Defesa Distribuída: Avaliação do Aprendizado Federado para Detecção de Ataques em Sistemas Operacionais

Mateus V. de Castro[1], Bruno Kimura[1], Allan M. de Souza[2], Leandro Villas[2], Joahannes B. D. da Costa[3]

[1]Universidade Federal de São Paulo (UNIFESP), São José dos Campos, Brasil
[2]Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil
[3]Universidade de São Paulo (USP), Escola Politécnica (Poli), São Paulo, Brasil

{mateus.vespasiano, bruno.kimura}@unifesp.br, {allanms, lvillas}@unicamp.br
joahannes@usp.br

Resumo. *A detecção dinâmica de aplicações móveis maliciosas requer rastros de execução amplos, recentes e representativos. Como a maioria das abordagens utiliza técnicas de aprendizado de máquina para essa detecção, há demanda por centralizar grandes volumes de dados para o treinamento dos modelos. Porém, essa centralização é dificultada por diversos desafios. A coleta de sequências de chamadas de sistema requer instrumentação custosa das aplicações, as diferenças entre dispositivos e versões do sistema geram distribuições heterogêneas de dados, e a concentração de dados compromete a privacidade e consome recursos críticos em dispositivos móveis e de borda. Assim, este trabalho propõe um mecanismo de detecção distribuído baseado em Aprendizado Federado, que permite treinar modelos localmente e compartilhar apenas suas atualizações com um servidor de agregação. O método alcança 0,900 de acurácia, 0,896 de F1-macro e 0,922 de AUPRC, além de reduzir significativamente o tráfego de dados em comparação com abordagens centralizadas, conciliando a efetividade das abordagens de aprendizado de máquina com as restrições de privacidade e recursos dos dispositivos.*

Abstract. *Dynamic detection of malicious mobile applications requires broad, recent, and representative execution traces. Since most approaches rely on machine learning techniques for this detection, there is a demand for centralizing large volumes of data to train the models. However, this centralization is hindered by several challenges. Collecting system call sequences requires costly instrumentation of applications, differences between devices and system versions generate heterogeneous data distributions, and data concentration compromises privacy and consumes critical resources on mobile and edge devices. Thus, this work proposes a distributed detection mechanism based on Federated Learning, which enables training models locally and sharing only their updates with an aggregation server. The method achieves 0.900 accuracy, 0.896 macro-F1, and 0.922 AUPRC, in addition to significantly reducing data traffic compared to centralized approaches, reconciling the effectiveness of machine learning approaches with the privacy and resource constraints of the devices.*

1. Introdução

A análise dinâmica de aplicações é uma abordagem relevante para a detecção de software malicioso, pois permite observar eventos produzidos durante a execução, em vez de depender exclusivamente de características extraídas do pacote da aplicação [Mehedi et al. 2026]. Entre esses eventos, as chamadas de sistema (*syscalls* em inglês)

constituem a interface utilizada pelos processos para solicitar serviços ao sistema operacional, como acesso a arquivos, memória e comunicação. Consequentemente, sequências dessas chamadas podem representar aspectos do comportamento de uma aplicação e têm sido empregadas na identificação de padrões anômalos e maliciosos [Duan et al. 2024].

Uma dificuldade para pesquisadores e desenvolvedores desse tipo de detector é a obtenção de conjuntos de dados dinâmicos suficientemente amplos, atualizados e representativos. A coleta de *syscalls* requer instalar e executar aplicações em ambientes monitorados, registrar eventos de execução e produzir rótulos confiáveis. Trabalhos recentes destacam tanto a escassez de conjuntos públicos de alta qualidade baseados em *syscalls* quanto o custo computacional e de armazenamento associado à coleta dinâmica em larga escala [Duan et al. 2024].

Além disso, uma coleção obtida em um único ambiente pode não representar adequadamente determinados ecossistema. Estudos que executaram as mesmas aplicações em dispositivos reais e virtuais, empregando aprendizado de máquina para a classificação, identificaram diferenças significativas nos perfis de *syscalls*, bem como uma queda no desempenho dos classificadores quando as plataformas de treinamento e teste eram distintas [Guerra-Manzanares and Välbe 2022]. Vieses temporais e diferenças entre as distribuições de treinamento e implantação também podem produzir estimativas excessivamente otimistas para classificadores de *malware* [Pendlebury et al. 2019].

Ademais, a coleta e centralização desses registros em um servidor de segurança introduzem desafios significativos relacionados à privacidade, segurança e uso de recursos computacionais. Dispositivos móveis e de borda frequentemente manipulam dados sensíveis, como informações pessoais ou corporativas, cujo compartilhamento em formato bruto pode violar requisitos legais (como a LGPD, no Brasil) ou políticas de confidencialidade. Além disso, limitações de largura de banda, energia e conectividade tornam inviável a transmissão contínua de grandes volumes de dados [Libardi et al. 2025].

Esses desafios motivam a adoção do Aprendizado Federado (AF), uma abordagem de aprendizado de máquina distribuído em que cada participante (denominado cliente) treina localmente um modelo utilizando seus próprios dados e compartilha apenas atualizações de parâmetros com um servidor central de agregação [McMahan et al. 2017]. Dessa forma, os dados sensíveis permanecem no dispositivo de origem, reduzindo riscos de vazamento e atendendo a requisitos de privacidade. Além disso, o modelo global resultante se beneficia da diversidade de dados entre os participantes, o que tende a aumentar sua capacidade de generalização, especialmente em cenários onde a distribuição dos dados é heterogênea/desbalanceada, também conhecido como dados não-Independentes e Identicamente Distribuídos (IID) [de Souza et al. 2024].

Nesse contexto, este trabalho apresenta a avaliação e a proposta de uma abordagem de defesa distribuída para a detecção de ataques em aplicação. A abordagem utiliza sequências de *syscalls* como base para a identificação de comportamentos maliciosos e emprega AF para o treinamento distribuído dos modelos, preservando a descentralização dos dados e favorecendo a detecção colaborativa de ameaças. De maneira geral, as principais contribuições incluem: i) a concepção de um *pipeline* completo de pré-processamento que constrói um vocabulário de chamadas de sistema, extrai informações temporais (Δt) entre eventos, normaliza os dados e organiza as sequências em janelas deslizantes, viabilizando inferência em tempo quase real; ii) a aplicação e comparação de três arquiteturas de aprendizado profundo (*Gated Recurrent Unit (GRU)*, *Long Short-Term Memory (LSTM)* e *Transformer*), capazes de capturar dependências sequenciais e padrões temporais complexos nas sequências de *syscalls*; iii) a implementação de uma

simulação de AF sob condições realistas de distribuição não-IID entre clientes, incluindo *client sampling*, e a avaliação de quatro estratégias de agregação (FedAvg, FedProx, FedYogi e FedAdam); e $iv)$ uma análise abrangente de desempenho utilizando métricas robustas a desbalanceamento de classes (F_1-macro e AUPRC), bem como métricas operacionais críticas para sistemas de segurança, como *recall* da classe maliciosa e taxa de falsos positivos. Adicionalmente, são discutidas as implicações práticas desses resultados para implantação em ambientes reais.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta e discute os principais trabalhos relacionados. A Seção 3 apresenta a fundamentação teórica. A Seção 4 descreve a metodologia de avaliação. A Seção 5 apresenta e discute os resultados obtidos. Por fim, a Seção 6 apresenta a conclusão e trabalhos futuros.

2. Trabalhos Relacionados

A detecção de intrusões baseada em *host* é explorada desde a década de 1990, com a premissa de que o padrão de *syscalls* de um programa legítimo permanece relativamente estável, enquanto ataques alteram essa sequência [Forrest et al. 1996]. Trabalhos pioneiros utilizaram *janela de n-gramas* de chamadas de sistema e algoritmos de aprendizado de máquina clássicos como árvores de decisão e máquinas de vetor de suporte para distinguir execuções benignas de comportamentos anômalos [Khreich et al. 2017]. Posteriormente, técnicas baseadas em modelos de Markov [Ahmadian Ramaki et al. 2018] e *sequence alignment* refinaram a capacidade de capturar dependências de longo prazo entre eventos [Tartakovsky et al. 2006].

Com o avanço de arquiteturas neurais, redes neurais recorrentes (LSTM, GRU) e convolucionais foram empregadas para modelar sequências de *syscalls* como séries temporais, demonstrando melhor capacidade de generalização em relação a métodos baseados em assinatura [Du et al. 2017, Liu et al. 2018]. Além disso, o desenvolvimento de *Transformers* e mecanismos de atenção permitiu aprender relacionamentos de longo alcance em séries de eventos com custo de treinamento comparável [Vaswani et al. 2017]. Entretanto, a maior parte dessas abordagens assume disponibilidade total dos *logs* em um ambiente centralizado, o que pode potencialmente violar requisitos de privacidade.

Nesse cenário, o AF se mostra uma alternativa viável para a defesa distribuída, possibilitando o treinamento de modelos com dados privados. Aplicações de AF nesse contexto incluem o trabalho de [Belarbi et al. 2023], que usou LSTM em um cenário federado para detectar *malwares* em dispositivos Android, e o de [Husnoo et al. 2024], que investigou *autoencoders* para detecção de anomalias em registros de rede.

De maneira geral, este estudo distingue-se dos trabalhos apresentados por combinar pré-processamento temporal robusto (normalização de Δt e janelamento consistente) com uma comparação sistemática de diferentes arquiteturas (GRU, LSTM e Transformer) e estratégias de agregação (FedAvg, FedProx, FedAdam e FedYogi) em um conjunto de dados público de *logs* de *syscalls*. Além disso, faz um estudo abrangente de exploração da sensibilidade das métricas em distribuição de dados não homogênea (não-IID) entre clientes e faz uma discussão sobre como as escolhas de modelo e algoritmo federado impactam a taxa de detecção e de falsos positivos no sistema.

3. Fundamentação Teórica

O AF surge como alternativa para treinar modelos de aprendizado de máquina em ambientes distribuídos preservando a privacidade dos dados. McMahan *et al.* [McMahan et al. 2017] formalizaram o algoritmo *FedAvg*, demonstrando que uma média

ponderada de gradientes locais pode produzir modelos comparáveis ao treinamento centralizado. Extensões posteriores, como o *FedProx* [Li et al. 2020] e os métodos adaptativos *FedAdam* e *FedYogi*, propostos por Reddi *et al.* [Reddi et al. 2021], incorporaram regularização proximal ou otimizadores adaptativos para mitigar divergências de distribuição entre clientes.

A Figura 1 ilustra o processo iterativo do AF, que possibilita o treinamento colaborativo de modelos sem o compartilhamento de dados brutos. O ciclo tem início com o servidor enviando o modelo global aos dispositivos clientes (Rótulo 1, linhas contínuas), que realizam treinamento local com seus próprios dados e recursos computacionais (Rótulo 2). Em seguida, apenas as atualizações resultantes desse treinamento, como pesos e parâmetros atualizados, são enviadas de volta ao servidor (Rótulo 3, linhas tracejadas), que as agrega para atualizar o modelo global e reiniciar o ciclo. A sequência composta pelo envio do modelo global aos clientes, pelo treinamento local e pelo retorno das atualizações ao servidor é denominada *rodada de treinamento*. Esse mecanismo preserva a privacidade dos dados locais e otimiza a comunicação, uma vez que apenas o conhecimento extraído, e não a informação sensível, trafega pela rede.

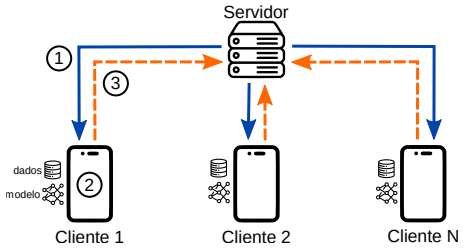


Figura 1. Processo de treinamento distribuído em Aprendizado Federado (AF).

Nesse contexto, a aplicação do AF torna-se particularmente vantajosa para a segurança de sistemas distribuídos, pois permite que diferentes nós da rede aprendam padrões de execução maliciosos sem expor *logs* de atividade sensíveis. Ao treinar modelos localmente com base em sequências de chamadas de sistema, o sistema torna-se capaz de capturar as particularidades de cada ambiente, ao mesmo tempo em que contribui para um modelo global de detecção mais robusto e generalista. Sob essa perspectiva, as próximas seções detalham como essa defesa distribuída é aplicada.

4. Metodologia

Esta seção descreve o conjunto de dados utilizado, o processo de preparação das sequências de *syscalls*, as arquiteturas de modelos avaliados e o protocolo federado empregado nas simulações.

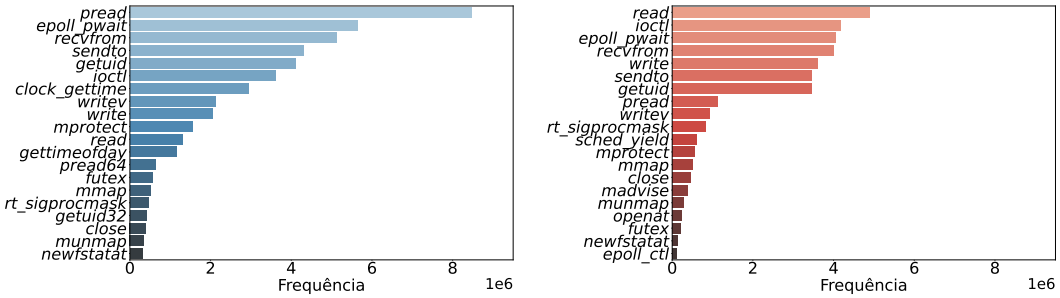
4.1. Conjunto de dados e pré-processamento

Os experimentos foram realizados com o conjunto de dados *Android System call Dataset*[1], que contém traços de execução de aplicações Android classificados como benignos ou maliciosos. Cada amostra é um arquivo *.csv* composto pelas colunas `timestamp` e `syscall`. O conjunto possui um total de 5.381 arquivos, dos quais 2.908 são rotulados como maliciosos e 2.473 como benignos, contendo uma leve assimetria a ser considerada na avaliação. Essa distribuição relativamente equilibrada, mas ligeiramente enviesada,

[1] `https://www.kaggle.com/datasets/akarshnair/android-system-call-dataset/`

justifica o uso de métricas macro (como *F1*-macro e AUPRC) para avaliar corretamente a capacidade dos modelos de detectar ataques sem favorecer a classe majoritária.

A Figura 2 apresenta as 20 *syscalls* mais frequentes em amostras benignas e maliciosas. Algumas chamadas, como `pread`, `epoll_pwait` e `recvfrom`, predominam em aplicações benignas, enquanto `read` e `ioctl` são muito mais comuns em amostras maliciosas. Outras diferenças notáveis são a presença de `sched_yield`, `madvise` e `epoll_ctl` apenas nas listas de ocorrências maliciosas (Figura 2(b)). Essas discrepâncias sugerem que certas *syscalls* podem ser indícios de comportamento suspeito. No entanto, a Figura 2 também indica que os dois conjuntos compartilham muitas chamadas (tais como `recvfrom`, `write`, `nmap`, etc), o que reforça a necessidade de analisar sequências e correlações em vez de contagens isoladas.



(a) Benignas. (b) Maliciosas.

Figura 2. Top-20 *syscalls* em amostras de diferentes classes.

De forma complementar, a Figura 3 ilustra a correlação entre as principais *syscalls* em amostras maliciosas. Observa-se a presença de correlações positivas mais intensas, representadas pelos tons de vermelho mais escuros, especialmente envolvendo chamadas como `close`, `openat`, `ioctl`, `mmap`, `newfstatat` e `futex`. Esses agrupamentos indicam que determinadas chamadas tendem a ocorrer de maneira associada, refletindo padrões recorrentes de acesso a arquivos, manipulação de descritores, mapeamento de memória, operações de entrada/saída e sincronização. Assim, os resultados sugerem que a caracterização do comportamento malicioso não depende apenas da frequência isolada das *syscalls*, mas também das relações entre elas, evidenciando combinações de chamadas que contribuem para diferenciar esse tipo de amostra.

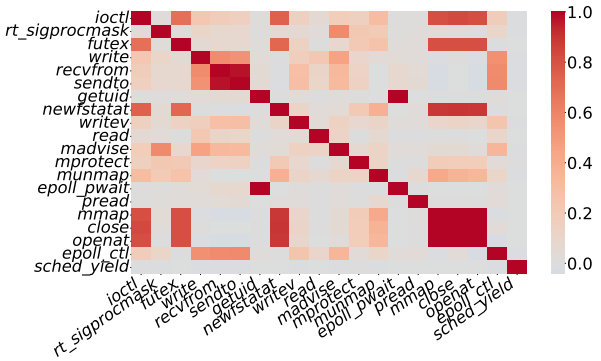
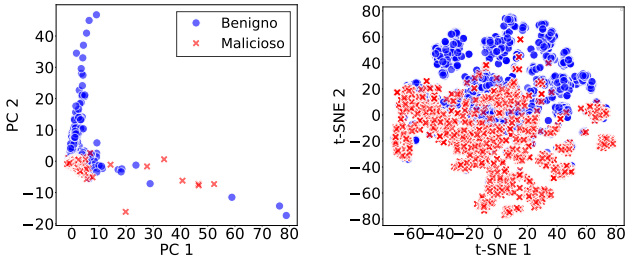


Figura 3. Correlação entre as *syscalls* em ocorrências maliciosas.

A Figura 4 compara duas técnicas de redução de dimensionalidade aplicadas às

janelas de *syscalls*, sendo a Análise de Componentes Principais (PCA) e o *t-Distributed Stochastic Neighbor Embedding (t-SNE)*. Essa representação permite visualizar a separabilidade entre amostras benignas e maliciosas no espaço de características das janelas de *syscalls*. Essas análises auxiliam na compreensão da estrutura dos dados e indicam se há padrões discriminativos que podem ser explorados pelos modelos de detecção.

Na Figura 4(a), observa-se que as duas primeiras componentes principais explicam apenas cerca de 12,2% da variância total do conjunto, resultando em um aglomerado denso com forte sobreposição entre amostras benignas e maliciosas. Esse comportamento indica que, na projeção bidimensional obtida pela PCA, a separação entre as classes é limitada, sugerindo que os padrões discriminativos das *syscalls* podem estar distribuídos em dimensões superiores ou depender de relações não lineares. Já com o t-SNE, conforme apresentado na Figura 4(b), observa-se uma estrutura não linear mais rica, composta por diferentes agrupamentos locais, alguns predominantemente associados a amostras benignas e outros a amostras maliciosas, além de regiões com sobreposição entre as classes. Isso sugere que os padrões nas frequências agregadas de *syscalls* não são facilmente separáveis por uma fronteira linear simples, indicando a presença de relações mais complexas no espaço de características. Dessa forma, a projeção sugere o uso de modelos capazes de capturar interações não lineares entre as chamadas, como redes neurais recorrentes, arquiteturas baseadas em atenção ou outros classificadores não lineares. Esse aspecto é relevante no contexto de AF, em que cada cliente pode observar distribuições locais distintas e apenas uma parte da variabilidade presente no domínio global.



(a) PCA. (b) t-SNE.

Figura 4. Aplicação de técnicas de redução de dimensionalidade.

De maneira geral, o pré-processamento proposto inicia-se com a i) leitura e sanitização dos arquivos CSV, em que eventuais variações de cabeçalho são corrigidas por meio da renomeação das duas primeiras colunas para `Timestamp` e `Syscall`, seguida da remoção de linhas com valores ausentes. Em seguida, ii) realiza-se a tokenização das chamadas, construindo um vocabulário global a partir de todos os arquivos e incluindo os símbolos especiais `<PAD>` e `<UNK>`. Com isso, cada sequência de *syscalls* é convertida em um vetor de inteiros, no qual cada valor corresponde ao índice da chamada no vocabulário. iii) Também é calculado o intervalo temporal entre chamadas consecutivas, representado por Δt, a partir da conversão dos `Timestamps` para segundos. Diferenças negativas, possivelmente causadas por reinicializações de relógio, são truncadas em zero, enquanto valores extremos são limitados ao percentil 95 para reduzir o efeito de *outliers*. Por fim, o vetor Δt é normalizado por média e desvio padrão, utilizando apenas as amostras de treino para evitar vazamento de informação. Após essa etapa, iv) cada execução é segmentada em janelas fixas de tamanho $T = 128$, com passo 96, resultando em sobreposição de 25%. Esse processo gera exemplos no formato $(\mathbf{x}, \delta\mathbf{t}, y)$, em que $\mathbf{x} \in \mathbb{N}^{128}$ representa a sequência tokenizada de chamadas, $\delta\mathbf{t} \in \mathbb{R}^{128}$ representa os inter-

valos temporais normalizados e y corresponde ao rótulo da amostra. Sequências menores que T são completadas com <PAD> e zeros. Por fim, v) os arquivos são distribuídos entre $K = 5$ clientes de forma não-IID por meio da distribuição de Dirichlet com parâmetro $\alpha = 1{,}2$, de modo que valores menores de α aumentam a concentração de classes por cliente. O particionamento preserva todas as janelas de um mesmo arquivo no mesmo cliente e realiza as divisões de treino, validação e teste por arquivo, e não por janela, evitando fuga de informação temporal entre os conjuntos.

4.2. Modelos de detecção

Os modelos tomam como entrada a janela de *syscalls* e o vetor de Δt normalizado, concatenam essas representações por *timestep* e aplicam uma camada de normalização antes do núcleo da rede para LSTM e GRU. Após o processamento sequencial, realizam um *pooling* mascarado que ignora posições de <PAD> e utilizam uma cabeça linear para produzir *logits* de classificação binária.

Os modelos considerados foram duas redes neurais recorrentes (GRU e LSTM) e um classificador *Transformer*. As redes recorrentes recebem vetores de dimensão $(\mathrm{emb\_dim} + 1)$ em cada passo, onde $\mathrm{emb\_dim} = 64$ é o tamanho do *embedding*. Ambas possuem camada recorrente com 128 unidades ocultas, 2 camadas e dropout de $0{,}3$ entre camadas. A saída final de cada janela é obtida por média ponderada (ignorando <PAD>). Já o *Transformer*, incorpora um codificador posicional senoidal e projeta o vetor de entrada para uma dimensão oculta 128 que é múltipla do número de cabeças ($n_{\mathrm{head}} = 4$). Utilizamos duas camadas de codificação e mecanismo de atenção multi-cabeças com *dropout* de $0{,}3$. Após o codificador, uma média ponderada sobre as posições válidas produz a representação global da janela.

4.3. Treinamento federado

O treinamento dos modelos é orquestrado pelo *framework* Flower e segue a seguinte dinâmica: em cada rodada o servidor inicializa o modelo global e o envia a todos os clientes; cada cliente realiza uma época de treinamento local ($E = 1$) com seu conjunto de treino, calcula as atualizações de pesos e devolve ao servidor; o servidor agrega as atualizações por média ponderada do tamanho do conjunto local. Repete-se esse processo por $R = 10$ rodadas. Para explorar variantes do algoritmo, também utilizou-se o FedProx [Li et al. 2020], que adiciona um termo proximal penalizando desvio do modelo global durante o treinamento local; o FedAdam [Reddi et al. 2021], que aplica otimização adaptativa no servidor; e o FedYogi [Reddi et al. 2021], que também utiliza momentos adaptativos no servidor, mas atualiza a estimativa de segunda ordem de forma mais conservadora, buscando controlar o crescimento excessivo das taxas adaptativas e melhorar a estabilidade da agregação em cenários federados com dados heterogêneos.

O conjunto de validação representa 20% dos dados de cada cliente e é utilizado apenas para ajustar o normalizador de tempo e monitorar sobre-ajuste local. Ao final de cada rodada, cada cliente avalia o modelo global usando seu próprio conjunto de testes. Somente as métricas resultantes são enviadas ao servidor, que as agrega para estimar o desempenho global. Para a avaliação dos modelos, foram consideradas tanto métricas globais quanto métricas por cliente. As métricas foram: **Acurácia**: fração de janelas classificadas corretamente; $\boldsymbol{F_1}$ **macro**: média harmônica do *precision* e *recall* calculada separadamente para cada classe e então média aritmética; $\boldsymbol{F_1}$ **weighted**: média ponderada do $\boldsymbol{F_1}$ por classe de acordo com sua prevalência; $\boldsymbol{Recall_+}$: razão entre verdadeiros positivos e o total de janelas maliciosas; indica a capacidade de capturar ataques (sensibilidade); **FPR** (taxa de falsos positivos): razão entre falsos positivos e o total de janelas

benignas; indicador de fadiga de alertas; e **AUROC** e **AUPRC**: área sob as curvas ROC e *precision-recall*, independentes do limiar de decisão.

Os experimentos foram conduzidos em um notebook modelo ASUS TUF Gaming F16, que possui o sistema operacional Windows 11 *Home Single Language* de 64 bits, equipado com processador Intel(R) Core(TM) 5 210H, com 12 CPUs, e 16 GB de memória RAM. A placa de vídeo utilizada é uma NVIDIA GeForce RTX 4050 Laptop GPU, com 6 GB de memória de vídeo dedicada. Além disso, o ambiente de desenvolvimento conta com Python 3.12.3, compilado com Ubuntu 13.3.0-6ubuntu2 24.04, versão 13.3.0, utilizado via *Windows Subsystem for Linux (WSL)*.

5. Resultados

Esta seção apresenta os resultados obtidos. A Tabela 1 mostra as principais métricas ao final da rodada 10 para cada combinação de arquitetura e estratégia de agregação. Cada valor corresponde à média $\bar{x}$ e ao desvio-padrão amostral s calculados a partir de três execuções independentes. As métricas *Recall*$_+$ e FPR são calculados com limiar de decisão 0,5. Para cada métrica, no contexto federado, os melhores valores (maiores para acurácia, F_1, *Recall*$_+$ e AUPRC, e menores para FPR) estão com as células destacadas. A implementação centralizada de cada modelo aparece na primeira linha correspondente.

Tabela 1. Desempenho global após a décima rodada.

Modelo	Estratégia	Métricas					
		Acurácia	F_1 macro	F_1 *weighted*	*Recall*$_+$	FPR	AUPRC
GRU	Centralizado	$0{,}896 \pm 0{,}036$	$0{,}894 \pm 0{,}039$	$0{,}896 \pm 0{,}037$	$0{,}869 \pm 0{,}092$	$0{,}083 \pm 0{,}024$	$0{,}961 \pm 0{,}011$
	FedAvg	$\mathbf{0{,}895 \pm 0{,}023}$	$\mathbf{0{,}890 \pm 0{,}025}$	$\mathbf{0{,}894 \pm 0{,}024}$	$0{,}826 \pm 0{,}050$	$\mathbf{0{,}055 \pm 0{,}009}$	$0{,}925 \pm 0{,}045$
	FedProx	$0{,}881 \pm 0{,}027$	$0{,}878 \pm 0{,}030$	$0{,}880 \pm 0{,}029$	$0{,}817 \pm 0{,}102$	$0{,}064 \pm 0{,}035$	$\mathbf{0{,}930 \pm 0{,}017}$
	FedAdam	$0{,}891 \pm 0{,}005$	$0{,}887 \pm 0{,}007$	$0{,}890 \pm 0{,}006$	$\mathbf{0{,}828 \pm 0{,}018}$	$0{,}063 \pm 0{,}003$	$0{,}861 \pm 0{,}086$
	FedYogi	$0{,}868 \pm 0{,}044$	$0{,}865 \pm 0{,}045$	$0{,}867 \pm 0{,}045$	$0{,}794 \pm 0{,}094$	$0{,}066 \pm 0{,}017$	$0{,}913 \pm 0{,}015$
LSTM	Centralizado	$0{,}934 \pm 0{,}021$	$0{,}927 \pm 0{,}013$	$0{,}934 \pm 0{,}021$	$0{,}913 \pm 0{,}007$	$0{,}057 \pm 0{,}025$	$0{,}966 \pm 0{,}006$
	FedAvg	$0{,}891 \pm 0{,}070$	$0{,}884 \pm 0{,}068$	$0{,}890 \pm 0{,}072$	$0{,}813 \pm 0{,}123$	$0{,}048 \pm 0{,}007$	$0{,}910 \pm 0{,}021$
	FedProx	$0{,}898 \pm 0{,}006$	$0{,}895 \pm 0{,}006$	$0{,}898 \pm 0{,}006$	$\mathbf{0{,}866 \pm 0{,}013}$	$0{,}077 \pm 0{,}003$	$0{,}919 \pm 0{,}013$
	FedAdam	$0{,}863 \pm 0{,}025$	$0{,}856 \pm 0{,}033$	$0{,}861 \pm 0{,}028$	$0{,}779 \pm 0{,}108$	$0{,}079 \pm 0{,}039$	$0{,}867 \pm 0{,}044$
	FedYogi	$\mathbf{0{,}900 \pm 0{,}030}$	$\mathbf{0{,}896 \pm 0{,}031}$	$\mathbf{0{,}899 \pm 0{,}031}$	$0{,}830 \pm 0{,}055$	$\mathbf{0{,}047 \pm 0{,}007}$	$\mathbf{0{,}922 \pm 0{,}006}$
Transformer	Centralizado	$0{,}903 \pm 0{,}014$	$0{,}895 \pm 0{,}016$	$0{,}903 \pm 0{,}013$	$0{,}853 \pm 0{,}025$	$0{,}068 \pm 0{,}023$	$0{,}936 \pm 0{,}027$
	FedAvg	$0{,}852 \pm 0{,}073$	$0{,}848 \pm 0{,}072$	$0{,}850 \pm 0{,}075$	$0{,}777 \pm 0{,}129$	$0{,}076 \pm 0{,}016$	$0{,}891 \pm 0{,}016$
	FedProx	$\mathbf{0{,}886 \pm 0{,}022}$	$\mathbf{0{,}876 \pm 0{,}018}$	$\mathbf{0{,}885 \pm 0{,}024}$	$0{,}800 \pm 0{,}063$	$\mathbf{0{,}056 \pm 0{,}015}$	$0{,}853 \pm 0{,}036$
	FedAdam	$0{,}850 \pm 0{,}045$	$0{,}843 \pm 0{,}044$	$0{,}848 \pm 0{,}046$	$0{,}757 \pm 0{,}076$	$0{,}079 \pm 0{,}013$	$0{,}886 \pm 0{,}039$
	FedYogi	$0{,}846 \pm 0{,}063$	$0{,}839 \pm 0{,}059$	$0{,}846 \pm 0{,}062$	$\mathbf{0{,}816 \pm 0{,}018}$	$0{,}138 \pm 0{,}113$	$\mathbf{0{,}906 \pm 0{,}015}$

Considerando inicialmente os modelos centralizados, observa-se que a LSTM apresentou o melhor desempenho geral. O GRU obteve desempenho próximo, com acurácia de $\mathbf{0{,}896 \pm 0{,}036}$, F_1 macro de $\mathbf{0{,}894 \pm 0{,}039}$ e AUPRC de $\mathbf{0{,}961 \pm 0{,}011}$, enquanto o *Transformer* apresentou acurácia de $\mathbf{0{,}903 \pm 0{,}014}$, F_1 macro de $\mathbf{0{,}895 \pm 0{,}016}$ e AUPRC de $\mathbf{0{,}936 \pm 0{,}027}$. Os resultados indicam que, neste cenário, a LSTM foi a configuração de referência mais forte, embora o desempenho da GRU e do *Transformer* permaneça relativamente próximo em algumas métricas.

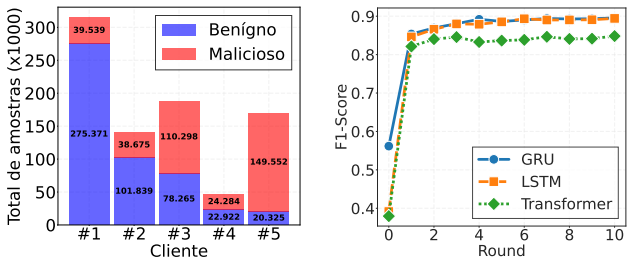
No cenário federado, a LSTM com FedYogi obteve o melhor desempenho agregado quando consideradas acurácia, F_1 macro, F_1 *weighted*, FPR e AUPRC, alcançando $\mathbf{0{,}900 \pm 0{,}030}$, $\mathbf{0{,}896 \pm 0{,}031}$, $\mathbf{0{,}899 \pm 0{,}031}$, $\mathbf{0{,}047 \pm 0{,}007}$ e $\mathbf{0{,}922 \pm 0{,}006}$, respectivamente. No entanto, a LSTM com FedProx obteve o maior *Recall*$_+$ entre as configurações federadas, com $\mathbf{0{,}866 \pm 0{,}013}$, indicando maior capacidade de detecção da classe maliciosa. Assim, a escolha entre FedYogi e FedProx depende do objetivo, se reduzir alarmes falsos ou maximizar a detecção de ataques.

Para o GRU, o FedAvg apresentou o melhor equilíbrio geral entre as estratégias federadas, obtendo acurácia de $\mathbf{0{,}895 \pm 0{,}023}$, F_1 macro de $\mathbf{0{,}890 \pm 0{,}025}$, F_1 *weighted*

de $\mathbf{0{,}894 \pm 0{,}024}$ e a menor FPR do grupo, com $\mathbf{0{,}055 \pm 0{,}009}$. Embora o FedAdam tenha obtido o maior *Recall*$_+$ para o GRU, com $\mathbf{0{,}828 \pm 0{,}018}$, a diferença em relação ao FedAvg foi pequena, enquanto sua AUPRC foi consideravelmente menor ($\mathbf{0{,}861 \pm 0{,}086}$). O FedProx apresentou a maior AUPRC para o GRU, com $\mathbf{0{,}930 \pm 0{,}017}$, indicando boa capacidade de ordenação das janelas maliciosas, mas com desempenho ligeiramente inferior em acurácia e $\boldsymbol{F_1}$. No caso do *Transformer*, o FedProx apresentou os melhores valores de acurácia, $\boldsymbol{F_1}$ macro, $\boldsymbol{F_1}$ *weighted* e FPR, com $\mathbf{0{,}886 \pm 0{,}022}$, $\mathbf{0{,}876 \pm 0{,}018}$, $\mathbf{0{,}885 \pm 0{,}024}$ e $\mathbf{0{,}056 \pm 0{,}015}$, respectivamente. Por outro lado, o FedYogi apresentou o maior *Recall*$_+$ ($\mathbf{0{,}816 \pm 0{,}018}$) e a maior AUPRC ($\mathbf{0{,}906 \pm 0{,}015}$) para o *Transformer*, mas também resultou na maior FPR ($\mathbf{0{,}138 \pm 0{,}113}$), o que pode aumentar o número de alarmes falsos em um ambiente operacional.

Em aplicações de segurança, o principal compromisso está entre detectar o maior número possível de ataques e limitar a quantidade de alarmes falsos. Sob essa perspectiva, configurações como LSTM com FedProx são atrativas quando o objetivo principal é maximizar a detecção de amostras maliciosas, pois apresentaram o maior *Recall*$_+$ entre os modelos federados. Por outro lado, LSTM com FedYogi e GRU com FedAvg são mais adequadas quando se deseja reduzir falsos positivos, mantendo desempenho global elevado. Essa distinção é relevante porque, em sistemas de triagem manual, pode ser aceitável aumentar a FPR para reduzir falsos negativos. Porém, em mecanismos de bloqueio automático, uma FPR menor tende a ser mais importante para evitar interrupções indevidas em execuções benignas.

Em relação ao treinamento centralizado, as melhores configurações federadas mantiveram desempenho competitivo. No GRU, o FedAvg apresentou acurácia praticamente equivalente à versão centralizada ($\mathbf{0{,}895 \pm 0{,}023}$ contra $\mathbf{0{,}896 \pm 0{,}036}$), além de menor FPR. Para a LSTM, o FedYogi ficou 3,4% abaixo da versão centralizada em acurácia, mas obteve a menor FPR entre as estratégias avaliadas. No caso do *Transformer*, o FedProx ficou 1,7% abaixo da versão centralizada em acurácia. Esses resultados indicam que o AF mentém desempenho próximo ao treinamento centralizado, embora o impacto varie conforme a arquitetura e a estratégia de agregação.



(a) Distribuição de amostras. (b) $\boldsymbol{F_1}$ *score*.

Figura 5. Análise da distribuição de dados e comparação entre modelos.

Embora as métricas globais resultem da agregação das avaliações locais, a análise por cliente mostra a heterogeneidade significativa decorrente do particionamento não-IID, conforme mostra a Figura 5(a). Alguns clientes apresentaram prevalência de até 90% de sequências maliciosas, enquanto outros tinham majoritariamente dados benignos. Essa assimetria afeta diretamente as métricas locais, pois clientes com baixa representação de uma das classes podem apresentar maior instabilidade no $\boldsymbol{F_1}$ macro e em métricas dependentes da presença de exemplos positivos e negativos. Nesse contexto, estratégias como

FedProx e FedYogi tornam-se relevantes por buscarem reduzir os efeitos da divergência entre distribuições locais durante a agregação global.

Outra observação é a rápida convergência dos modelos: a maior parte do ganho ocorre na primeira rodada de agregação, passando de um $\boldsymbol{F_1}$ macro em torno de 0,39 para valores acima de 0,84 já na rodada 1, conforme mostrado na Figura 5(b). Nas rodadas seguintes, as curvas tendem a se estabilizar, com oscilações pequenas até a décima rodada. Esse comportamento sugere que, para as configurações avaliadas, dez rodadas são suficientes para atingir um platô de desempenho, especialmente para os modelos recorrentes. O *Transformer*, por sua vez, apresenta desempenho inferior nas configurações avaliadas e maior sensibilidade à estratégia de agregação, indicando que pode demandar mais rodadas, maior volume de dados ou ajustes adicionais de hiperparâmetros para explorar plenamente sua capacidade de modelagem.

Além do desempenho preditivo, também foi avaliado o custo de comunicação entre clientes e servidor. Na abordagem centralizada, seria necessário transferir aproximadamente 2,6 GB de dados brutos para um servidor central antes do treinamento. Esse volume corresponde ao tráfego do próprio conjunto de dados e representa não apenas maior consumo de banda, mas também maior exposição de informações sensíveis, uma vez que os registros de chamadas de sistema deixariam os dispositivos de origem. No AF, por outro lado, os dados permanecem localmente nos clientes, e apenas parâmetros do modelo e métricas de avaliação são transmitidos. A Tabela 2 resume o tráfego estimado para $\boldsymbol{K = 5}$ clientes e $\boldsymbol{R = 10}$ rodadas.

Tabela 2. Estimativa de tráfego de comunicação no treinamento federado.

Modelo	Parâmetros por mensagem	Treino por rodada	Treino em 10 rodadas	Com avaliação em 10 rodadas
GRU	725.008 bytes	7,25 MB	72,5 MB	108,8 MB
LSTM	956.944 bytes	9,57 MB	95,7 MB	143,5 MB
Transformer	4.208.648 bytes	42,1 MB	420,9 MB	631,3 MB

De modo geral, os resultados indicam que o AF é uma alternativa viável para detecção de ataques baseada em *syscalls*, pois evita a centralização dos dados brutos e mantém desempenho competitivo em relação ao treinamento centralizado.

6. Conclusão

Este trabalho avaliou a viabilidade de detectar comportamentos maliciosos em sistemas operacionais móveis, sem a necessidade de centralizar os *logs* de execução. Para isso, foi proposto um *pipeline* de preparação dos dados que preserva a coerência temporal das chamadas, incorpora a dinâmica entre eventos e permite o treinamento distribuído de modelos com AF em um cenário com dados desbalanceados (não-IID). Os resultados indicaram que as estratégias federadas alcançaram desempenho competitivo em relação ao treinamento centralizado com poucas rodadas de comunicação. Apesar dos resultados promissores, permanecem desafios relacionados à representação de distribuições reais de *syscalls*, à otimização de hiperparâmetros e à investigação de arquiteturas mais adequadas para sequências curtas. Trabalhos futuros incluem a validação em outros conjuntos de dados, a análise do custo energético em dispositivos móveis reais e a comparação com *baselines* da literatura, incluindo outras técnicas de aprendizado de máquina e soluções tradicionais baseadas em assinaturas, comumente empregadas em produtos antivírus.

Disponibilidade de Artefatos

O código-fonte e os dados utilizados nos experimentos deste trabalho estão disponíveis em: `https://github.com/mateusvdcastro/DefesaDistribuida`.

Uso de Inteligência Artificial (IA) Generativa

Utilizou-se a ferramenta ChatGPT (modelo GPT-5.5) para auxiliar no aprimoramento e na correção textual. A ferramenta também foi empregada para revisar a tradução do resumo para o *abstract* em língua inglesa.

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.

Referências

Ahmadian Ramaki, A., Rasoolzadegan, A., and Javan Jafari, A. (2018). A systematic review on intrusion detection based on the hidden markov model. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11(3):111–134.

Belarbi, O., Spyridopoulos, T., Anthi, E., Mavromatis, I., Carnelli, P., and Khan, A. (2023). Federated deep learning for intrusion detection in iot networks. In *2023 IEEE Global Communications Conference (GLOBECOM)*, pages 237–242, Kuala Lumpur, Malaysia. IEEE.

de Souza, A. M., Maciel, F., da Costa, J. B. D., Bittencourt, L. F., Cerqueira, E., Loureiro, A. A., and Villas, L. A. (2024). Adaptive Client Selection with Personalization for Communication Efficient Federated Learning. *Ad Hoc Networks*, 157:103462.

Du, M., Li, F., Zheng, G., and Srikumar, V. (2017). DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298.

Duan, G., Liu, H., Cai, M., Sun, J., and Chen, H. (2024). Madroid: A maliciousness-aware multifeatured dataset for detecting android malware. *Computers Security*, 144:103969.

Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for unix processes. In *Proceedings 1996 IEEE symposium on security and privacy*, pages 120–128. IEEE.

Guerra-Manzanares, A. and Välbe, M. (2022). Cross-device behavioral consistency: Benchmarking and implications for effective android malware detection. *Machine Learning with Applications*, 9:100357.

Husnoo, M. A., Anwar, A., Haque, M. E., and Mahmood, A. N. (2024). Decentralized federated anomaly detection in smart grids: a p2p gossip approach. *arXiv preprint arXiv:2407.15879*.

Khreich, W., Khosravifar, B., Hamou-Lhadj, A., and Talhi, C. (2017). An anomaly detection system based on variable n-gram features and one-class svm. *Information and Software Technology*, 91:186–197.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450.

Libardi, G. M. A., Kimura, B. Y. L., and da Costa, J. B. D. (2025). Menos é Mais: Avaliação do Impacto da Compressão de Modelos na Eficiência do Aprendizado Federado. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 290–297. SBC.

Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. (2018). Host-Based Intrusion Detection System with System Calls: Review and Future Trends. *ACM computing surveys (CSUR)*, 51(5):1–36.

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.

Mehedi, S. T., Islam, C., Ramachandran, G., and Jurdak, R. (2026). DySec: A Machine Learning-Based Dynamic Analysis for Detecting Malicious Packages in PyPI Ecosystem. *IEEE Transactions on Information Forensics and Security*, 21:1316–1331.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., and Cavallaro, L. (2019). TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, Santa Clara, CA. USENIX Association.

Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., and McMahan, H. B. (2021). Adaptive Federated Optimization. In *International Conference on Learning Representations*.

Tartakovsky, A., Rozovskii, B., Blazek, R., and Kim, H. (2006). A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. *IEEE Transactions on Signal Processing*, 54(9):3372–3382.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.