SigAPI AutoCraft: uma ferramenta de seleção de características com capacidade de generalização

Laura C. Tschiedel[1], Vanderson Rocha[2], Diego Kreutz[1]
Hendrio Bragança[2], Silvio E. Quincozes[1], Angelo G. D. Nogueira[1], Joner Assolin[2]

[1]LEA, PPGES, Universidade Federal do Pampa (UNIPAMPA) – Alegrete, Brasil

[2]IComp, Universidade Federal do Amazonas (UFAM) – Manaus, Brasil

Resumo. *Neste trabalho apresentamos a SigAPI AutoCraft, uma ferramenta de seleção de características com generalização nativamente incorporada. A partir de uma avaliação empírica extensa, utilizando dez datasets distintos e heterogêneos, podemos concluir que a SigAPI AutoCraft possui uma capacidade de generalização superior à maioria dos métodos sofisticados de seleção de características no contexto de malwares Android.*

1. Introdução

A seleção de características, uma etapa essencial e anterior ao treinamento de modelos de aprendizado de máquina (ML), pode aprimorar a performance e a precisão dos modelos, eliminando características irrelevantes e reduzindo a dimensionalidade do conjunto de dados [Venkatesh and Anuradha, 2019, Golrang et al., 2021]. Isso não apenas otimiza o uso de recursos computacionais, reduzindo o consumo de memória e o tempo de processamento, mas também diminui o tempo necessário para o treinamento e melhora a capacidade de generalização das classificações realizadas. Ademais, a seleção criteriosa das características facilita a interpretação dos modelos, tornando mais fácil a identificação de padrões significativos nos conjuntos de dados [Wei et al., 2020].

Estudos recentes indicam que a maioria dos métodos sofisticados de seleção de características do contexto de detecção de *malware* Android é definitivamente incapaz de apresentar uma generalização mínima para diferentes conjuntos de dados [Neves et al., 2023, Soares et al., 2022]. Por exemplo, métodos como JOWMDroid, RFG e FSDroid apresentam taxas de desempenho significativamente inferiores ao *baseline* (*dataset* completo) para três ou mais conjuntos de dados utilizados na avaliação [Neves et al., 2023]. Similarmente, o SigAPI, apresentando em [Galib, A. H. et. al., 2020], também apresenta uma falta de capacidade de generalização para diferentes *datasets* [Soares et al., 2022, Costa, E. et. al., 2022].

No contexto de detecção de *malware* Android, as chamadas de API e permissões estão entre as características mais frequentemente utilizadas [Soi D. et. al., 2024, Qiu, J. et. al., 2023, Maniriho P. et. al., 2023]. As chamadas de API são importantes porque permitem entender qual a funcionalidade da aplicação ao avaliar o tipo de interações que ocorrem no sistema Android, possibilitando identificar se é uma aplicação maliciosa ou não a partir do seu comportamento [Alazab M. et. al., 2020]. Por outro lado, as permissões são responsáveis por formar uma camada de proteção às chamadas de API. Neste contexto, métodos como o SigAPI [Galib, A. H. et. al., 2020] objetivam otimizar a seleção de características como chamadas de API. Entretanto, o desenvolvimento desses

métodos não leva em consideração a capacidade de generalização e aplicação para além de um ou dois conjuntos de dados utilizados nos trabalhos originais.

Neste trabalho, propomos a ferramenta SigAPI AutoCraft, que consiste na reprodução, evolução e disponibilização de uma versão pública do SigAPI. O SigAPI AutoCraft automatiza e generaliza as condições de parada do método original, levando a uma capacidade de generalização para diferentes conjuntos de dados. Para demonstrar a eficácia do novo método, utilizamos dez *datasets* distintos. Os resultados indicaram uma melhora de até 20% em relação ao método original, com uma excelente capacidade de generalização.

2. SigAPI AutoCraft: Fluxo de Execução e Implementação

O SigAPI AutoCraft é uma reprodução e evolução do SigAPI, um método para seleção de características composto por três etapas principais, conforme ilustrado na Figura 1: **i)** seleção incremental de características, utilizando seis técnicas de ranqueamento; **ii)** seleção da melhor técnica de pré-seleção; e **iii)** eliminação de características irrelevantes para a predição. Subsequentemente às três etapas, o *dataset* resultante (reduzido) é avaliado utilizado método de aprendizado de máquina *Random Forest*.

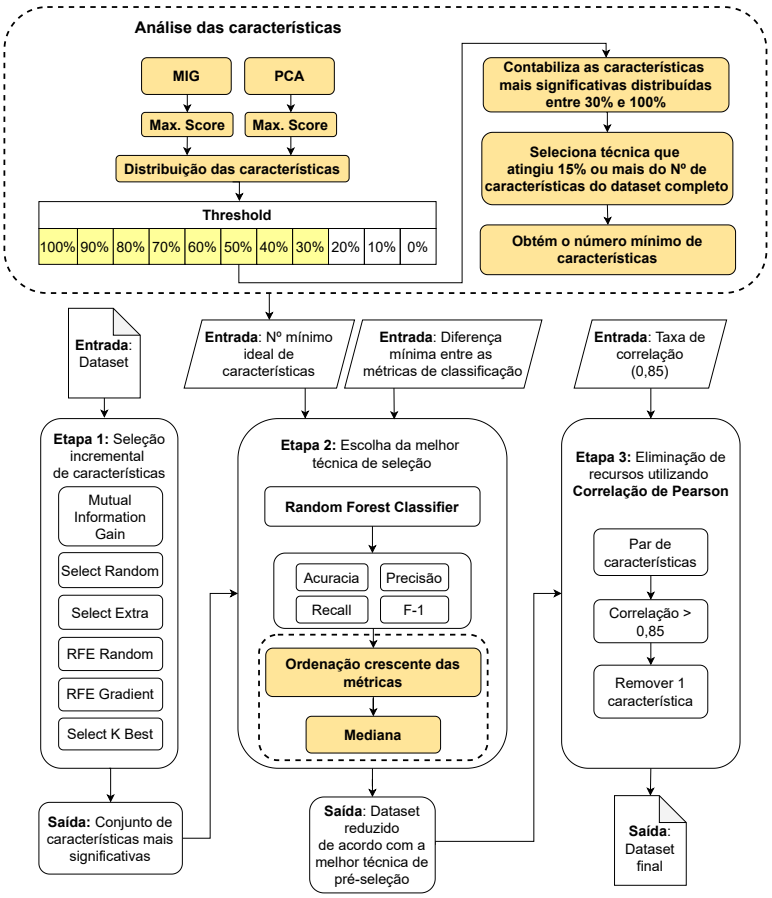


Figura 1. SigAPI AutoCraft: as linhas tracejadas indicam os novos recursos incorporados ao SigAPI original.

Na **primeira etapa**, o método utiliza seis técnicas distintas para identificar as características mais significativas, como chamadas de API, na detecção de *malware* Android a partir do *dataset* completo. O usuário identifica a melhor técnica de pré-seleção e o número ideal de características analisando gráficos das métricas para diferentes quantidades de características avaliadas.

Na **segunda etapa**, ocorre a escolha da melhor técnica de pré-seleção, utilizando os conjuntos de dados reduzidos, os quais foram obtidos com as seis técnicas avaliadas na etapa anterior, e o número mínimo de características. Adicionalmente, uma condição de parada é definida pela *taxa de estabilidade desejável* das métricas. O método procura identificar o conjunto de dados que obtém o melhor resultado de predição como o menor número (subconjunto) de características, visando maximizar a redução do conjunto de dados sem reduzir as taxas de predição. Para identificar autonomamente o número mínimo de características, incorporamos um módulo denominado *Análise das Características*.

No módulo de *Análise das Características*, utilizamos as técnicas de *Mutual Information Gain* (MIG) e *Principal Component Analysis* (PCA)[1] para implementar uma condição de parada dinâmica e automática, independente do *dataset* de entrada. Isso implica na determinação do número mínimo de características necessário para cada *dataset* de maneira automatizada.

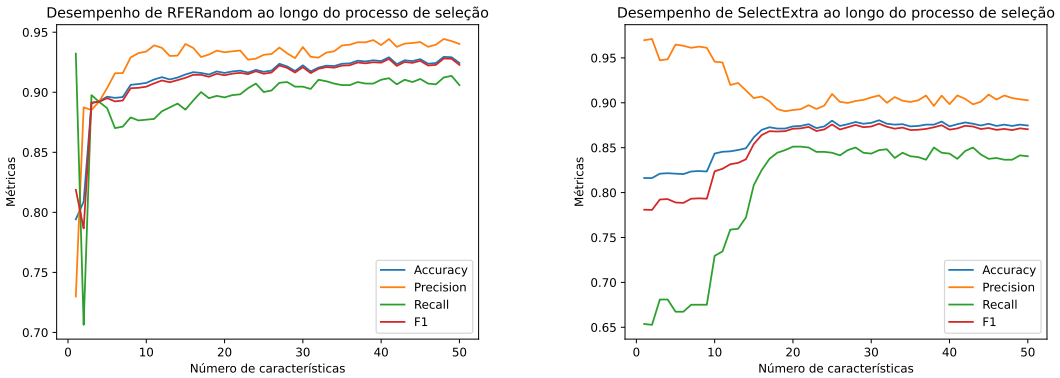
Utilizamos o PCA e o MIG para identificar a importância relativa das características e atribuir uma pontuação a cada uma delas. A pontuação resultante é utilizada para classificar as características por sua taxa de relevância e, assim, contabilizá-las. A técnica selecionada (PCA ou MIG), aquela que apresentar os melhores resultados, deve agrupar um número de características que corresponda a pelo menos 15% do total *dataset* completo, garantindo que as características mais relevantes não sejam desconsideradas.

Experimentos empíricos extensivos indicam que um *threshold* de 30% (ou seja, características entre 30% e 100%) para o PCA ou MIG é ideal para a maioria dos *datasets*. Utilizando este *threshold* e analisando as métricas nos gráficos das métricas resultantes para todos os métodos de pré-seleção, como ilustrado por dois gráficos da Figura 2, identificamos um padrão na distribuição das características por taxa de relevância. As análises indicam que o *threshold* de 30% agrupa as características mais relevantes e define um padrão para o número mínimo de características.

Observamos, ao analisar gráficos de mais de dez conjuntos de dados distintos, que a taxa de crescimento das métricas tende a diminuir, estabilizando-se e apresentando bons resultados a partir de 15% das características dos *datasets*. Com base nessa observação, utilizamos 15% das características do conjunto de dados como um número mínimo dinâmico em casos excepcionais, onde PCA e MIG não atingem 15% do tamanho do *dataset* com *thresholds* de 30% a 100%. Nesses casos, 15% do tamanho do *dataset* completo é utilizado.

Adicionalmente, verificamos que, enquanto em alguns *datasets* o MIG mostrou-se eficaz ao selecionar um número ideal de características, em outros, a seleção foi insuficiente, ao passo que o PCA identificou um conjunto mais adequado. Portanto, na presente proposta, a técnica que não atender aos requisitos estabelecidos é desconsiderada.

[1]É importante destacarmos que o PCA tem sido utilizado amplamente como uma técnica de seleção de características, sendo eficaz na identificação das características com maior relevância presentes no *dataset*.



(a) Defensedroid Katz. (b) Adroit.

Figura 2. Métricas para os *datasets* Defensedroid Katz e Adroit.

Além de incorporar uma abordagem dinâmica e automática para o parâmetro do número mínimo de características, propomos uma nova metodologia para a escolha da técnica de pré-seleção. Originalmente, o método considerava a acurácia como a única métrica para definir a melhor técnica de pré-seleção. No entanto, ao desconsiderar as demais métricas, essa escolha pode introduzir um viés negativo.

Por exemplo, se outras métricas, como *recall* e pontuação F1, apresentarem valores abaixo do esperado em comparação com as métricas do *dataset* completo, o resultado de predição do *dataset* reduzido pode ser prejudicado. Para mitigar esse problema, no SigAPI AutoCraft, utilizamos todas as métricas (acurácia, precisão, *recall* e pontuação F1) e o cálculo da mediana para definir a técnica de pré-seleção. A técnica que apresentar a maior mediana é selecionada, com o objetivo de aprimorar o resultado da predição do *dataset* reduzido gerado pelo processo do método.

Finalmente, na **terceira etapa** ocorre a eliminação de características utilizando o coeficiente de correlação de *Pearson* [Cohen, I. et. al., 2009]. Nesta etapa, pares de características com correlação maior que 0,85 (valor estipulado pelos autores do método) são encaminhados para eliminação, com base em sua importância relativa determinada pelo *Random Forest*. Em cada par de correlação identificado, a característica menos importante é eliminada, minimizando o número de chamadas de API sem afetar o desempenho da classificação.

3. Resultados Experimentais

Nesta seção apresentamos os *datasets* utilizados (Seção 3.1), o ambiente de testes (Seção 3.2) e os resultados obtidos após a otimização do método SigAPI (Seção 3.4).

3.1. *Datasets*

Na Tabela 1 apresentamos os detalhes dos dez *datasets* utilizados nos experimentos. A tabela apresenta as características e os números de amostras dos conjuntos de dados originais (Amostras Originais) e balanceados (Amostras Balanceadas). O processo de limpeza, normalização e redução de dimensionalidade (e.g., balanceamento e remoção preliminar de características) foi necessário para eliminar impurezas (e.g., amostras incompletas

ou duplicadas), características irrelevantes ou ainda resolver o problema de desbalanceamento significativo entre as classes. Os detalhes e as referências dos *datasets* originais e balanceados, bem como o código do processo de limpeza, podem ser encontrados no repositório GitHub da SigAPI AutoCraft[2]

Tabela 1. Conjuntos de dados utilizados nos experimentos.

Datasets		**Amostras Balanceadas**			**Amostras Originais**		
Datasets	**Cars.**	**Ben.**	**Mal.**	**Total**	**Ben.**	**Mal.**	**Total**
Androcrawl	81: P,A	10.170	10.170	20.340	86.562	10.170	96.732
Drebin-215	215: P,A	5.555	5.555	11.110	9.476	5.555	15.031
Adroit	166: P	3.418	3.418	6.836	8.058	3.418	11.476
Android Permissions	151: P	9.077	9.077	18.154	9.077	17.787	26.864
Kronodroid Real Device	286: P,A	36.755	36.755	73.510	36.755	41.382	78.137
Defensedroid Katz	200: A	5.222	5.222	10.444	5.254	5.222	10.476
Defensedroid Degree	200: A	5.222	5.222	10.444	5.254	5.222	10.476
Defensedroid Closeness	200: A	5.222	5.222	10.444	5.222	5.254	10.476
Defensedroid PRS	200: A	5.975	5.975	11.950	11.975	5.975	17.950
MH-100K	200: P,A	10.000	10.000	20.000	89.213	12.721	101.934

O balanceamento das amostras foi realizado com o uso do método *Random Under Sampler* disponível na biblioteca `imblearn` do Python. Já para a remoção preliminar de características utilizamos o *score* do qui-quadrado da biblioteca `scikit-learn`, selecionando as 200 características com os maiores valores. O método *Random Under Sampler* equilibra o número de amostras removendo aleatoriamente amostras do conjunto majoritário para que ambas as classes tenham a mesma proporção de amostras benignas e malignas. Para o *dataset* MH-100K, o balanceamento foi realizado utilizando a função `resample`, também da biblioteca `scikit-learn`, pois foram reduzidas as amostras de todas as classes devido ao enorme tamanho (mais de 100k amostras e 24k características) do conjunto de dados original.

3.2. Ambiente

Os experimentos foram executados em um computador com processador Intel Core i5-8265U CPU @ 1.60GHz (8 núcleos) e 8GB de memória RAM, rodando o sistema operacional Ubuntu 22.04.4 LTS com Kernel 6.5.0-35-generic. A linguagem de programação utilizada foi Python na versão 3.10.12, juntamente com as bibliotecas `scikit-learn` (versão 1.4.0), `pandas` (versão 2.2.0), `numpy` (versão 1.26.3), `joblib` (versão 1.3.2), `matplotlib` (versão 3.5.1) e `imblearn` (versão 0.0). A implementação completa do SigAPI AutoCraft e os extensos resultados dos experimentos estão disponíveis no GitHub [Tschiedel et al., 2024].

3.3. Utilização

Os parâmetros utilizados para a execução do método são: `-m`, que indica o módulo Python do método, `-d`, que indica o *dataset* a ser processado, `-o`, que indica o nome do arquivo de saída que irá armazenar o *dataset* reduzido gerado pelo método e `-i`, que indica o incremento a ser utilizado no módulo de *Análise das Características*. Mais detalhes sobre os parâmetros e exemplos de utilização podem ser visto no GitHub da ferramenta [Tschiedel et al., 2024].

[2]`https://github.com/SBSegSF24/SigAPI-AutoCraft`

3.4. Resultados

A Tabela 2 apresenta a comparação dos resultados de predição (RoC_AuC) e da porcentagem de redução em relação ao *dataset* original (completo) obtidos pelo modelo treinado com o conjunto de dados completo, com os conjuntos de dados gerados pelo SigAPI e pelo SigAPI AutoCraft. A avaliação das métricas foi realizada utilizando o *Random Forest*, o mesmo modelo empregado no SigAPI original [Galib, A. H. et. al., 2020].

Tabela 2. SigAPI AutoCraft *versus* SigAPI nos dez *datasets* avaliados.

Datasets	Dataset Completo		SigAPI Original		SigAPI AutoCraft	
	Caracts.	RoC	Redução	RoC	Redução	RoC
Androcrawl	81	97,4%	97,5%	82,8%	76,6%	95,6%
Drebin-215	215	99,8%	99,5%	78,5%	85,2%	99,4%
Adroit	166	91,3%	99,3%	82,4%	89,8%	90,1%
Android Permissions	151	70,8%	98,6%	58,8%	76,9%	69,9%
Kronodroid Real Device	286	99,7%	99,3%	82,6%	87,5%	99,1%
Defensedroid Katz	200	98,8%	99,0%	79,5%	88,0%	96,1%
Defensedroid Degree	200	98,8%	99,0%	79,5%	88,5%	95,3%
Defensedroid Closeness	200	99,1%	99,0%	78,8%	88,0%	95,5%
Defensedroid PRS	200	96,9%	99,0%	78,1%	86,0%	95,5%
MH-100K	200	96,6%	99,5%	95,4%	97,5%	95,4%

No SigAPI AutoCraft, a redução do número de características no *dataset* é realizada de forma mais gradual em comparação com a versão tradicional do SigAPI, o que melhora a capacidade de predição do *dataset* reduzido. Como consequência, o SigAPI AutoCraft descartou apenas 87,5% das características originais, o que possibilitou ao método manter uma métrica RoC_AuC acima de 99%. Já o SigAPI reduziu o *dataset Kronodroid Real Device* em 99,3%, resultando em uma predição de apenas 82,6%, ou seja, mais de 17% inferior à do SigAPI AutoCraft.

Como podemos observar, os resultados do SigAPI AutoCraft são promissores para todos os conjuntos de dados avaliados. A taxa de predição foi muito próxima àquela obtida com o *dataset* completo e a porcentagem de redução permaneceu acima de 75% para todos os conjuntos de dados avaliados. Diferentemente do SigAPI original, os resultados do SigAPI AutoCraft indicam uma ótima capacidade de generalização, isto é, obter bons resultados para *datasets* quaisquer.

É importante destacarmos também a importância de comparar os resultados do SigAPI AutoCraft com outros métodos sofisticados de seleção de características para detecção de *malwares* Android, como JOWMDroid, RFG, SigPID, FSDroid, SemiDroid e MT. Esses métodos fizeram parte de estudos recentes [Costa, E. et. al., 2022, Neves et al., 2023], onde foram investigadas as capacidades de generalização e qualidade dos resultados para diferentes conjuntos de dados.

Ao observarmos os resultados, podemos afirmar que SigAPI AutoCraft apresenta uma redução de dimensionalidade significativa, assim como outros métodos, porém apresenta uma estabilidade significativamente maior em relação a capacidade de predição para os mais diversos *datasets*. Exemplos notáveis incluem os *datasets* Kronodroid Real Device e Drebin-215, utilizados em ambos os estudos. O SigAPI AutoCraft demonstrou uma capacidade preditiva de 99,4% para o Drebin-215 e 99,1% para o Kronodroid Real

Device, valores semelhantes aos obtidos com os conjuntos de dados completos. Essas taxas foram alcançadas mantendo uma taxa de redução de dimensionalidade superior a 85%.

Observando os resultados dos métodos apresentados em [Costa, E. et. al., 2022, Neves et al., 2023], constatamos que o SigAPI AutoCraft supera os resultados desses métodos para *datasets* como Androcrawl, Android Permissions, Defensedroid PRS, Drebin-215 e Kronodroid Real Device, demonstrando sua maior capacidade de generalização. Por exemplo, no caso do JOWMDroid, a capacidade de predição do *dataset* reduzido Kronodroid Real Device diminuiu em 43,5%, apesar de considerar um número maior de características, isto é, uma redução de apenas 77,3%. No caso do método SemiDroid houve uma redução no tamanho do *dataset* de 89,5%, levando a um número menor de características. Apesar disso, o SemiDroid, assim como o SigAPI AutoCraft, manteve uma capacidade de predição de 95,6%, muito próxima do *dataset* completo. Isto significa que métodos como o SemiDroid e SigAPI AutoCraft são capazes de seletivamente escolher as características que mais influenciam os modelos.

4. Conclusão e Demonstração

Conclusão. O SigAPI é um método sofisticado de seleção de características para o domínio de detecção de *malwares* Android. Entretanto, a implementação do SigAPI não está disponível e o método apresenta problemas de usabilidade, eficiência e capacidade de generalização. O SigAPI AutoCraft apresenta uma reprodução do SigAPI e endereça os demais problemas através de uma interface simples e técnicas elaboradas para a seleção automática do número mínimo de características e de identificação dos pontos de estabilidade das métricas que sejam próximos aos resultados ideais (e.g., resultados com o *dataset* completo).

Os resultados indicam que o SigAPI AutoCraft consegue atingir altas taxas de redução (acima de 75%) sem comprometer o desempenho final, isto é, métricas resultantes muito boas. Utilizando dez *datasets* significativamente heterogêneos, conseguimos demonstrar que o SigAPI AutoCraft apresenta uma excelente capacidade de generalização, ou seja, consegue manter altas taxas de redução e métricas muito boas para quaisquer conjuntos de dados, algo que é bastante incomum para a maioria dos métodos sofisticados de seleção de características, como demonstrado em pesquisas recentes [Soares et al., 2022, Neves et al., 2023].

É importante destacar também que a melhoria apresentada pelo SigAPI AutoCraft tem impacto direto no uso de recursos computacionais. A redução significativa da quantidade de características necessárias para atingir bons resultados leva a uma redução substancial no consumo de memória e processamento geral do *pipeline* em produção, além de reduzir também o tempo de treinamento e aumentar a capacidade de generalização do modelo de classificação. Outro benefício da seleção eficiente de características é a melhora na interpretação do modelo, que deixa mais intuitivo a identificação de padrões nos conjuntos reduzidos de características.

Como trabalhos futuros podemos destacar: (a) avaliar o SigAPI AutoCraft em *datasets* com um número substancial de características e amostras; (b) avaliar as técnicas de pré-seleção utilizadas no método para verificar se há possibilidade de otimizá-las, buscando reduzir o tempo de processamento dos *datasets*; e (c) otimizações gerais para in-

corporar o método a *pipelines* de detecção de *malware* em tempo real e em contexto com limitação de recursos computacionais (e.g., IoT).

Demonstração. A ferramenta será apresentada em dispositivo próprio dos autores. Para a demonstração de funcionamento da SigAPI AutoCraft, utilizaremos os seguintes passos: (i) apresentação das funcionalidades e parâmetros de execução; (ii) apresentação da execução e resultados para diferentes *datasets*.

Agradecimentos. Este trabalho foi parcialmente financiado pela CAPES – Código de Financiamento 001, e pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), por meio dos editais 08/2023 e 09/2023.

Referências

Alazab M. et. al. (2020). Intelligent mobile malware detection using permission requests and api calls. *Future Generation Computer Systems*, 107:509–521.

Cohen, I. et. al. (2009). Pearson correlation coefficient. *Noise reduction in speech processing*.

Costa, E. et. al. (2022). FS3E: uma ferramenta para execução e avaliação de métodos de seleção de características para detecção de malwares android. In *XXII SBSeg*. SBC.

Galib, A. H. et. al. (2020). Significant API calls in Android malware detection (using feature selection techniques and correlation based feature elimination). In *The 32nd SEKE*.

Golrang, A., Yayilgan, S. Y., and Elezaj, O. (2021). The multi-objective feature selection in Android malware detection system. In *Intelligent Tech. and Applications*, page 311.

Maniriho P. et. al. (2023). API-MalDetect: Automated malware detection framework for windows based on api calls and deep learning techniques. *JNCA*, 218:103704.

Neves, N., Rocha, V., Kreutz, D., Bragança, H., and Feitosa, E. (2023). Avaliação de métodos de seleção de características de amostras android com a ferramenta FS3E (v2). In *Anais da XX ERRC*. SBC.

Qiu, J. et. al. (2023). Cyber code intelligence for android malware detection. *IEEE Transactions on Cybernetics*, 53(1):617–627.

Soares, T., Kreutz, D., Rocha, V., Costa, E., Leão, L., Pontes, J., Assolin, J., Rodrigues, G., and Feitosa, E. (2022). Uma análise de métodos de seleção de características aplicados à detecção de malwares android. In *Anais do XXII SBSeg*. SBC.

Soi D. et. al. (2024). Enhancing android malware detection explainability through function call graph apis. *Journal of Information Security and Applications*, 80:103691.

Tschiedel, L. C. et al. (2024). GitHub: SigAPI AutoCraft. `https://github.com/SBSegSF24/SigAPI-AutoCraft`.

Venkatesh, B. and Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies*, 19(1):3–26.

Wei, G., Zhao, J., Feng, Y., He, A., and Yu, J. (2020). A novel hybrid feature selection method based on dynamic feature importance. *Applied Soft Computing*, 93:106337.