs (*Float Point Operations*) necessários para treinar esses modelos, ou seja, diretamente relacionado com custo computacional de treinamento de cada um deles.

Em seguida analisamos, durante a simulação, a relação entre o custo de comunicação de cada um dos tipos de ajustes e a performance atingida pelo modelo, uma vez que maiores modelos também tendem a atingir maiores performances. Essa análise ajuda a entender as vantagens de aumentar alguns custos de comunicação e computação em troca de mais performance nas detecções. Além disso, ajuda a dimensionar necessidades em diferentes cenários e aplicações entendendo até onde vale a pena ampliar o modelo. Por exemplo, em situações

Tabela 2. Comparação do tamanho das mensagens por cliente por rodada e número de parâmetros treináveis para diferentes tipos de ajuste.

Tamanho do modelo transmitido (Mb)				Número de parâmetros treináveis (milhões)			
Modelo	135M	360M	1.7B	Modelo	135M	360M	1.7B
R = 2	0.48	0.84	1.59	R = 2	0.12	0.20	0.39
R = 8	1.86	3.29	6.30	R = 8	0.46	0.82	1.57
R = 32	7.39	13.12	25.18	R = 32	1.84	3.28	6.29
R = 128	29.51	52.45	100.68	R = 128	7.37	13.11	25.17
FFT	538.09	1447.32	6845.53	FFT	134.52	361.82	1711.38

não criticas e com dispositivos móveis ou *IoT* pode ser mais vantajoso possuir modelos que, apesar de conter uma taxa de erros maiores, consomem menos memória e processamento, permitindo que a aplicação seja utilizada em ambientes restritos de recursos. Enquanto em cenários de cooperação de grandes servidores em aplicações que requerem altas performances de detecção podem optar por utilizar modelos maiores mesmo que isso implique em um alto custo.

Para isso, a Figura 5 apresenta o custo computacional total, que considera o número de clientes selecionados por rodada, tamanho dos modelos transmitidos e número de rodadas para atingir performance máxima naquela configuração. Os resultados indicam que, para modelos menores, o tamanho do adaptador é mais relevante do que para modelos maiores trazendo ganhos expressivos com seu aumento. Pode-se observar que a técnica proposta atinge acurácias próximas do modelo completo mesmo comunicando de 10 a 100 vezes menos.

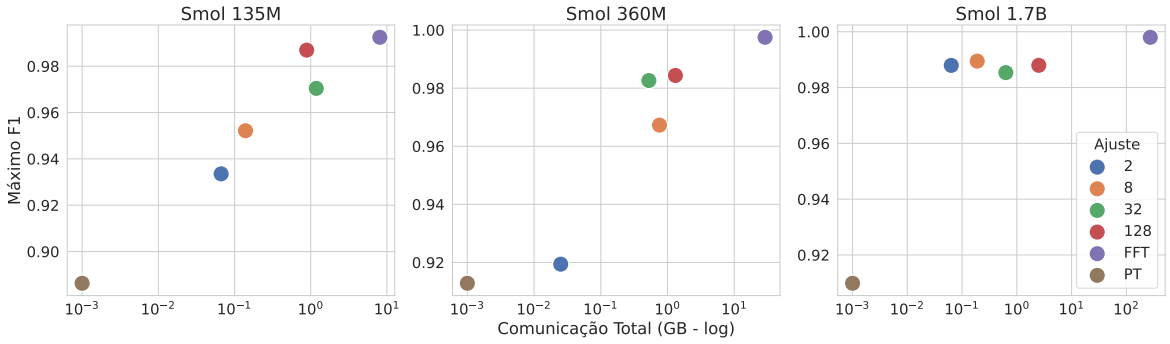

Figura 5. Total de Comunicação em escala logarítmica e máximo F1 na detecção de anomalias para cada modelo e tipo do ajuste.

Outro ponto importante a se destacar é que no uso do maior modelo de 1.7B com o menor adaptador (R = 2) ainda mostrou performances muito altas. Assim, esse caso é preferível quando se trata de redução de comunicação, uma vez que obteve baixos valores de transmissão para atingir seu F1 máximo. Entretanto, como esse modelo possui muitos parâmetros, mesmo no caso apenas do uso do modelo para inferência, o custo é elevado requerendo assim dispositivos que disponham de alta capacidade de memória para utilização assim como grande capacidade de computação para processar os dados. A Tabela 3 evidencia a medição realizada para 1000 amostras aleatórias do conjunto de validação e coletados o tempo médio de inferência e o uso de memória da GPU ao realizar esses cálculos. Nesse caso, a adição dos adaptadores do LoRA não aumentariam a latência uma vez que são somados diretamente nos pesos originais do modelo segundo a Equação 3 [Hu et al. 2021].

Tabela 3. Tempos de processamento e consumo de memória durante a inferência de modelos de diferentes tamanhos.

Modelo	Tempo médio por amostra ± DP (ms)	Uso máximo de memória (MB)
135M	30.8908 ± 0.3973	1021.37
360M	62.3438 ± 0.8539	1981.91
1.7B	253.0354 ± 7.1095	7712.16

Os modelos menores precisam de até 7x menos memória e podem fazer inferências até 8x mais rápido, em troca de uma maior comunicação com o servidor ao longo do aprendizado. Dessa forma, ainda pode ser preferível a utilização de modelos menores em casos limitados de recursos na borda ou em aplicações críticas que demandam baixa latência na predição e identificação de anomalias.

5. Conclusão

Neste trabalho, exploramos uma abordagem eficiente em recursos de computação e comunicação para o ajuste de modelos de linguagem destinados à detecção de anomalias em *logs* de sistemas privados e distribuídos entre múltiplos clientes da rede. Através do FL, possibilitamos um processo colaborativo e privado, permitindo que clientes treinem modelos sem expor seus dados. Além disso, empregamos técnicas de treinamento eficientes que resultam em uma redução significativa do uso de recursos computacionais, alcançando uma performance superior a 98% no F1-score para a detecção de anomalias.

Esta proposta, abrangendo todas as etapas de pré-processamento dos dados, ajuste eficiente e tomada de decisão para a detecção, estabelece diretrizes que contribuem para a construção de sistemas mais seguros, eficientes, privados e colaborativos. Isso possibilita avanços significativos em cenários tanto em dispositivos móveis com limitações de recursos quanto em ambientes com maior poder de processamento, além da criação de soluções para aplicações críticas com baixa latência. Por fim, trabalhos futuros incluem a realização de testes em mais cenários e *datasets*, abrangendo diferentes tipos de *logs* gerados para diversos contextos. Além disso, exploraremos mais a fundo as possíveis diferenças nas distribuições de dados dos clientes (dados não-IIDs).

6. Agradecimentos

Este projeto foi apoiado pelo Ministério da Ciência, Tecnologia e Inovações, com recursos da Lei nº 8.248, de 23 de outubro de 1991, no âmbito do PPI-SOFTEX, coordenado pela Softex e publicado Arquitetura Cognitiva (Fase 3), DOU 01245.003479/2024-10. Este trabalho foi parcialmente patrocinado pelo projeto #23/00673-7, São Paulo Instituto de Pesquisa (FAPESP), CNPq projeto #405940/2022-0 e CAPES projeto #88887.954253/2024-00 e PIND UNICAMP projeto #519.287

Disponibilidade de Artefatos

Em linha com princípios e práticas de Ciência Aberta, o acesso ao conjunto de dados utilizado (Hadoop Distributed File System - HDSF [Xu et al. 2009]) pode ser realizado através do link: `https://github.com/logpai/loghub/tree/master/HDFS`. Da mesma forma, disponibilizamos a implementação da nossa abordagem através do repositório: `https://github.com/GabrielTalasso/FL-LLM-AD`.

Referências

Allal, L. B., Lozhkov, A., Bakouch, E., von Werra, L., and Wolf, T. (2024). Smollm - blazingly fast and remarkably powerful.

Almodovar, C., Sabrina, F., Karimi, S., and Azad, S. (2024). Logfit: Log anomaly detection using fine-tuned language models. *IEEE Transactions on Network and Service Management*.

Cho, Y. J., Wang, J., and Joshi, G. (2020). Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv preprint arXiv:2010.01243*.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Guan, W., Cao, J., Qian, S., and Gao, J. (2024). Logllm: Log-based anomaly detection using large language models. *arXiv preprint arXiv:2411.08561*.

Guo, H., Yuan, S., and Wu, X. (2021). Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Li, B., Ma, S., Deng, R., Choo, K.-K. R., and Yang, J. (2022). Federated anomaly detection on system logs for the internet of things: A customizable and communication-efficient approach. *IEEE Transactions on Network and Service Management*, 19(2):1705–1716.

Liu, Y. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Pang, G., Shen, C., Cao, L., and Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38.

Qi, J., Huang, S., Luan, Z., Yang, S., Fung, C., Yang, H., Qian, D., Shang, J., Xiao, Z., and Wu, Z. (2023). Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 273–280. IEEE.

Souza, A., Bittencourt, L., Cerqueira, E., Loureiro, A., and Villas, L. (2023). Dispositivos, eu escolho vocês: Seleção de clientes adaptativa para comunicação eficiente em aprendizado federado. In *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 1–14, Porto Alegre, RS, Brasil. SBC.

Wang, F., Zhang, Z., Zhang, X., Wu, Z., Mo, T., Lu, Q., Wang, W., Li, R., Xu, J., Tang, X., et al. (2024). A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*.

Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 117–132, New York, NY, USA. Association for Computing Machinery.

Ye, R., Wang, W., Chai, J., Li, D., Li, Z., Xu, Y., Du, Y., Wang, Y., and Chen, S. (2024). Openfedllm: Training large language models on decentralized private data via federated learning.