Tune3: Otimização Multiestágio de Hiperparâmetros com Redução de Custo e Maior Estabilidade em Modelos de Detecção de Malware Android

Lucas Ferreira Areias de Oliveira[1] **, Angelo Gaspar Diniz**[1]
Diego Kreutz[1] **, Dionatan Schmidt**[1] **, Rodrigo Mansilha**[1]

[1] AI Horizon Labs
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

{lucasareias,angelonogueira}.aluno@unipampa.edu.br
{diegokreutz,dionatanschmidt,rodrigomansilha}@unipampa.edu.br

Resumo. *O Tune3 é um método adaptativo multiestágio para otimizar hiperparâmetros em modelos neurais para cibersegurança. Ele integra inicialização informada, amostragem de extremos e refinamento iterativo, reduzindo o custo e o número de execuções necessárias. Em testes com conjuntos Android, o Tune3 iguala ou supera o Random Search com até 44% menos tempo de busca, mantendo estabilidade e qualidade na geração e classificação de dados.*

1. Introdução

A escolha adequada dos hiperparâmetros em modelos de classificação e geração afeta diretamente o custo computacional, a estabilidade do treinamento e a qualidade dos dados produzidos [Liao et al. 2022]. Configurações inadequadas podem resultar em redes instáveis, convergência lenta ou dados sintéticos de baixa fidelidade, especialmente à medida que o espaço de busca cresce. A literatura recente sobre otimização de hiperparâmetros em redes neurais abrange desde ajustes manuais e estratégias clássicas, como *Grid Search* e *Random Search* [Bergstra and Bengio 2012], até abordagens automatizadas e adaptativas baseadas em exploração estocástica, metaheurísticas e métodos bayesianos [Bischl et al. 2023].

Apesar desses avanços, observa-se que técnicas automatizadas permanecem pouco exploradas no contexto de modelos geradores, em particular aqueles voltados à síntese de dados tabulares, como *CTGAN* e *TVAE*, amplamente utilizados em ferramentas como MalDataGen[1] e SDV[2]. Em classificadores neurais recorrentes, como *LSTM* e *GRU*, as abordagens existentes variam significativamente quanto a intervalos de épocas, tamanhos de lote, taxas de aprendizado e funções de ativação, frequentemente sem justificativas metodológicas claras ou análise sistemática de estabilidade.

Além disso, a maior parte dos trabalhos similares (ver Seção 2) adota espaços de busca amplos e homogêneos, sem mecanismos de priorização por risco, como a penalização explícita de falsos negativos, e sem estratégias progressivas de refinamento capazes de reduzir o custo total da busca e evitar reavaliações redundantes. Em domínios

[1] https://github.com/SBSeg25/MalDataGen
[2] https://github.com/sdv-dev/SDV

sensíveis como a cibersegurança, onde decisões incorretas podem resultar em falhas de detecção e aumento de superfície de ataque, tais limitações tornam-se particularmente críticas.

Na detecção de malware Android, o ajuste eficiente de hiperparâmetros é essencial para garantir robustez, capacidade de generalização e sensibilidade a ameaças reais [Ozkan-Okay et al. 2024]. Abordagens tradicionais, como *Grid Search* e *Random Search*, tornam-se rapidamente inviáveis em espaços extensos devido ao alto custo computacional, à variabilidade dos resultados e à incapacidade de direcionar a busca para regiões promissoras [Bergstra and Bengio 2012].

Motivado por essas limitações, este trabalho apresenta o método *Tune3*, uma estratégia multiestágio adaptativa projetada para reduzir o número de execuções necessárias para encontrar configurações de alto desempenho. O método foi avaliado em dois conjuntos de dados Android amplamente utilizados na literatura, *DREBIN-215* e *DefenseDroid_API_Degree*, demonstrando ganhos expressivos de eficiência e estabilidade.

2. Trabalhos Relacionados

A Tabela 1 apresenta um panorama comparativo de estudos recentes que aplicam diferentes estratégias de otimização de hiperparâmetros em redes neurais artificiais (RNAs). Em geral, a otimização de hiperparâmetros em modelos geradores e classificadores segue três abordagens principais: ajuste manual, *Grid Search* e *Random Search*. No caso de modelos geradores tabulares, estudos como [Xu et al. 2024] e [Li et al. 2022] recorrem ao ajuste manual para definir parâmetros como número de épocas, taxa de aprendizado e tamanho de *batch*, o que limita a escalabilidade e depende fortemente de tentativa e erro.

Tabela 1. Abordagens de otimização de hiperparâmetros em RNAs.

Referência	Abordagem de Otimização	Intervalos ou Parâmetros Avaliados	Modelos
[Xu et al. 2024]	Ajuste manual	Épocas: 100–5000; *Batch size*: 20–300	*CTGAN*, *TVAE*
[Li et al. 2022]	Ajuste manual	LR: 1×10^{-4} (G) / 3×10^{-4} (D); $\lambda = 1$; *Batch*: 32	*TTS-CGAN*
[Zhou et al. 2020]	*Grid Search*	Dilatação: 8–64; Filtros: 16–256; Kernel/Pool: 2–5; *Dropout*: 10–60%	LSTM-AE, TCN, GAN
[Basri et al. 2023]	*Grid Search*	*Batch*: 50–200; LR (G/D): 10^{-3}–10^{-5}	*CTGAN*
[Nurhayati et al. 2021]	*Grid / Random Search*	Camadas: 3; Unidades: 16–128; Ativação: Sigmoid, ReLU	*GRU*, *LSTM*, *RNN*
Este trabalho	Método proposto	Densidade de camadas: 128–1024; Épocas: 50, 100, 200, 500, 1000	*CGAN*

Trabalhos como [Zhou et al. 2020] e [Basri et al. 2023] aplicam *Grid Search* em arquiteturas recorrentes e GANs, explorando grandes combinações de hiperparâmetros. Embora sistemática, essa abordagem é custosa e cresce rapidamente com o tamanho do espaço de busca. Por outro lado, [Nurhayati et al. 2021] combinam *Grid Search* e *Random Search* em modelos *RNN*, *GRU* e *LSTM*, o que reduz o custo do processo, mas permanece uma abordagem não direcionada, avaliando inúmeras configurações pouco promissoras.

Apesar dos avanços, os trabalhos existentes apresentam limitações importantes. Em geral, não adotam mecanismos capazes de reduzir progressivamente o espaço de busca, nem estratégias que priorizem métricas fundamentais como o *Recall*, essencial em detecção de malware devido ao impacto dos falsos negativos. Além disso, poucos estudos analisam a estabilidade temporal da busca, a convergência ao longo das iterações ou o custo computacional decorrente de reavaliações redundantes.

3. Método Proposto

O método empregado neste trabalho é o *Tune3*, uma abordagem multiestágio para otimização de hiperparâmetros, formalizada originalmente pelos autores sob o nome *HPO3*[3]. Neste artigo, adotamos o nome *Tune3* para representar a versão atual do método. A principal melhoria do *HPO3* para o *Tune3* é que a versão inicial do método dependia fortemente do conhecimento prévio de quem iria executá-lo, e sua primeira etapa exigia um levantamento detalhado da literatura antes da otimização. Com o *Tune3*, o Estágio 1 pode partir de valores já conhecidos e consolidados, ou até de parâmetros que o próprio executor deseja testar diretamente, sem a necessidade de uma fase preliminar de estudo teórico. A implementação do método, na sua forma mais recente, está disponível no repositório GitHub[4]. No presente artigo, utilizamos essa mesma abordagem, apresentando apenas um resumo das etapas principais e sua adaptação ao cenário de detecção de malware Android.

O *Tune3* organiza o processo de otimização em três fases: (i) inicialização informada por evidências, que estabelece um espaço de busca amplo porém plausível; (ii) amostragem de borda com variação controlada, responsável por explorar regiões contrastantes e filtrar execuções instáveis; e (iii) refinamento adaptativo orientado a risco, que concentra a busca nas combinações mais promissoras segundo métricas como *Recall* e *F1-score*.

Nesta aplicação, o método foi ajustado para modelos *CGAN* voltados à síntese de dados Android e ao treinamento subsequente de classificadores. O uso de redes adversariais condicionais foi introduzido para permitir aprendizado guiado por rótulos de classe [Mirza and Osindero 2014]e [Goodfellow et al. 2014], sendo uma estratégia já adotada na síntese de dados tabulares e em detecção de malware [Huang and et al. 2020] e [Nogueira et al. 2024]. As adaptações principais incluem a redução do espaço de busca, a priorização explícita de métricas sensíveis a risco e a incorporação de uma análise detalhada do custo computacional e da convergência temporal, aspectos expandidos na literatura como fatores críticos na otimização e avaliação de modelos generativos [Bischl et al. 2023, Li et al. 2021]. Essas adaptações também se alinham a recomendações de uso de GANs condicionais em fluxos de síntese seguida de classificação por classe, facilitando medir impacto e desempenho do classificador final [Esteban et al. 2017, Li et al. 2021].

4. Tune3: Arquitetura e Implementação

A arquitetura do *Tune3* foi projetada para conduzir ciclos iterativos de otimização de maneira controlada, reprodutível e eficiente. Ela é composta por cinco módulos principais que operam de forma encadeada, permitindo monitoramento contínuo, redução progressiva do espaço de busca e eliminação de execuções redundantes. A seguir, descrevemos seus componentes e o fluxo operacional.

Orquestrador. O orquestrador atua como núcleo do sistema. Suas responsabilidades incluem gerar as combinações iniciais de hiperparâmetros provenientes da Fase 1, coordenar a amostragem de borda (Fase 2) e acionar o refinamento adaptativo (Fase 3). Também

[3]`https://archive.org/details/hpo3-english`
[4]`https://github.com/AILabs4All/Tune3`

aplica filtros de viabilidade (como detecção de *Out of Memory*, instabilidade das redes, estouro de tempo ou baixo desempenho), registra o estado global da busca (execuções válidas, inválidas e descartadas) e define critérios de parada com base em convergência, limite de execuções ou janela temporal.

Executor. Diferentemente de abordagens nas quais o treinamento é executado internamente, o *Tune3* utiliza um executor responsável por invocar ferramentas externas de geração e avaliação por meio de chamadas de linha de comando parametrizadas. Esse módulo constrói comandos incorporando os hiperparâmetros definidos pelo orquestrador, executa o processo em ambiente isolado com controle de tempo, captura logs e saídas padrão, identifica falhas externas (como *timeouts*, erros da ferramenta ou colapso do gerador) e extrai métricas relevantes para posterior consolidação.

Mecanismo de Cache e Persistência. Implementado em `SQLite`, o cache armazena hiperparâmetros, métricas e tempos de execução, funcionando como catálogo histórico que evita reavaliações redundantes. Além disso, fornece ao orquestrador informações fundamentais para priorização de combinações promissoras.

Extrator e Consolidador de Métricas. Após cada execução, as métricas extraídas pelo executor são consolidadas e registradas no banco `SQLite`. Esses valores estruturados alimentam o processo de seleção do *top-k* e orientam a progressão entre as fases da otimização.

5. Avaliação Experimental

Os experimentos compararam o método proposto *Tune3* com o *Random Search*, adotado como *baseline*. Abordagens mais sofisticadas, como Otimização Bayesiana ou Algoritmos Evolutivos, embora consolidadas na literatura, foram excluídas desta etapa inicial por exigirem recursos computacionais muito superiores ao escopo pretendido. Para a avaliação, foram utilizados dois conjuntos amplamente empregados na detecção de malware Android: o *DREBIN-215*, baseado em permissões, e o *Defense-Droid_API_Degree*, baseado em chamadas de API.

Em ambos os casos, as características foram reduzidas para até 200 atributos por meio do teste χ^2, visando padronizar o custo computacional e reduzir ruído estatístico. As execuções foram realizadas em máquinas com configurações semelhantes, assegurando repetibilidade. O espaço de busca adotado por ambos os métodos incluiu a variação do tamanho das camadas densas do gerador e do discriminador entre 128 e 1024 unidades com incremento de 64, além do número de épocas de treinamento nos valores 50, 100, 200, 500 e 1000. Esse espaço totaliza $15 \times 15 \times 5 = 1.125$ configurações possíveis, que constituem o grid completo utilizado como referência na análise comparativa.

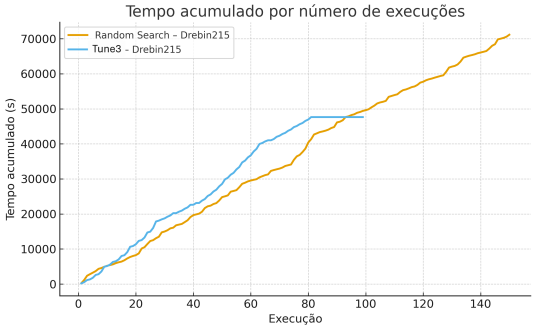
Os demais hiperparâmetros permaneceram fixos durante todos os experimentos. O fator de *dropout* aplicado ao gerador foi definido como 0.5, valor também utilizado para o discriminador. O tamanho do lote de treinamento foi mantido em 128 amostras, a validação cruzada empregou cinco partições, o algoritmo de otimização utilizado foi o Adam e a função de ativação adotada nas camadas ocultas foi a LeakyReLU.

Foram realizadas até 150 execuções por método e por *dataset*, cada uma gerando 10 mil amostras (5 mil benignas e 5 mil maliciosas). As avaliações utilizaram *Recall* como métrica principal, priorizando a detecção de amostras maliciosas, e *F1-score* como

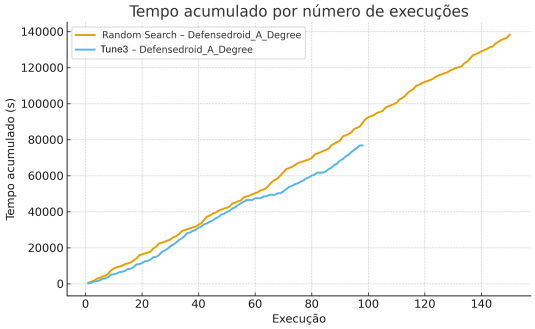
métrica secundária para avaliar o equilíbrio entre precisão e sensibilidade. Essas métricas refletem a natureza orientada a risco do domínio, no qual falsos negativos são particularmente críticos por representarem ameaças não detectadas.

5.1. Resultados Comparativos

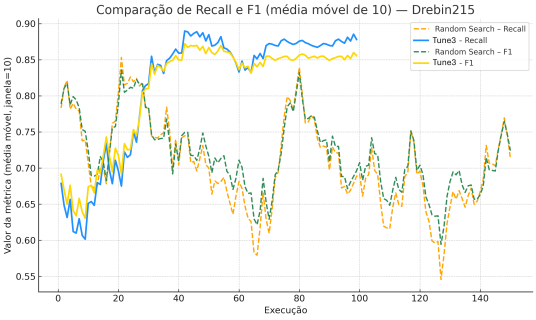
Os resultados desta seção comparam o desempenho do *Tune3* e do *Random Search* nos conjuntos *DREBIN-215* e *DefenseDroid_API_Degree*. O objetivo é avaliar se o *Tune3* reduz o custo computacional e acelera a convergência sem perda de desempenho.



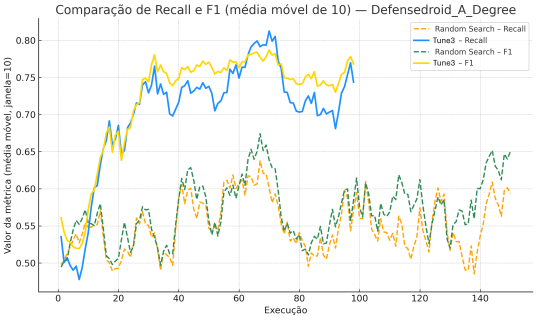
(a) Tempo acumulado — *DREBIN-215*



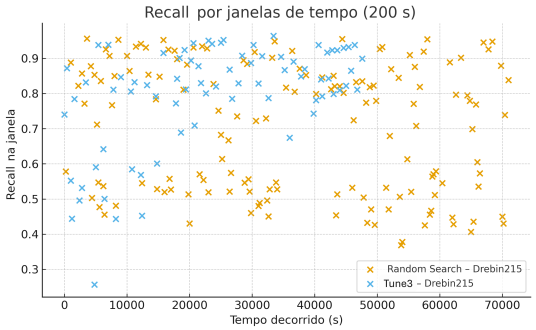
(b) Tempo acumulado — *DefenseDroid_API_Degree*



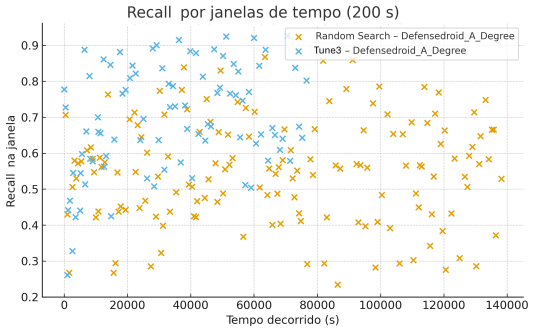
(c) Tendência do *Recall* e *F1* — *DREBIN-215*



(d) Tendência do *Recall* e *F1* — *DefenseDroid_API_Degree*



(e) *Recall* por janelas de 200 s — *DREBIN-215*



(f) *Recall* por janelas de 200 s — *DefenseDroid_API_Degree*

Figura 1. Comparação entre *Tune3* e *Random Search* em termos de custo acumulado, tendência das métricas e estabilidade temporal do *Recall*.

A Figura 1 reúne os resultados comparativos. As subfiguras (a) e (b) apresentam o tempo acumulado, enquanto (c) e (d) mostram a tendência do *Recall* e do *F1-score* por média móvel (janela = 10). Por fim, (e) e (f) ilustram o *Recall* médio por janelas de 200 s, destacando a estabilidade temporal de cada método.

No *DREBIN-215*, o *Tune3* alcançou *Recall* máximo de aproximadamente 0,96, desempenho semelhante ao do *Random Search* (0,95), porém com custo total cerca de 34% menor, como visto na subfigura (a). O *Tune3* atingiu níveis elevados de desempenho em torno da 40ª execução, enquanto o *Random Search*, por não direcionar a busca, obteve bons resultados apenas de forma pontual e manteve comportamento mais irregular. A tendência observada na subfigura (c) mostra que o *Tune3* estabiliza após aproximadamente 30 execuções, ao passo que o *Random Search* oscila entre 0,55 e 0,86 ao longo de toda a busca. A análise temporal na subfigura (e) confirma maior consistência do *Tune3*, que manteve *Recall* médio superior na maior parte das janelas de 200 s.

No *DefenseDroid_API_Degree*, os ganhos foram mais evidentes: o *Tune3* atingiu *Recall* final de aproximadamente 0,92, contra 0,87 do *Random Search*. O tempo acumulado foi reduzido em cerca de 44% de acordo com a subfigura (b). A média móvel apresentada na subfigura (d) indica convergência entre a 25ª e a 35ª execuções, enquanto o *Random Search* permaneceu irregular durante toda a busca. As janelas temporais da subfigura (f) mostram que o *Tune3* manteve *Recall* médio entre 0,70 e 0,80, ao passo que o *Random Search* permaneceu concentrado entre 0,50 e 0,60.

De forma geral, os resultados indicam que o *Tune3* preserva o desempenho global (*Recall* e *F1-score*) ao mesmo tempo em que reduz significativamente o custo computacional. No *DREBIN-215*, o ganho se manifesta principalmente em menor tempo e maior estabilidade; no *DefenseDroid_API_Degree*, observa-se melhora simultânea de qualidade e eficiência. Em ambos os casos, o *Tune3* atinge rapidamente regiões promissoras do espaço de busca e mantém resultados estáveis entre execuções, contrastando com o comportamento mais errático do *Random Search*.

6. Conclusão

Este trabalho apresentou o *Tune3*, um método multiestágio adaptativo para otimização de hiperparâmetros aplicado à detecção de malware Android. Nos experimentos com os conjuntos *DREBIN-215* e *DefenseDroid_API_Degree*, o método obteve desempenho equivalente ou superior ao *Random Search*, com reduções de 34% a 44% no tempo acumulado e menor variabilidade entre execuções. Embora o *Random Search* possa produzir bons resultados pontuais em espaços de busca pequenos, sua eficácia diminui à medida que o espaço cresce, enquanto o *Tune3* se beneficia de uma busca estruturada que concentra esforços em regiões mais promissoras. Os resultados indicam que o *Tune3* é adequado a cenários de detecção de ameaças, nos quais tempo, estabilidade e sensibilidade a falsos negativos são fatores críticos. Como trabalhos futuros, propõe-se a inclusão de métricas de custo energético e a avaliação do método em contextos de aprendizado federado, ampliando sua aplicabilidade a ambientes distribuídos e restritos em recursos.

Agradecimentos. Esta pesquisa recebeu apoio parcial da CAPES[5], sob o Código de Financiamento 001, e da FAPERGS[6], por meio dos termos de outorga 24/2551-0001368-

[5] `https://www.gov.br/capes/pt-br`

[6] `https://fapergs.rs.gov.br`

7, 24/2551-0000726-1 e 22/2551-0000841-0.

Referências

Basri, A., Hassan, A., and Yusof, N. (2023). A hyperparameter tuning framework for tabular synthetic data generation methods. *JCVIS*, 5(1).

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The journal of machine learning research*, 13(1).

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., et al. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIRE*, 13(2).

Esteban, C., Hyland, S. L., and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Huang, W. and et al. (2020). Malware detection using deep learning techniques: A survey. *ACM Computing Surveys*, 53(4):1–36.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2021). A survey of hyperparameter optimization in deep learning. *IEEE Access*, 9:125676–125694.

Li, Q., Zhang, Y., and Wang, J. (2022). Tts-cgan: Time-series conditional generative adversarial network for tabular data synthesis. *arXiv preprint arXiv:2206.13676*.

Liao, L., Li, H., Shang, W., and Ma, L. (2022). An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM TOSEM*, 31(3).

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024). MalSyn-Gen: redes neurais artificiais na geração de dados tabulares sintéticos para detecção de malware. In *Anais do XXIV SBSeg*. SBC.

Nurhayati, R., Sari, A., and Pratama, M. (2021). Bitcoin prediction model using deep learning with hyperparameter optimization. In *2021 9th ICIM*.

Ozkan-Okay, M., Akin, E., Aslan, Ö., Kosunalp, S., Iliev, T., Stoyanov, I., and Beloev, I. (2024). A comprehensive survey: Evaluating the efficiency of artificial intelligence and machine learning techniques on cyber security solutions. *IEEe Access*, 12.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2024). Rigorous experimental analysis of tabular data generated using ctgan and tvae. *IJACSA*, 15(4).

Zhou, K., Zhang, Y., and Wu, X. (2020). Time series forecasting and classification models using deep learning and attention mechanisms. *Sensors*, 20(24):7211.