Análise Comportamental com Aprendizado de Máquina para Detecção de Atividades Suspeitas em Sistemas Corporativos

Kayo Galdino Gomes Rocha

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2026

Kayo Galdino Gomes Rocha

Análise Comportamental com Aprendizado de Máquina para Detecção de Atividades Suspeitas em Sistemas Corporativos

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Prof. Dr. Adriano Mendonça Rocha

Monte Carmelo - MG
2026

Dedico este trabalho aos meus pais,
que sob muito sol fizeram-me chegar
até aqui, na sombra.

Agradecimentos

Agradeço a Deus, por me guiar até aqui. Aos meus pais, por todo sacrifício e apoio incondicional. Ao meu sócio Felipe Oliveira, pela parceria e confiança ao longo dessa jornada. Ao meu orientador professor Adriano e ao meu professor Silvio, pela paciência e direcionamento ao longo deste trabalho. E a todos que, de alguma forma, contribuíram para a minha formação.

"SUA VIDA PODE SER DIVIDIDA EM DUAS FASES...
ANDANDO NO SEU CAMINHO OU ANDANDO NO CAMINHO DE DEUS "
(Pastor Giovani Gomes - baseado na leitura bíblica de Isaias 55:8)

Resumo

Este trabalho propõe e implementa um sistema de análise comportamental com Aprendizado de Máquina para detecção de atividades suspeitas em um software de gerenciamento de corretagem de café. O objetivo é verificar se modelos de classificação supervisionada, implementados com TensorFlow.js, superam métodos tradicionais baseados em regras estáticas na identificação de comportamentos potencialmente maliciosos dos usuários. Para isso, foi desenvolvido um sistema composto por duas camadas: um *backend* em Node.js com Express e SQLite, responsável pelo treinamento do modelo, e um *frontend* em React com TypeScript, responsável pela inferência em tempo real no navegador. O modelo utiliza uma rede neural densa com 4.993 parâmetros, treinada sobre um conjunto de 10.000 amostras sintéticas distribuídas em oito perfis comportamentais, a partir de 30 *features* numéricas organizadas em três categorias: ambiente, comportamento e contexto. O sistema incorpora um mecanismo de *Human-in-the-Loop* que permite ao administrador revisar incidentes, confirmar ou descartar ameaças e alimentar o re-treinamento incremental do modelo. Nos experimentos, o modelo alcançou F1-Score de 99,68% contra 82,16% do *baseline* de regras estáticas, o que corresponde a uma melhoria relativa de 21,3% e a uma diferença absoluta de 17,52 pontos percentuais. A diferença mais expressiva foi observada na revocação: o modelo detectou 99,48% das ações suspeitas, enquanto as regras detectaram apenas 71,98%, deixando escapar aproximadamente 28% das ameaças. A matriz de confusão do modelo apresentou apenas 4 falsos negativos em 1.500 amostras de teste, contra 216 do *baseline*. Os resultados permitem rejeitar a hipótese nula de equivalência entre as duas abordagens no cenário avaliado, com duas ressalvas: os dados são sintéticos e separáveis por construção, e o *baseline* foi definido no próprio trabalho.

Palavras-chave: Aprendizado de Máquina, Análise Comportamental, Classificação Supervisionada, Segurança de Sistemas de Informação, Detecção de Anomalias.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
AUC-ROC	*Area Under the Receiver Operating Characteristic Curve*
CMU-CERT	*Carnegie Mellon University Computer Emergency Response Team*
CRUD	*Create, Read, Update, Delete*
ERP	*Enterprise Resource Planning*
FN	Falso Negativo
FP	Falso Positivo
FPR	*False Positive Rate*
HDFS	*Hadoop Distributed File System*
HITL	*Human-in-the-Loop*
IA	Inteligência Artificial
IP	*Internet Protocol*
LDAP	*Lightweight Directory Access Protocol*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MUI	*Material UI*
PCA	*Principal Component Analysis*
REST	*Representational State Transfer*
SHAP	*SHapley Additive exPlanations*

SMOTE	*Synthetic Minority Over-sampling Technique*
SSO	*Single Sign-On*
SVM	*Support Vector Machine*
TPR	*True Positive Rate*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

Sumário

Capítulo 1

Introdução

A corretagem de café intermedia produtores e compradores e tem papel central no funcionamento do mercado cafeeiro. Com o avanço das tecnologias da informação, surgiram softwares de gerenciamento, como os *Enterprise Resource Planning* (ERP) personalizados, para facilitar e otimizar esse processo. Esse volume crescente de dados gerados e armazenados, porém, traz desafios de segurança e de uso adequado da informação. Sistemas corporativos concentram informações sensíveis e ficam expostos tanto a agentes externos quanto a usuários internos que abusam das próprias permissões. Homoliak et al. (2019) organizam esse segundo cenário em uma taxonomia de ameaças internas, e Liu et al. (2018) revisam as técnicas disponíveis para detectá-las e preveni-las.

1.1 Motivação

Este trabalho mostra como a análise comportamental com Aprendizado de Máquina (ML, do inglês *Machine Learning*) pode ser aplicada ao software de gerenciamento da corretagem de café para monitorar e identificar atividades indevidas ou errôneas dos usuários. A partir dos logs de eventos que registram as ações no sistema, é possível perceber padrões de comportamento criados para copiar dados de clientes ou para realizar consultas em tempo muito menor do que o esperado em operações regulares. A literatura documenta a viabilidade dessa leitura em diferentes fontes de registro: Du et al. (2017) extraem anomalias de logs de sistema com redes recorrentes, Berlin, Slater e Saxe (2015) detectam comportamento malicioso em logs de auditoria do Windows e Ranjan e Kumar (2022) separam usuários legítimos de mal-intencionados a partir de padrões de navegação em logs de aplicação.

Essa aplicação permite criar modelos que descrevem o comportamento dos usuários e funcionam como garantia de segurança e confidencialidade das informações, protegendo os interesses de produtores, compradores e da própria corretora. A abordagem também ajuda a identificar indícios de fraude e a fundamentar ações preventivas contra prejuízos financeiros e danos à imagem da empresa. A detecção de anomalias é o alicerce

técnico desse uso: Chandola, Banerjee e Kumar (2009) sistematizam as técnicas disponíveis e mostram que um mesmo comportamento pode ser normal ou anômalo conforme o contexto, enquanto Buczak e Guven (2016) organizam a aplicação dessas técnicas em segurança cibernética. Nesse sentido, a análise comportamental com ML se apresenta como ferramenta de governança corporativa e de gestão de riscos na corretagem de café.

1.2 Problema

Aplicar técnicas de ML à análise comportamental em um ERP personalizado de corretagem de café se justifica pela necessidade de garantir a segurança e a integridade das informações do sistema. Detectar atividades suspeitas cedo permite à empresa agir de forma preventiva, seja bloqueando o acesso do usuário, seja alertando as partes envolvidas, o que evita prejuízos financeiros e danos à reputação da corretora.

A literatura sobre análise comportamental e ML em sistemas de gerenciamento é consolidada, mas se concentra em outros domínios. Du et al. (2017) trabalham sobre logs de sistemas distribuídos, Berlin, Slater e Saxe (2015) sobre logs de auditoria do Windows, Jiang et al. (2018) sobre o conjunto CMU-CERT, que modela um ambiente corporativo genérico, Ranjan e Kumar (2022) sobre logs de aplicações web e Bolt, de Leoni e van der Aalst (2018) sobre logs de processos de negócio. Nenhum desses trabalhos trata de um ERP setorial, e a especificidade da corretagem de café exige uma abordagem adaptada. O mercado cafeeiro apresenta características próprias que influenciam diretamente os padrões de comportamento dos usuários do sistema: a sazonalidade das safras, que concentra o volume de operações em períodos específicos do ano; a sensibilidade das informações de preço e de carteira de clientes, alvos naturais de tentativas de exfiltração de dados; e a diversidade de perfis de usuários, que vai de corretores experientes a estagiários em fase de aprendizado, cada um com padrões de acesso e frequência de ações distintos. Portanto, é necessário desenvolver um modelo de ML que considere essas características do mercado cafeeiro e dos perfis de usuários do ERP, de modo a produzir uma solução eficaz e adequada para o problema proposto.

1.3 Hipótese

Este trabalho investiga o potencial da classificação supervisionada, via biblioteca TensorFlow.js, para melhorar a segurança em sistemas de corretagem de café. A hipótese central é que essa abordagem identifica atividades suspeitas ou mal-intencionadas dos usuários com mais eficiência do que os métodos tradicionais baseados em regras estáticas.

Hipótese Principal: a implementação de modelos de classificação supervisionada com TensorFlow.js em sistemas de corretagem de café aprimora a detecção de compor-

tamentos suspeitos, aumentando a segurança do sistema frente a métodos convencionais baseados em regras fixas.

O estudo parte das seguintes premissas:

- A classificação supervisionada com TensorFlow.js melhora a detecção de padrões anômalos, tornando o sistema mais eficiente na identificação de comportamentos suspeitos.
- Os modelos de Aprendizado de Máquina têm vantagens sobre as abordagens tradicionais, com maior precisão na análise e na mitigação de ameaças.
- O TensorFlow.js tem características próprias que influenciam seu desempenho na análise de dados em tempo real, como flexibilidade, escalabilidade e velocidade de processamento.
- A natureza adaptativa dos modelos de Aprendizado de Máquina permite uma resposta dinâmica às mudanças nos padrões de comportamento dos usuários, mantendo o aprimoramento contínuo da segurança.

Para testar a hipótese, modelos de classificação supervisionada com TensorFlow.js foram desenvolvidos e testados com conjuntos de dados que representam os comportamentos dos usuários em sistemas de corretagem de café. Os experimentos reúnem evidências empíricas sobre a viabilidade e a eficácia da abordagem na detecção de anomalias, e o alcance dessas evidências é discutido no Capítulo 4.

1.4 Objetivos

O objetivo geral é desenvolver e implementar um modelo de ML para análise comportamental em um software de gerenciamento de corretagem de café, voltado à detecção e prevenção de ações indevidas ou suspeitas dos usuários. O modelo busca melhorar a segurança e a integridade das informações do sistema, protegendo os interesses de todos os *stakeholders* envolvidos na corretagem.

Para alcançá-lo, foram definidos os seguintes objetivos específicos:

- **Estudo das características comportamentais dos usuários:** investigar e documentar os padrões de comportamento dos usuários no software, identificando quais ações são normais ou suspeitas no contexto da corretagem de café.
- **Análise de logs de eventos:** examinar os logs gerados pelas ações dos usuários para entender como estruturar e usar esses dados no treinamento do modelo.

- ❑ **Desenvolvimento de um modelo de Aprendizado de Máquina:** construir e treinar um modelo a partir dos logs de eventos, com classificação supervisionada em TensorFlow.js.
- ❑ **Validação e teste do modelo:** avaliar a eficácia do modelo na detecção de comportamentos suspeitos, comparando seu desempenho com um *baseline* de regras estáticas por meio de métricas como acurácia, precisão, revocação, F1-Score e AUC-ROC.
- ❑ **Proposta de medidas preventivas:** sugerir medidas preventivas e ações corretivas a partir dos resultados da análise comportamental, contribuindo para a melhoria contínua da segurança do sistema.
- ❑ **Discussão sobre implicações éticas e legais:** refletir sobre as questões éticas e legais da vigilância comportamental e da privacidade dos usuários, propondo diretrizes para o uso responsável do ML.
- ❑ **Recomendação de práticas de segurança:** elaborar um conjunto de boas práticas de segurança da informação para o software de corretagem de café, protegendo o sistema contra ações mal-intencionadas.

1.5 Contribuições

Este trabalho contribui tanto para a prática da gestão de corretagem de café quanto para a pesquisa acadêmica em ML e análise comportamental. As principais contribuições são:

- ❑ **Aprimoramento da segurança em sistemas de corretagem de café:** o desenvolvimento de um modelo de classificação supervisionada para identificar comportamentos suspeitos ou maliciosos nesses sistemas. Com TensorFlow.js, a abordagem oferece uma forma de melhorar a segurança e a integridade das informações, o que importa para proteger dados sensíveis de clientes e operações comerciais.
- ❑ **Aplicação de ML à análise comportamental:** o trabalho explora o uso de algoritmos de Aprendizado de Máquina na análise comportamental em ambientes corporativos. O uso do TensorFlow.js para classificação supervisionada em um cenário realista de corretagem de café mostra como essas técnicas podem ser aplicadas no contexto empresarial.
- ❑ **Melhoria na eficiência operacional:** ao identificar comportamentos anômalos com mais eficiência e precisão, o trabalho contribui para a eficiência operacional das corretoras. A detecção precoce de atividades suspeitas permite intervir rápido e reduzir o risco de danos ou perdas financeiras.

- ❑ **Contribuições metodológicas:** o estudo apresenta um *framework* para implementar e avaliar modelos de ML em sistemas de segurança, que pode servir de base para pesquisas futuras em outros setores.

- ❑ **Avanço do conhecimento em segurança de sistemas de informação:** o trabalho amplia o conhecimento sobre a aplicação de ML à segurança de sistemas de informação, podendo inspirar pesquisas futuras e incentivar a adoção de abordagens baseadas em Aprendizado de Máquina em outros contextos.

Essas contribuições vêm acompanhadas de contrapartidas, e o trabalho as reconhece. Monitorar cada ação do usuário amplia a coleta de dados pessoais e cria uma tensão entre segurança e privacidade, tema retomado entre os objetivos específicos. Um classificador com poder de suspender acessos também transfere para o modelo uma decisão que antes era humana, e cada falso positivo interrompe o trabalho de alguém que nada fez de errado. Sommer e Paxson (2010) apontam o custo dos falsos positivos como o principal obstáculo ao uso de ML em detecção de intrusão fora do laboratório. Somam-se a isso a dependência de dados rotulados de qualidade e o risco de que a própria técnica sirva a atacantes, ponto levantado por Danziger e Henriques (2017). O mecanismo de *Human-in-the-Loop* descrito no Capítulo 3 responde em parte a essas contrapartidas, porque mantém a decisão final com um administrador humano e torna reversível toda suspensão automática.

1.6 Organização da Monografia

Este trabalho está organizado para dar uma visão completa do uso da análise comportamental com ML em sistemas de gerenciamento de corretagem de café. A estrutura é a seguinte:

Capítulo 1: Introdução. Estabelece o contexto e a importância da pesquisa, apresentando a motivação, o problema central, a hipótese, os objetivos, as contribuições esperadas e a estrutura da monografia.

Capítulo 2: Fundamentação Teórica e Trabalhos Relacionados. Traz a revisão da literatura sobre análise comportamental e ML, com os trabalhos relacionados, os conceitos fundamentais e as métricas usadas na avaliação. Mostra como o ML tem sido aplicado em contextos semelhantes e discute técnicas pertinentes à análise comportamental em sistemas empresariais.

Capítulo 3: Sistema Desenvolvido. Apresenta o sistema em detalhe: a estrutura geral da aplicação, as tecnologias utilizadas, os módulos da interface (gestão de clientes e contratos, gestão de incidentes, *dashboard* de métricas e visualização de logs) e os principais trechos de código relacionados ao modelo e à inferência em tempo real.

Capítulo 4: Experimentos e Análise dos Resultados. Detalha a geração do conjunto de dados sintéticos, a engenharia de *features*, o *baseline* de regras estáticas e os

resultados dos experimentos. É o capítulo que confronta a hipótese com as evidências e discute o alcance do que foi medido.

Capítulo 5: Conclusão. Discute as conclusões do estudo, resume as principais contribuições, aponta as limitações e sugere direções para trabalhos futuros, destacando como a pesquisa contribui para o campo de ML aplicado à segurança de sistemas de informação e à análise comportamental.

Capítulo 2

Fundamentação Teórica e Trabalhos Relacionados

Este capítulo apresenta os fundamentos teóricos que sustentam o desenvolvimento deste trabalho. A Seção 2.1 trata do Aprendizado de Máquina, com as duas abordagens de aprendizado relevantes para a proposta e as redes neurais artificiais que dão base ao modelo implementado. Vêm em seguida a análise comportamental (Seção 2.2) e a detecção de anomalias em sistemas de informação (Seção 2.3). As métricas usadas na avaliação são definidas na Seção 2.4, e o capítulo se encerra com os trabalhos relacionados (Seção 2.5), que exploram a intersecção entre análise comportamental, segurança da informação e Aprendizado de Máquina.

2.1 Aprendizado de Máquina

O Aprendizado de Máquina (ML, do inglês *Machine Learning*) é o ramo da Inteligência Artificial (IA) que estuda como sistemas podem aprender a partir de dados. Tem papel central na análise de dados e na identificação de padrões, em especial na segurança da informação.

- **Definição e origem:** o ML é a capacidade de sistemas aprenderem a partir de experiências, sem programação explícita. Essa característica é o que permite identificar comportamentos suspeitos em grandes volumes de dados, como no estudo de Ranjan e Kumar (2022).
- **Tipos de aprendizado:** há três categorias principais, o aprendizado supervisionado, o não supervisionado e o por reforço. As duas primeiras são detalhadas nas Seções 2.1.1 e 2.1.2. Em segurança da informação, o supervisionado predomina, como no uso do XGBoost para detecção de ameaças internas por Jiang et al. (2018).

- **Aplicações em segurança da informação:** o ML detecta padrões de ataque e identifica comportamentos maliciosos. Du et al. (2017), por exemplo, usam redes neurais recorrentes para encontrar padrões anormais em logs de sistema.

- **Algoritmos comuns:** a escolha do algoritmo depende dos dados e do problema. Árvores de decisão, redes neurais e máquinas de vetores de suporte (SVM) são opções frequentes, como na previsão de usuários mal-intencionados de Ranjan e Kumar (2022). Entre os métodos baseados em árvores, as florestas aleatórias (BREIMAN, 2001) combinam várias árvores para reduzir a variância, e o XGBoost (CHEN; GUESTRIN, 2016) aplica *boosting* de gradiente e costuma ir bem em dados tabulares.

- **Desafios e limitações:** o ML lida com o risco de *overfitting* e com a necessidade de muitos dados rotulados. A segurança dos próprios modelos também preocupa, ponto levantado por Danziger e Henriques (2017).

2.1.1 Aprendizado Supervisionado

O aprendizado supervisionado treina modelos com dados rotulados, ou seja, exemplos de entradas e saídas esperadas, o que ensina o modelo a mapear uma na outra (BERLIN; SLATER; SAXE, 2015). Durante o treinamento, o modelo produz uma predição para cada exemplo, compara o resultado com o rótulo verdadeiro por meio de uma função de perda e ajusta seus parâmetros para reduzir o erro. Ao final, espera-se que o modelo generalize, isto é, que acerte também em exemplos que não viu no treinamento.

As tarefas do aprendizado supervisionado se dividem em duas. Na classificação, a saída é uma categoria discreta, como rotular uma ação de usuário em normal ou suspeita. Na regressão, a saída é um valor contínuo, como estimar o tempo até o próximo acesso. Este trabalho se enquadra na classificação binária, já que cada ação registrada recebe um entre dois rótulos possíveis.

Na análise comportamental, o aprendizado supervisionado serve para classificar o comportamento do usuário em categorias definidas, como normal ou anormal, sinalizando possíveis ameaças (JIANG et al., 2018).

Para medir a generalização de forma honesta, o conjunto rotulado é dividido em três partições com funções distintas. O treino fornece os exemplos com os quais o modelo ajusta seus parâmetros. Durante o processo, a validação serve para acompanhar o desempenho em dados não vistos e detectar o início do *overfitting*, situação em que o erro de treino continua caindo enquanto o de validação estaciona ou sobe. Já o teste fica reservado até o fim e é usado uma única vez, para estimar o desempenho final. Como o modelo nunca ajustou nada com base nele, essa estimativa não é otimista por construção. A prática comum destina a maior parte dos dados ao treino e divide o restante entre validação e teste, e a proporção adotada neste trabalho é detalhada na Seção 4.1.1.

Outro ponto recorrente do aprendizado supervisionado em segurança é o desbalanceamento de classes. A situação é natural nesse domínio, pois eventos maliciosos são raros diante do volume de atividade legítima. O modelo tende a favorecer a classe majoritária e pode alcançar acurácia alta sem detectar quase nenhuma ameaça. Existem técnicas para reduzir esse impacto. O SMOTE (*Synthetic Minority Over-sampling Technique*), de Chawla et al. (2002), cria amostras sintéticas da classe minoritária por interpolação e equilibra o *dataset* antes do treinamento. Quando os dados são gerados de forma controlada, o equilíbrio entre as classes pode ser definido na própria geração, o que dispensa o *oversampling* posterior.

2.1.2 Aprendizado Não Supervisionado

O aprendizado não supervisionado trabalha com dados sem rótulo e busca padrões e estruturas ocultas (RANJAN; KUMAR, 2022).

Suas principais tarefas são o agrupamento e a associação. No agrupamento, o algoritmo separa as amostras em grupos de acordo com a similaridade entre elas, como o *K-means*, que particiona os dados em torno de centroides. Na associação, o objetivo é descobrir regras que relacionem a ocorrência conjunta de eventos, no formato de que a presença de um item sugere a presença de outro. Técnicas de redução de dimensionalidade, como a análise de componentes principais (PCA), costumam acompanhar essas tarefas para tornar os padrões mais visíveis.

As duas abordagens não competem entre si, pois respondem a perguntas diferentes. O aprendizado supervisionado se aplica quando as categorias de saída já são conhecidas e há rótulos disponíveis; o não supervisionado é exploratório e cabe quando os padrões não são conhecidos de antemão (DU et al., 2017). Em segurança da informação, o supervisionado aparece na detecção de intrusão e na prevenção de fraudes, enquanto o não supervisionado ajuda a identificar ataques ou anomalias inéditas (DANZIGER; HENRIQUES, 2017).

2.1.3 Redes Neurais Artificiais

Entre os algoritmos de classificação supervisionada, as redes neurais artificiais foram escolhidas neste trabalho por dois motivos. O primeiro é a capacidade de representar relações não lineares entre variáveis, necessária quando o indício de comportamento suspeito não está em nenhuma *feature* isolada, mas na combinação entre elas, como a associação entre horário do acesso, frequência de ações e módulo acessado. O segundo é a disponibilidade do TensorFlow.js, que permite treinar no servidor e executar a inferência no navegador, característica central da arquitetura descrita no Capítulo 3.

O *Deep Learning* é o subcampo do Aprendizado de Máquina baseado em redes neurais profundas. Sua principal vantagem é deixar o próprio modelo aprender representações dos dados brutos, sem exigir engenharia manual extensiva de *features* (LECUN; BENGIO;

HINTON, 2015). A segurança da informação tem aproveitado isso cada vez mais. Yuan e Wu (2021), por exemplo, revisam as arquiteturas de *Deep Learning* aplicadas à detecção de ameaças internas e listam os obstáculos da área, como a falta de dados rotulados e os ataques adaptativos. Uma arquitetura recorrente de destaque é a rede de memória de longo e curto prazo (LSTM, do inglês *Long Short-Term Memory*), voltada a dados sequenciais, empregada por Du et al. (2017) na detecção de anomalias em logs.

Treinar redes neurais profundas depende de técnicas que evitam o *overfitting* e estabilizam o processo. O *Dropout*, de Srivastava et al. (2014), desativa aleatoriamente parte dos neurônios durante o treinamento, o que obriga a rede a aprender representações mais robustas e reduz a dependência de neurônios específicos. A *Batch Normalization*, de Ioffe e Szegedy (2015), normaliza as ativações entre camadas, acelera a convergência e estabiliza o treinamento. Para a otimização, o algoritmo Adam (KINGMA; BA, 2015) reúne *momentum* e taxa de aprendizado adaptativa, sendo amplamente usado pela eficiência e robustez. Essas três técnicas formam a base do modelo desenvolvido neste trabalho. A Figura 1 mostra como elas se posicionam em uma arquitetura densa.

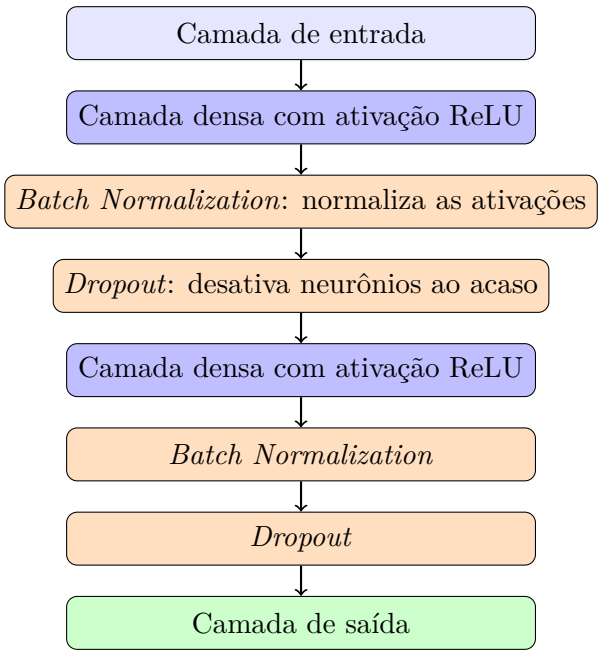

Figura 1 – Posição das técnicas de regularização em uma rede neural densa. Cada camada densa é seguida por *Batch Normalization* e *Dropout*, em laranja. A configuração concreta adotada neste trabalho, com o número de neurônios de cada camada, é apresentada na Seção 3.4.1.

2.2 Análise Comportamental

A análise comportamental estuda e interpreta padrões de comportamento humano para identificar desvios e anomalias. Em sistemas de informação corporativos, ela permite monitorar as ações dos usuários e detectar atividades maliciosas antes que causem dano.

Parte da ideia de que comportamentos são respostas a estímulos externos e podem ser observados, medidos e modificados (DU et al., 2017). Com ela, padrões sutis se tornam visíveis, o que viabiliza previsões e intervenções mais eficazes (RANJAN; KUMAR, 2022).

O Aprendizado de Máquina oferece várias técnicas para analisar o comportamento do usuário. A detecção de anomalias é a principal delas, identificando padrões que fogem da norma (DU et al., 2017). A modelagem de sequências e o *clustering* comportamental também ajudam a agrupar comportamentos parecidos e a revelar tendências (JIANG et al., 2018). Juntas, essas técnicas tornam os sistemas de detecção de ameaças mais eficientes.

As aplicações são amplas. Na detecção de fraudes, a análise comportamental aponta comportamentos suspeitos que podem indicar atividade fraudulenta (RANJAN; KUMAR, 2022). Na segurança cibernética, identifica anomalias que sinalizam tentativas de intrusão (BERLIN; SLATER; SAXE, 2015). O método também se aplica a áreas como *marketing*, saúde mental e criminologia, onde entender o comportamento humano importa (DANZIGER; HENRIQUES, 2017).

2.3 Detecção de Anomalias e Prevenção de Fraudes em Sistemas de Informação

A detecção de anomalias e a prevenção de fraudes são aplicações centrais do Aprendizado de Máquina na segurança de sistemas, sobretudo em contextos como a corretagem de café, onde a integridade dos dados e a rastreabilidade das ações são críticas. Esta seção apresenta os fundamentos dessas técnicas, as categorias de *features* que representam o comportamento dos usuários e os desafios de implementá-las em ambientes reais.

Em setores como a corretagem de café, a segurança e a integridade dos dados são vitais, o que torna a detecção de anomalias indispensável. O ML identifica padrões irregulares e sinaliza comportamentos potencialmente fraudulentos, processando grandes volumes de dados e apontando desvios que indicam atividade anômala (BERLIN; SLATER; SAXE, 2015).

A análise de logs de eventos complementa essas técnicas. Os logs trazem dados detalhados que, uma vez analisados, alimentam o monitoramento de segurança e a análise comportamental (DANZIGER; HENRIQUES, 2017). Essa combinação identifica padrões suspeitos e ajuda a entender o comportamento do usuário, base para prevenir fraudes e proteger os dados.

A detecção de anomalias é um campo já consolidado, sistematizado no trabalho de Chandola, Banerjee e Kumar (2009), que organiza as técnicas existentes pela abordagem adotada e separa três tipos de anomalia, as pontuais, as contextuais e as coletivas. Essa taxonomia deixa claro que um mesmo comportamento pode ser normal ou anômalo conforme o contexto, princípio que se aplica de forma direta à análise de usuários, na qual

a hora do acesso, a sequência de ações e a localização determinam como cada evento é interpretado. No âmbito de redes, Ahmed, Mahmood e Hu (2016) revisam técnicas de detecção de anomalias, enquanto Buczak e Guven (2016) organizam os métodos de mineração de dados e ML aplicados à detecção de intrusões. Vale registrar a ressalva de Sommer e Paxson (2010) sobre as dificuldades de usar ML em detecção de intrusão no mundo real, com destaque para o alto custo dos falsos positivos e a dificuldade de obter dados de treinamento representativos, ponto que motiva o uso do mecanismo de *Human-in-the-Loop* neste trabalho. No tema das ameaças internas, Homoliak et al. (2019) propõem uma taxonomia que organiza as ameaças por intenção do agente, tipo de acesso e contramedidas, e Liu et al. (2018) revisam métodos de detecção e prevenção, reforçando a análise do comportamento do usuário como vetor central.

Para que essas técnicas operem, o comportamento do usuário precisa ser traduzido em variáveis mensuráveis. A literatura organiza essas variáveis em três categorias, conforme a natureza da informação que carregam. Na categoria de ambiente entram os detalhes técnicos e de acesso, como o momento e a natureza de cada ação. O comportamento agrupa o que descreve a atividade do usuário ao longo do tempo, como frequência, variedade e sequência das ações. O contexto, por fim, informa a origem do acesso, o que permite identificar tentativas não autorizadas ou fora do comum. A Tabela 1 resume as três categorias.

Tabela 1 – Categorias de *features* para representação do comportamento do usuário.

Feature	**Descrição**
Categoria: Ambiente	
Timestamp	Data e hora exatas em que a ação foi realizada.
Identificador do usuário	Código único atribuído a cada usuário.
Nome do usuário	Nome real ou nome de usuário atribuído.
Nível de acesso	Nível de permissão do usuário.
Tipo de ação	Natureza da ação realizada (leitura, escrita, edição, exclusão).
Detalhes da ação	Informações específicas, como identificadores de registros afetados.
Resultado da ação	Indica se a ação foi bem-sucedida.
Dispositivo	Tipo de dispositivo usado no acesso.
Navegador e versão	Informações sobre o navegador utilizado.
Módulos acessados	Partes do sistema acessadas pelo usuário.
Duração da sessão	Tempo total da sessão do usuário.
Mensagens de erro	Erros gerados durante a sessão.
Categoria: Comportamento	
Frequência de ações	Quantidade de vezes que o usuário realiza uma ação.
Variedade de ações	Diversidade de tipos de ação realizados.
Sequência de ações	Ordem e padrões repetitivos nas ações.
Mudanças de perfil	Alterações de configuração ou ações que exigem privilégio elevado.
Acesso a dados sensíveis	Frequência de acesso a informações confidenciais.
Comportamento de cópia	Quantidade de vezes que a ação de copiar é realizada.
Categoria: Contexto	
IP de acesso	Endereço IP utilizado, indicando a localização de origem.
Localização geográfica	Localização física do acesso.
Conectividade de rede	Qualidade da conexão utilizada.
Alterações de configuração	Mudanças feitas pelo usuário nas configurações do sistema.
Tentativas de *login*	Número de tentativas necessárias para acessar o sistema.

Vale um comentário sobre a entropia de Shannon, uma das *features* de comportamento. Ela foi proposta por Shannon (1948) na teoria da informação e mede o quanto uma sequência de eventos é imprevisível. Na análise comportamental, serve para quantificar a regularidade das ações do usuário, sequências previsíveis e repetitivas dão entropia baixa, sequências variadas e atípicas dão entropia alta. É esse contraste que ajuda a separar a rotina do que foge do padrão.

Usadas junto com técnicas de Aprendizado de Máquina e análise de logs, essas *features* formam uma abordagem promissora para detectar fraudes e melhorar a segurança dos sistemas. A área ainda evolui e sua eficácia precisa de validação contínua, mas os estudos de Du et al. (2017) e Berlin, Slater e Saxe (2015) indicam o potencial dessas abordagens na identificação de comportamentos suspeitos.

Implementar esses sistemas em ambientes como o da corretagem de café traz desafios. O primeiro é coletar e processar dados precisos e confiáveis, já que a qualidade dos dados determina a eficácia dos modelos, como mostra Berlin, Slater e Saxe (2015), que enfatizam a importância de uma coleta robusta e representativa.

Outro desafio é integrar fontes de informação diversas. Em sistemas complexos, os

dados vêm de vários lugares e formatos, o que dificulta a integração e a análise. Danziger e Henriques (2017) destacam essa complexidade ao discutir a aplicação de ML em ambientes de segurança dinâmicos e heterogêneos.

Há ainda a evolução constante das técnicas de ataque, que exige sistemas de detecção sempre atualizados. As ameaças são dinâmicas e demandam modelos capazes de se adaptar a novos padrões, como discute Du et al. (2017). Adaptar-se rápido a novas estratégias de ataque é o que mantém a detecção eficaz.

Portanto, implementar esses sistemas na corretagem de café e em contextos parecidos exige uma abordagem adaptativa, atenta tanto à qualidade e integração dos dados quanto à resposta às mudanças nas táticas de ameaça. Pesquisa contínua e novas técnicas são essenciais para enfrentar esses desafios e garantir a segurança dos sistemas de informação.

2.4 Métricas de Avaliação

A avaliação de um classificador binário parte da matriz de confusão, tabela 2×2 que cruza as classes reais com as preditas. Dela derivam as demais métricas usadas neste trabalho. A matriz distingue quatro resultados possíveis: os Verdadeiros Positivos (VP), ações suspeitas corretamente identificadas; os Verdadeiros Negativos (VN), ações normais corretamente classificadas; os Falsos Positivos (FP), ações normais sinalizadas como suspeitas; e os Falsos Negativos (FN), ações suspeitas que passaram sem detecção.

A partir dessas quatro contagens, definem-se as métricas a seguir.

- **Acurácia (*Accuracy*):** proporção total de classificações corretas, dada por $(VP + VN)$/total. Indica o desempenho geral do classificador, mas perde utilidade em conjuntos desbalanceados.

- **Precisão (*Precision*):** dentre as ações classificadas como suspeitas, quantas realmente eram, dada por $VP/(VP + FP)$. Mede a confiabilidade dos alertas.

- **Revocação (*Recall*):** dentre todas as ações realmente suspeitas, quantas o modelo detectou, dada por $VP/(VP + FN)$. Em segurança, é a métrica mais crítica, pois mede a capacidade de não deixar ameaças passarem.

- **F1-Score:** média harmônica entre precisão e revocação:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

  Equilibra as duas e serve como métrica principal na comparação entre métodos.

- **AUC-ROC:** área sob a curva ROC (*Receiver Operating Characteristic*), que relaciona a taxa de verdadeiros positivos (TPR) com a taxa de falsos positivos (FPR) ao longo dos possíveis limiares de decisão. Mede a capacidade do modelo de separar

as duas classes independentemente do limiar escolhido, e valores próximos de 1,0 indicam boa discriminação.

2.5 Trabalhos Relacionados

A segurança da informação e a detecção de ameaças internas avançaram bastante com o Aprendizado de Máquina. Esta seção examina estudos influentes na intersecção entre análise comportamental, segurança da informação e ML, formando a base para o sistema proposto neste trabalho.

Ranjan e Kumar (2022) usaram *Big Data Analytics* com Aprendizado de Máquina para prever usuários mal-intencionados e legítimos a partir de padrões de navegação extraídos de logs de camada de aplicação. O modelo combinou árvores de decisão e florestas aleatórias na classificação, mostrando a aplicabilidade dessas técnicas em segurança cibernética. O uso conjunto dos dois algoritmos deu robustez ao método e melhorou a precisão dos resultados, com foco nos padrões de navegação como principal fonte de informação comportamental.

O estudo tem limitações relevantes. O modelo depende muito da qualidade e da representatividade dos dados de navegação, então dados incompletos ou enviesados comprometem o resultado. Há também risco de falsos positivos e negativos em ambientes com padrões complexos, e dificuldade de generalização para outros tipos de aplicação ou rede, o que exige ajustes em cada caso. Neste trabalho, a abordagem de Ranjan e Kumar (2022) inspirou o perfil *Data Exfiltration*, que simula leituras excessivas e sequenciais como indício de tentativa de extração de dados.

Bolt, de Leoni e van der Aalst (2018) propuseram comparar variantes de processo com técnicas de Aprendizado de Máquina aplicadas a logs de eventos. O objetivo era identificar diferenças de comportamento e de regras de negócio entre variantes, revelando distinções operacionais e estratégicas em ambientes corporativos. A metodologia é flexível e se aplica a vários contextos para entender e gerir processos.

Entre as limitações, a metodologia pode ser complexa de implementar, exigindo *expertise* em ML e bom conhecimento dos processos de negócio. A eficácia depende da qualidade e integridade dos logs, e a generalização para processos fora do padrão é limitada. Neste trabalho, a abordagem de Bolt, de Leoni e van der Aalst (2018) embasou as *features* de comportamento, como frequência de ações e entropia de Shannon, que capturam padrões comportamentais nos logs.

Du et al. (2017) apresentaram o DeepLog, abordagem baseada em redes neurais *Long Short-Term Memory* (LSTM) para detectar anomalias e diagnosticar falhas a partir de logs. O sistema aprende padrões de log automaticamente e se adapta a novos padrões com o tempo, sem intervenção manual extensa. Além de detectar anomalias, auxilia no diagnóstico de falhas, e prevê a incorporação de *feedback* do usuário para se atualizar

quando há falsos positivos.

As limitações do DeepLog incluem a dependência de logs de qualidade, o custo computacional das redes profundas e a dificuldade de interpretar os resultados, que exige conhecimento especializado. Neste trabalho, a abordagem de Du et al. (2017) inspirou tanto o perfil *Insider Threat*, que simula atividade anômala em logs, quanto o mecanismo de *Human-in-the-Loop* com re-treinamento incremental, parecido com a atualização por *feedback* proposta pelos autores.

Berlin, Slater e Saxe (2015) usaram regressão logística para construir um classificador de comportamento malicioso em logs de auditoria do Windows. O trabalho identificou informações-chave nos logs para detectar atividades suspeitas, mostrando que focar em dados relevantes aumenta a precisão. A técnica se aplica bem a ambientes corporativos e oferece uma ferramenta eficiente de monitoramento.

Entre as limitações estão a dependência da qualidade dos logs, o risco de falsos positivos e negativos em ambientes dinâmicos e a necessidade de ajustes frequentes diante da evolução dos ataques. Neste trabalho, o estudo de Berlin, Slater e Saxe (2015) embasou o perfil *Privilege Escalation*, que simula tentativas de ações fora das permissões do usuário como sinal de escalação de privilégios.

Jiang et al. (2018) usaram o XGBoost para analisar comportamento de usuários e detectar ameaças internas, com o *dataset* CMU-CERT e mais de 135 milhões de eventos. O modelo classificou usuários como normais ou maliciosos com alta precisão, superando outros algoritmos tradicionais. Para enfrentar o desbalanceamento severo dos dados (proporção de 315.693:1 entre amostras normais e anômalas), aplicaram a técnica de *oversampling* SMOTE antes do treinamento.

As limitações incluem a dependência de dados de alta qualidade, a complexidade de implementar e ajustar o modelo e a dificuldade de interpretar suas decisões. Neste trabalho, a abordagem de Jiang et al. (2018) inspirou o perfil *Account Compromise*, que simula mudanças de contexto de acesso como indício de conta comprometida, e a decisão de adotar distribuição balanceada entre classes no *dataset* sintético, justamente porque os autores mostraram que o balanceamento é essencial para a classe minoritária.

Danziger e Henriques (2017) trataram o uso de ML em segurança da informação por dois ângulos, discutindo como as técnicas servem tanto para fortalecer a segurança quanto para fins maliciosos. O estudo dá uma visão ampla dos riscos do uso adversarial de ML, alertando que atacantes podem empregar essas técnicas para criar métodos de ataque mais sofisticados, e incentiva estratégias proativas de prevenção.

Entre as limitações, o artigo carece de exemplos práticos e estudos de caso em ambientes reais, não dá orientações concretas de contramedidas e aponta a necessidade de mais pesquisa. Neste trabalho, a perspectiva de Danziger e Henriques (2017) contribuiu para o perfil *Insider Threat*, que representa um colaborador legítimo agindo de forma maliciosa, e para a análise das limitações das abordagens tradicionais baseadas em regras estáticas.

2.5.1 Análise Comparativa

A Tabela 2 apresenta a comparação entre o presente trabalho e os correlatos a partir de características técnicas concretas, o algoritmo, os tipos de ataque cobertos, a fonte e o volume de dados, o número de *features*, as métricas reportadas, a adaptação online e o contexto de aplicação.

Tabela 2 – Análise comparativa detalhada entre o presente trabalho e os trabalhos correlatos.

Trabalho	**Algoritmo**	**Tipos de Ataques**	**Fonte de Dados**	**Amostras**	**Features**	**Melhor Métrica**	**Adaptação Online**	**Contexto**
Ranjan e Kumar (2022)	Árvores de decisão + Random Forest	Usuários mal-intencionados e legítimos	Logs de aplicação web	Não informado	Não detalhadas	Acurácia: 70%	Não	Aplicação web genérica
Bolt, de Leoni e van der Aalst (2018)	Clustering + classificação	Variações de processos de negócios	Logs de eventos de processo	Variável	Não detalhadas	Não reporta classificação	Não	Processos de negócios
Du et al. (2017)	LSTM	Anomalias de sistema	Logs de sistema (HDFS)	15.923 sessões	Sequências de log keys	F1: 96% (HDFS)	Parcial	Logs de sistema e rede
Berlin, Slater e Saxe (2015)	Regressão logística	Comportamento malicioso	Logs de auditoria Windows	32.078	6.898.953 (binárias)	TPR: 89% a 1% FPR	Não	Malware em Windows
Jiang et al. (2018)	XGBoost	Ameaças internas	CMU-CERT	135 milhões	Por usuário	Foco em AUC	Não	Ameaças internas
Danziger e Henriques (2017)	N/A (teórico)	Uso malicioso de ML	N/A	N/A	N/A	N/A	N/A	Análise teórica
Presente Trabalho	**Rede neural densa (TensorFlow.js)**	**Exfiltration, Privilege Escalation, Account Compromise, Insider Threat**	**Logs de ERP (sintéticos)**	**10.000**	**30 (3 categorias)**	**F1: 99,68%; Recall: 99,48%**	**Sim (HITL)**	**ERP corretagem de café**

A Tabela 2 indica que o presente trabalho se distingue dos demais em três pontos. Primeiro, é o único com inferência em tempo real no navegador do usuário via TensorFlow.js, já que os outros processam tudo no servidor. Segundo, o mecanismo de *Human-in-the-Loop* com re-treinamento incremental só aparece, e ainda assim de forma parcial, no DeepLog de Du et al. (2017), que prevê atualização por *feedback* mas não traz painel

de gestão com histórico auditável nem bloqueio automático. Terceiro, este trabalho usa uma engenharia de *features* explícita, com 30 variáveis em três categorias, ao passo que Ranjan e Kumar (2022) e Bolt, de Leoni e van der Aalst (2018) não detalham suas *features* e Berlin, Slater e Saxe (2015) usa quase 7 milhões de *features* binárias extraídas automaticamente, sem categorização semântica.

Por outro lado, há limitações em relação aos correlatos. Os trabalhos de Jiang et al. (2018), Berlin, Slater e Saxe (2015) e Du et al. (2017) usam dados reais de operação, com volumes bem maiores, até 135 milhões de eventos, enquanto este estudo usa 10.000 amostras sintéticas. O DeepLog de Du et al. (2017) ainda incorpora modelagem temporal com LSTM, capturando dependências sequenciais que a arquitetura densa deste trabalho não modela de forma explícita. Essas limitações são discutidas no Capítulo 4, e os caminhos para superá-las aparecem entre os trabalhos futuros.

Capítulo 3

Sistema Desenvolvido

Este capítulo apresenta o sistema desenvolvido para análise comportamental com Aprendizado de Máquina em um ERP de corretagem de café. São descritas a estrutura geral da aplicação, as tecnologias utilizadas, os principais componentes do código-fonte e as telas que compõem a interface.

3.1 Estrutura Geral

A solução foi desenvolvida para monitorar em tempo real as ações dos usuários de um sistema de gerenciamento de corretagem de café, classificando cada ação como normal ou suspeita por meio de um modelo de Aprendizado de Máquina. A arquitetura tem dois componentes principais: a interface web (*frontend*) e o servidor de aplicação (*backend*).

O *frontend* foi desenvolvido em React com TypeScript e cuida da interface gráfica, da captura dos logs de ações e da inferência em tempo real do modelo diretamente no navegador. Com essa decisão, cada ação do usuário é classificada na hora, sem depender de comunicação com o servidor para a predição.

Executar a inferência no cliente levanta a questão do custo imposto ao navegador. O modelo tem 4.993 parâmetros e é carregado uma única vez, no início da sessão, o que corresponde a cerca de 20 kB de pesos em precisão de 32 bits. Cada predição opera sobre um vetor de 30 posições e se resolve em poucos milissegundos, sem travar a interface. A decisão tem um limite conhecido: se o modelo não carregar no navegador, a ação continua sendo registrada no servidor e a trilha de auditoria permanece íntegra, mas a classificação em tempo real daquela sessão deixa de acontecer. Espelhar a inferência no servidor, como conferência, é a evolução natural da arquitetura.

O *backend* foi implementado em Node.js com Express e responde pelo gerenciamento dos dados, pelo treinamento do modelo e pela API REST que alimenta o *frontend*. O banco de dados SQLite (via sql.js) guarda os dados de usuários, logs de eventos, incidentes e *feedbacks* do administrador.

A Figura 2 mostra a tela de autenticação, ponto de entrada para todos os usuários.

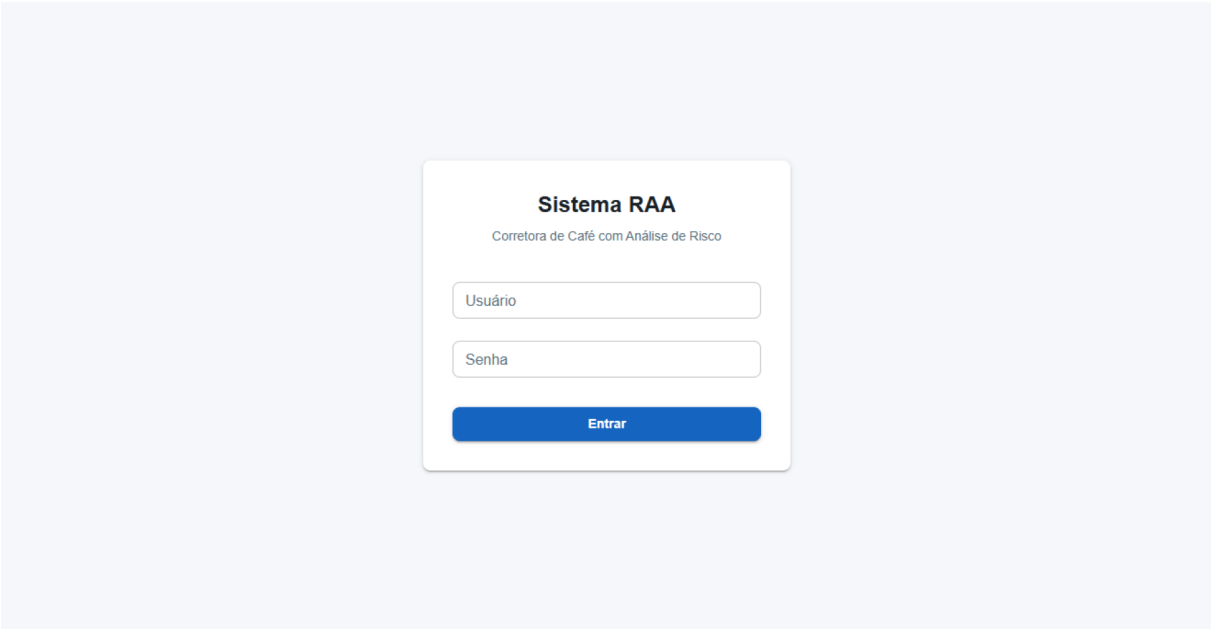

Figura 2 – Tela de login do sistema.

3.2 Tecnologias Utilizadas

A Figura 3 situa cada tecnologia na camada em que atua e mostra o fluxo entre as duas pontas: o *backend* treina o modelo e o exporta, o *frontend* carrega esse modelo e classifica as ações, e a comunicação entre eles ocorre por uma API REST.

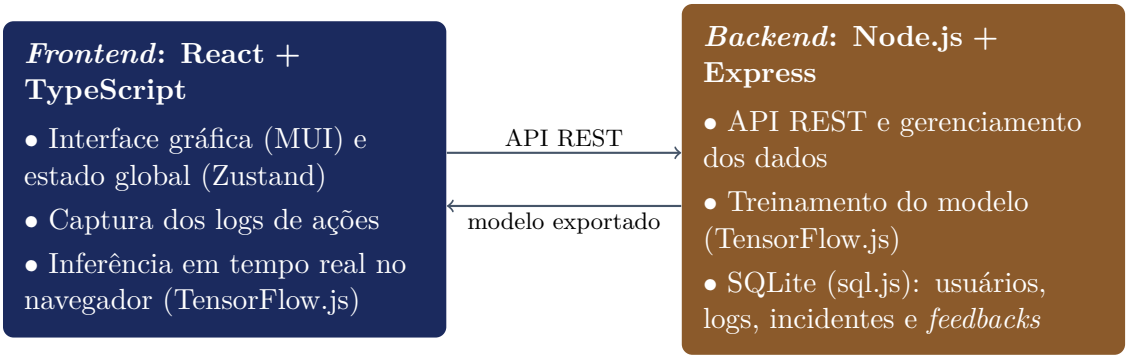

Figura 3 – Arquitetura do sistema e atuação de cada tecnologia por camada.

A Tabela 3 detalha a função de cada tecnologia.

Tabela 3 – Tecnologias utilizadas, camada de atuação e função no sistema.

Tecnologia	**Camada**	**Função no sistema**
React com TypeScript	*Frontend*	Construção da interface gráfica, com tipagem estática para reduzir erros em tempo de desenvolvimento.
Zustand	*Frontend*	Gerenciamento do estado global da aplicação: usuário autenticado, incidentes e logs.
Material UI (MUI)	*Frontend*	Componentes visuais no padrão *Material Design*, usados em tabelas, formulários, painéis e diálogos.
TensorFlow.js (SMILKOV et al., 2019)	*Frontend* e *Backend*	Treinamento do modelo no servidor e inferência em tempo real no navegador.
Node.js com Express	*Backend*	Ambiente de execução e *framework* web responsável pela API REST.
SQLite (via sql.js)	*Backend*	Banco relacional embutido, que persiste os dados sem exigir um servidor de banco separado.

3.3 Módulos do Sistema

O sistema tem quatro módulos principais: Gestão de Clientes e Contratos, Gestão de Incidentes, *Dashboard* de Métricas do Modelo e Visualização de Logs. As subseções seguintes detalham cada um e suas interfaces.

3.3.1 Gestão de Clientes e Contratos

O módulo de gestão de clientes permite cadastrar, visualizar e editar os dados dos clientes da corretora. Cada interação do usuário com o módulo gera um log de evento. O *frontend* captura esse log, extrai dele as *features* correspondentes e as submete ao modelo, que devolve o *score* de risco da ação. A Figura 4 mostra a tela de cadastro e visualização de clientes.

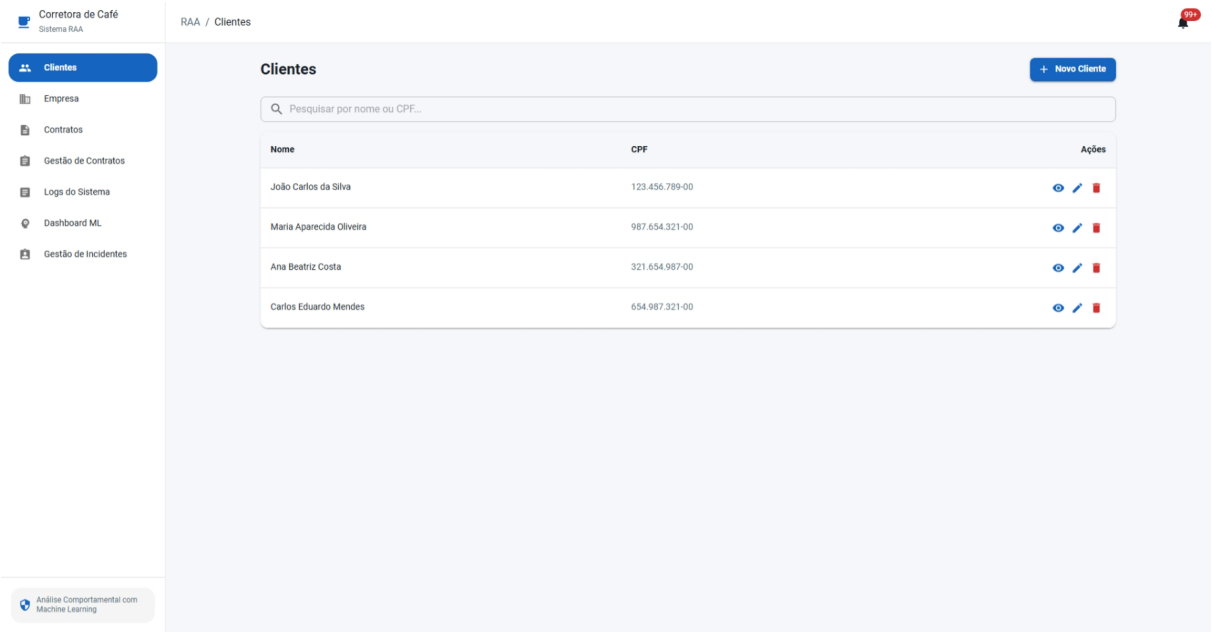

Figura 4 – Tela de cadastro, visualização e edição de clientes.

O módulo de contratos permite gerar, visualizar e baixar contratos de corretagem, além de acompanhar o progresso de cada um. As Figuras 5 e 6 mostram, respectivamente, a tela de geração de contratos e sua versão estendida com mais detalhes.

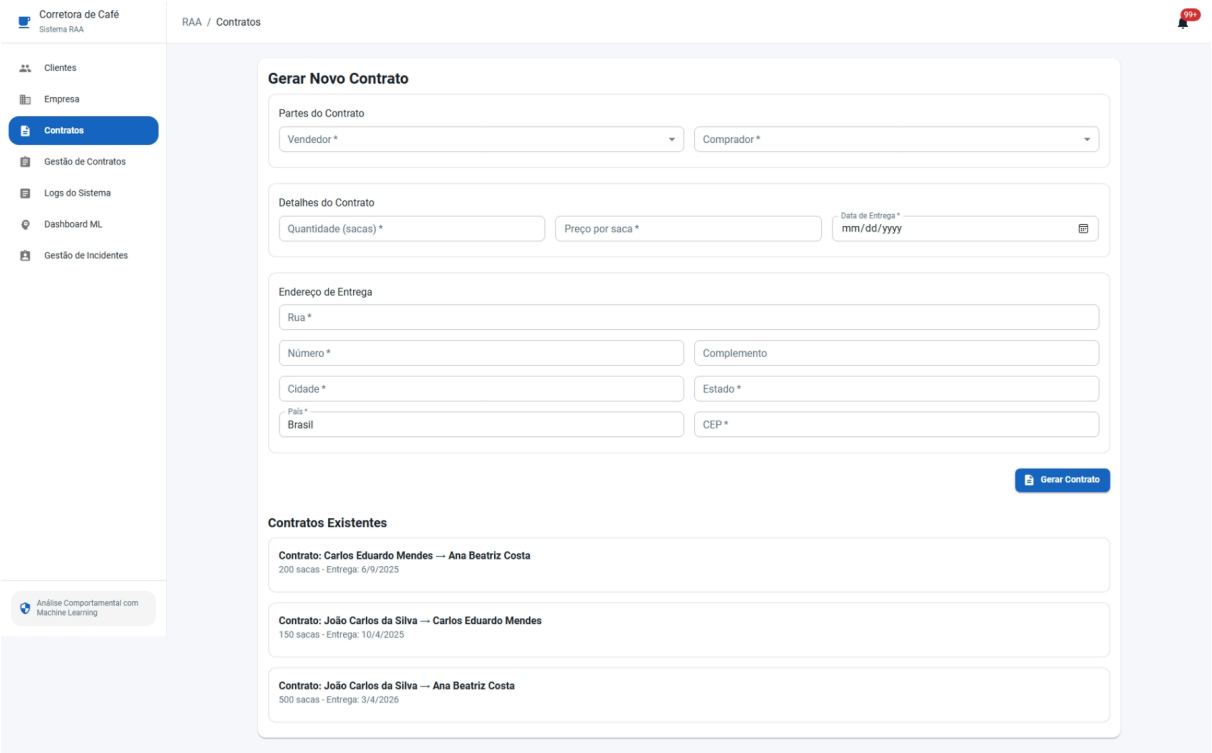

Figura 5 – Tela de geração, visualização e *download* de contratos.

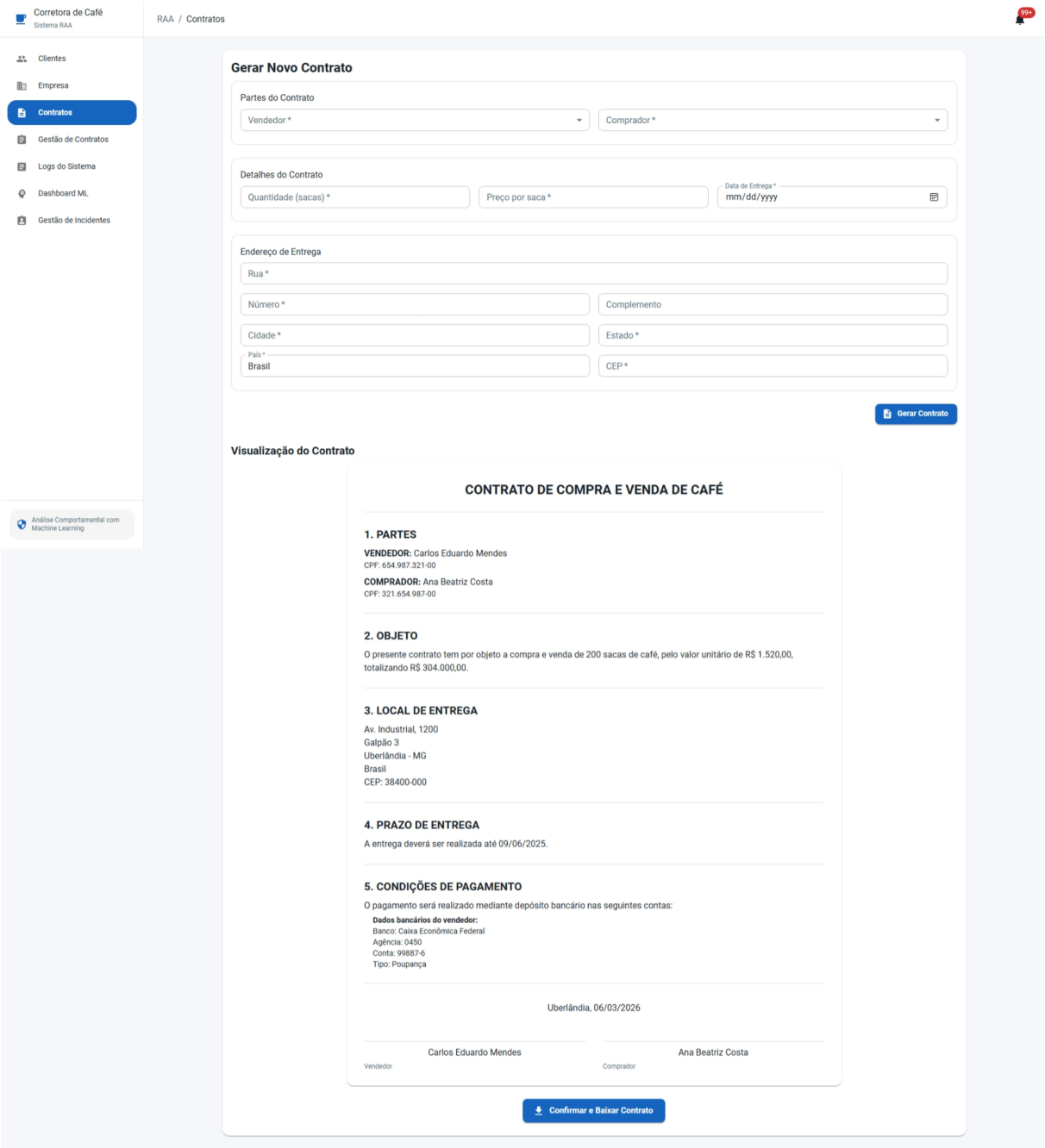

Figura 6 – Tela de contratos com visualização estendida.

A Figura 7 mostra a tela de gestão de contratos com edição de progresso, e a Figura 8 mostra a tela de dados da empresa.

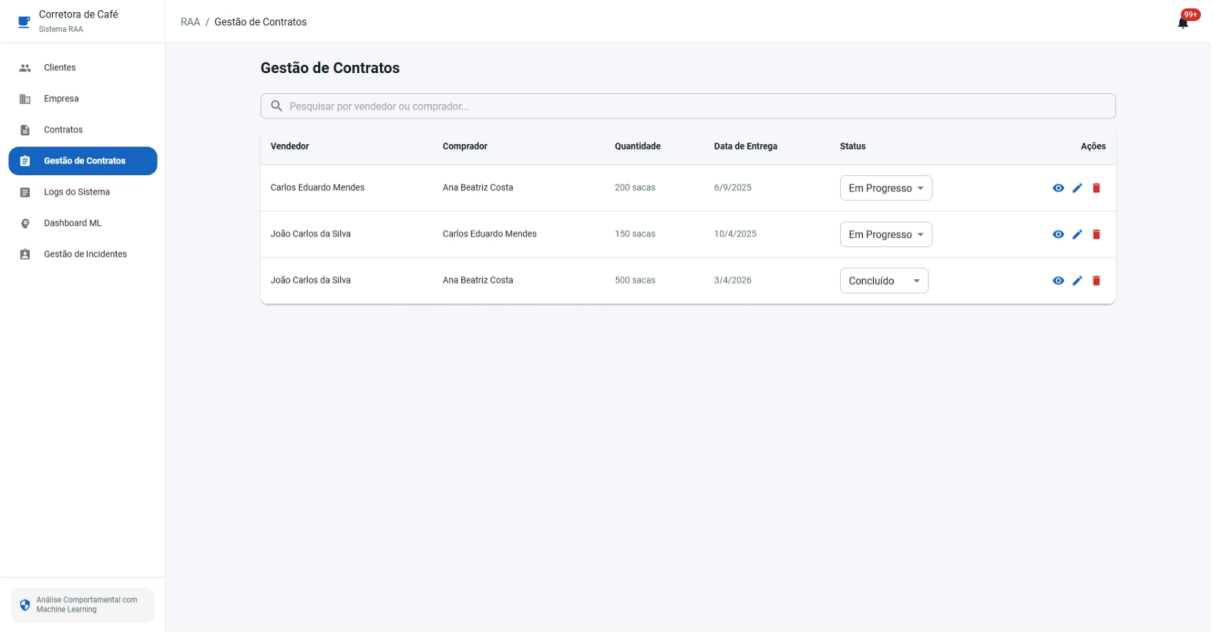

Figura 7 – Tela de gestão de contratos e edição de progresso.

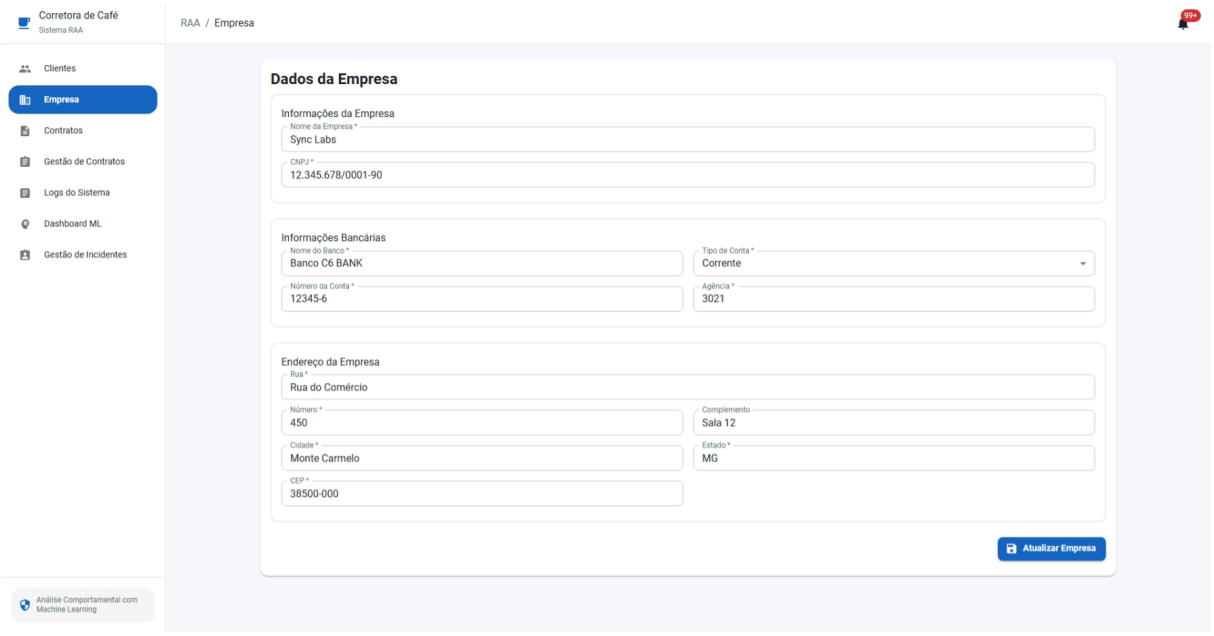

Figura 8 – Tela de dados da empresa e edição dos dados.

3.3.2 Gestão de Incidentes e Human-in-the-Loop

O módulo de gestão de incidentes é o centro da proposta de segurança do sistema. Quando o modelo classifica uma ação com *score* de risco igual ou acima do limiar configurado (70%), um incidente é criado automaticamente e o usuário é suspenso de forma

temporária. O valor de 70% foi definido a partir dos resultados experimentais: com AUC-ROC de 0,9999 (Capítulo 4), o modelo mantém alta discriminação entre as classes em uma faixa ampla de limiares, o que permite escolher o ponto de corte com base no custo relativo dos erros. Em segurança da informação, falsos negativos são mais custosos do que falsos positivos, pois uma ameaça não detectada causa dano direto, enquanto uma suspensão indevida é revertida pelo SuperAdmin na mesma interface. O limiar foi fixado em 70%, acima do ponto de corte padrão de 50%, de modo que apenas ações com probabilidade igual ou superior a esse valor são sinalizadas como suspeitas. Os 216 falsos negativos do *baseline* de regras estáticas (Seção 4.2.2) ilustram o custo concreto de um sistema menos sensível. O limiar pode ser ajustado pelo SuperAdmin sem alteração no código, o que permite calibrá-lo conforme o apetite de risco de cada organização.

O SuperAdmin acessa o painel de incidentes para analisar cada caso e decidir entre confirmar como ameaça ou liberar como legítimo. A Figura 9 mostra a tela de decisão, onde o administrador vê os detalhes da ação suspeita e o *score* de risco atribuído pelo modelo.

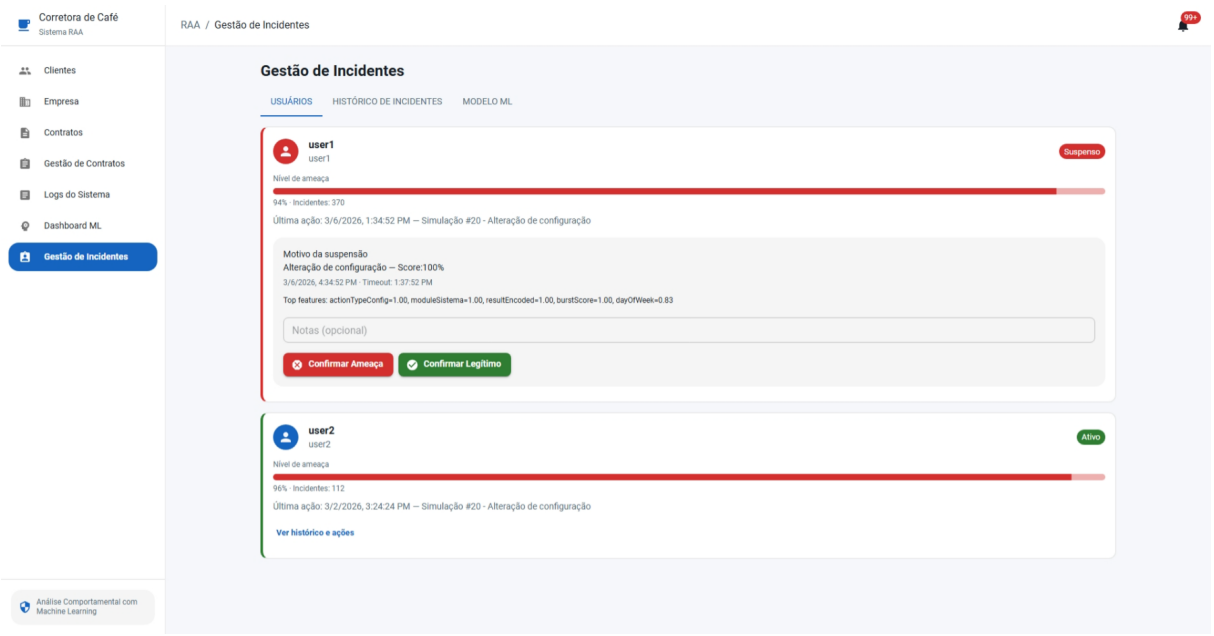

Figura 9 – Tela de gestão de incidentes com decisão de ameaça ou legítimo.

A Figura 10 mostra o histórico de incidentes registrados, que permite ao administrador consultar decisões anteriores e acompanhar a evolução dos eventos de segurança.

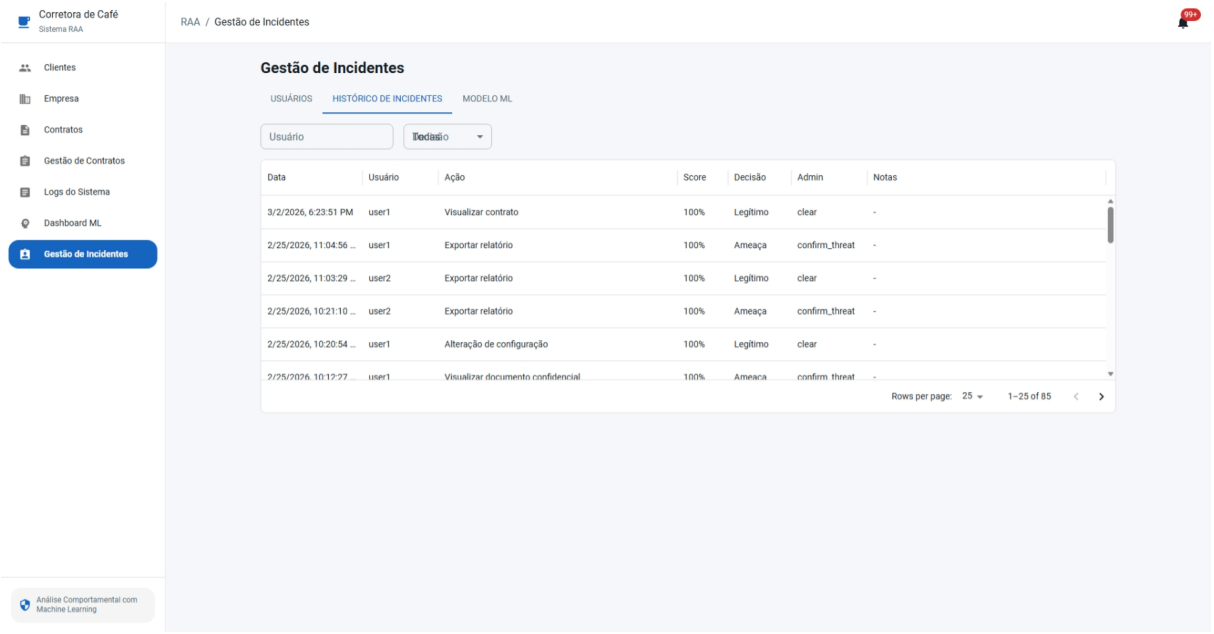

Figura 10 – Tela de histórico de incidentes registrados.

As decisões do administrador alimentam o ciclo de *Human-in-the-Loop*: cada confirmação ou liberação vira uma nova amostra rotulada, incorporada ao *dataset* de *feedback*. Quando há *feedbacks* suficientes (mínimo de 20 decisões), o painel de re-treinamento deixa o SuperAdmin disparar uma nova sessão de treinamento. A Figura 11 mostra essa funcionalidade.

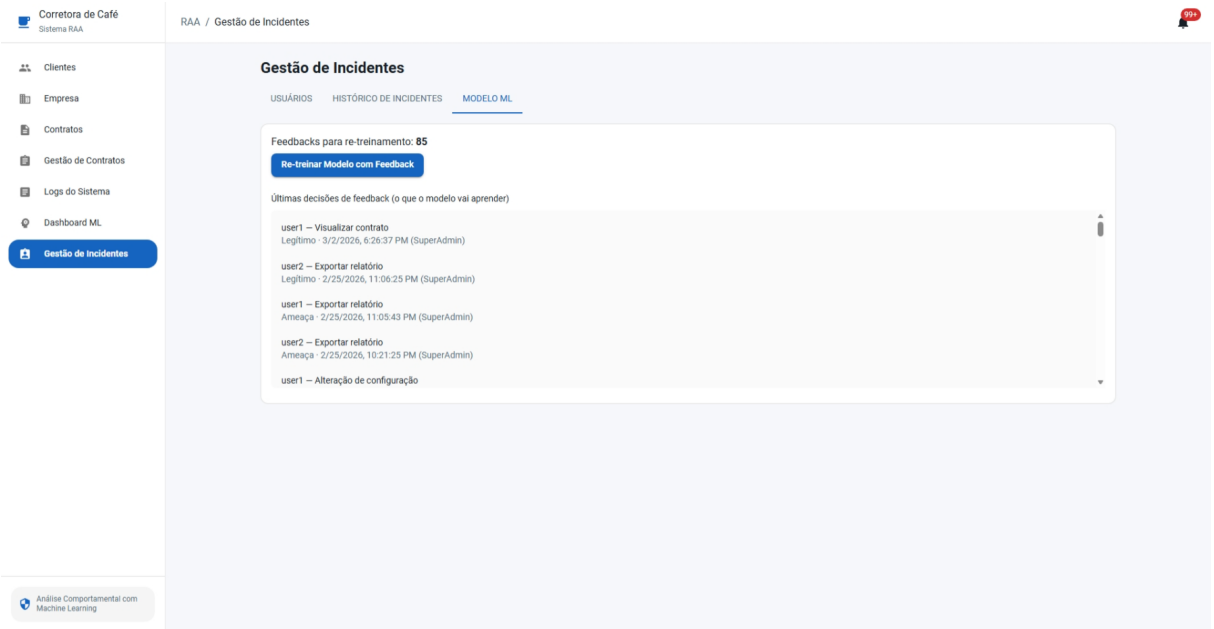

Figura 11 – Tela de re-treinamento do modelo com *feedbacks* acumulados.

3.3.3 Dashboard de Métricas do Modelo

O *dashboard* de métricas permite ao administrador acompanhar o desempenho do modelo ao longo do tempo. Ele exibe acurácia, precisão, revocação, F1-Score e AUC-ROC, todas definidas na Seção 2.4, além da versão do modelo em uso e do número de amostras de *feedback* acumuladas. A Figura 12 mostra essa interface.

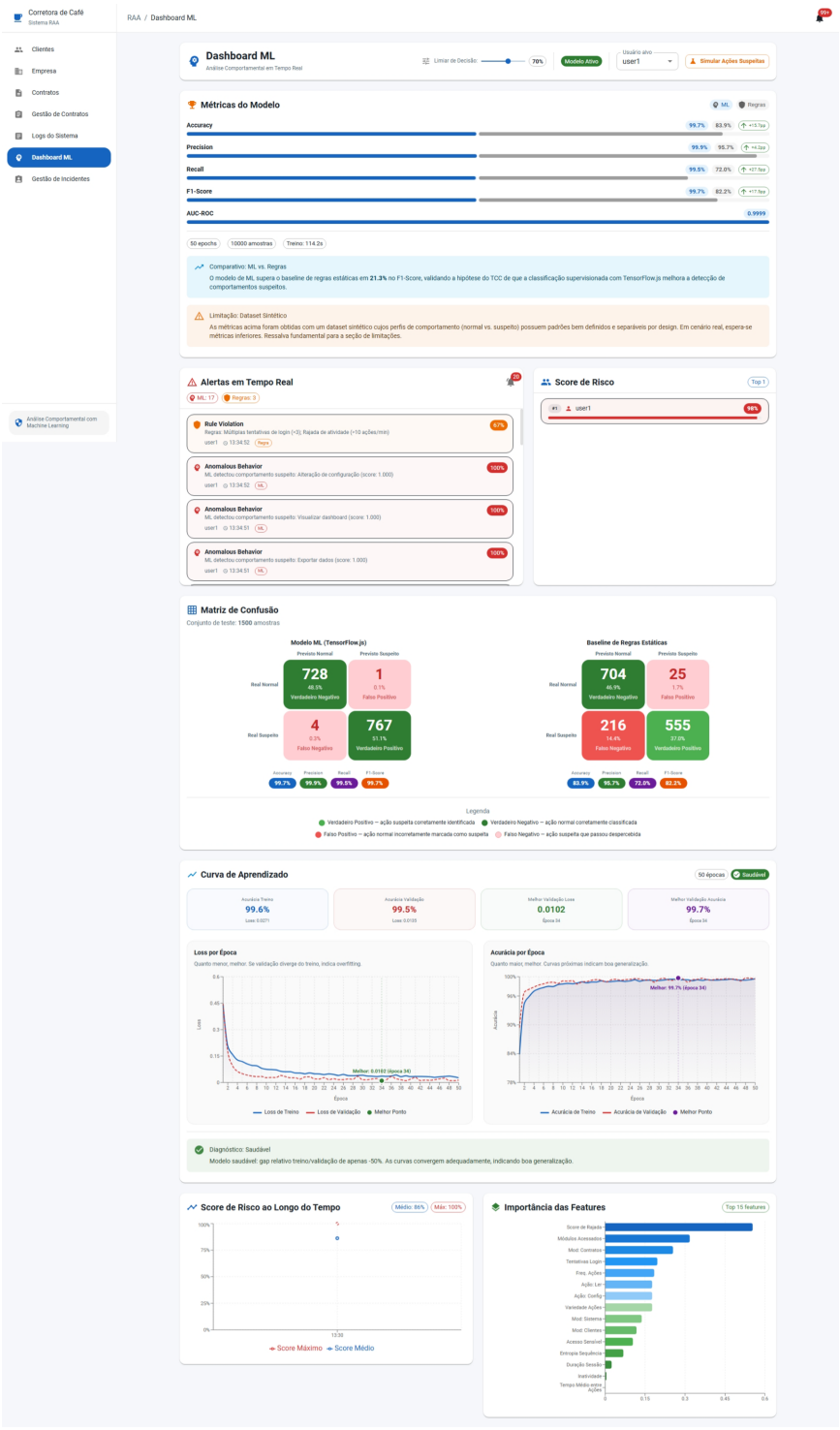

Figura 12 – *Dashboard* de métricas do modelo de Aprendizado de Máquina.

3.3.4 Visualização de Logs

O módulo de visualização de logs permite consultar todos os eventos registrados, incluindo *timestamp*, identificação do usuário, tipo de ação, módulo acessado, resultado da ação e o *score* de risco atribuído pelo modelo. A funcionalidade é o que viabiliza a auditoria e a rastreabilidade das ações. A Figura 13 mostra a tela de visualização dos logs.

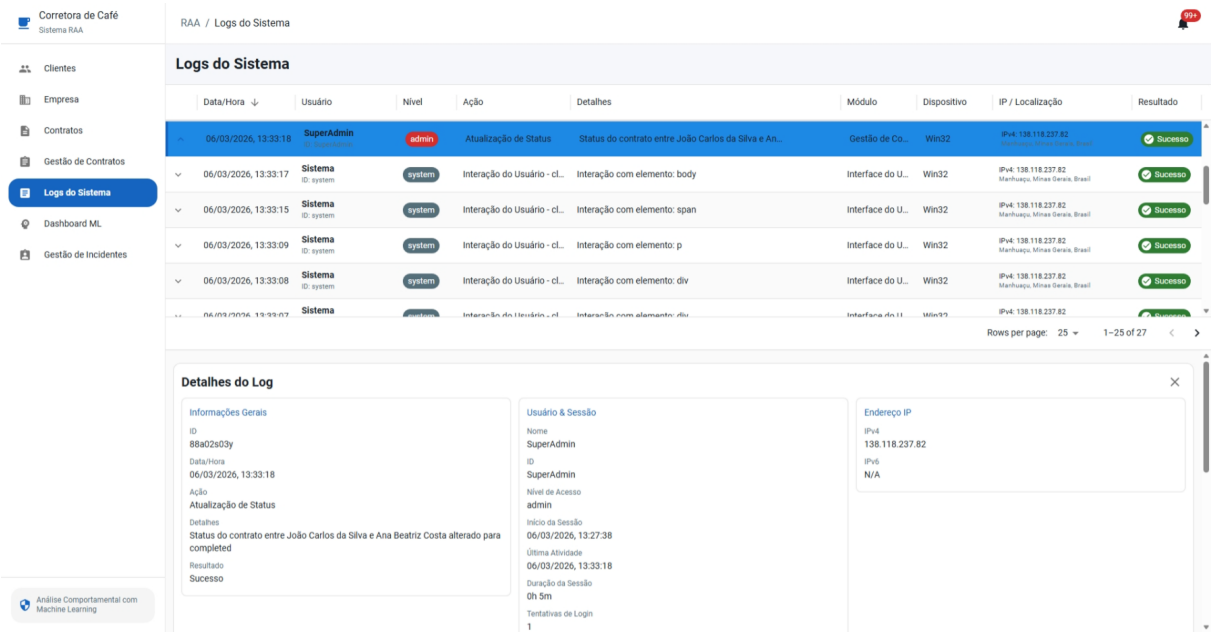

Figura 13 – Tela de visualização dos logs do sistema.

3.4 Implementação do Modelo de Aprendizado de Máquina

Esta seção apresenta os principais trechos de código relacionados ao modelo de Aprendizado de Máquina, o componente central da proposta.

3.4.1 Implementação da Arquitetura do Modelo

O modelo é uma rede neural densa com quatro camadas além da entrada. A camada de entrada recebe as 30 *features* normalizadas. Seguem-se três camadas ocultas com 64, 32 e 16 neurônios, todas com ativação ReLU, em uma configuração em funil que reduz a dimensionalidade a cada estágio. As duas primeiras camadas ocultas são acompanhadas de *Batch Normalization* e *Dropout*, com taxas de 30% e 20%, conforme o padrão de regularização descrito na Seção 2.1.3. Na saída, um único neurônio com ativação *sigmoid* devolve a probabilidade de a ação ser suspeita. O total soma 4.993 parâmetros treináveis, e a arquitetura completa está resumida na Figura 14.

Figura 14 – Arquitetura da rede neural densa: 4.993 parâmetros treináveis, otimizador Adam com taxa de aprendizado de 0,0005 e perda *binary crossentropy*.

O modelo de classificação binária foi implementado com a API sequencial do TensorFlow.js. O trecho a seguir define a arquitetura da rede neural densa com *BatchNormalization* e *Dropout*:

```
const model = tf.sequential();

model.add(tf.layers.dense({
  inputShape: [30],
  units: 64,
  activation: 'relu'
}));
model.add(tf.layers.batchNormalization());
model.add(tf.layers.dropout({ rate: 0.3 }));

model.add(tf.layers.dense({ units: 32, activation: 'relu' }));
model.add(tf.layers.batchNormalization());
model.add(tf.layers.dropout({ rate: 0.2 }));

```

```
model.add(tf.layers.dense({ units: 16, activation: 'relu' }));

model.add(tf.layers.dense({ units: 1, activation: 'sigmoid' }));

model.compile({
  optimizer: tf.train.adam(0.0005),
  loss: 'binaryCrossentropy',
  metrics: ['accuracy']
});
```

Listing 3.1 – Definição da arquitetura do modelo de classificação.

A rede recebe um vetor de 30 *features* normalizadas e devolve um valor entre 0 e 1, que representa a probabilidade de a ação ser suspeita. A ativação *sigmoid* na camada de saída faz o resultado ser interpretado como probabilidade, e o limiar de decisão (configurável, padrão 70%) define a classificação final.

3.4.2 Inferência em Tempo Real no Frontend

Um dos diferenciais da arquitetura é executar a inferência direto no navegador do usuário. Depois do treinamento no *backend*, o modelo é exportado e carregado no *frontend* com TensorFlow.js. A cada ação no sistema, as 30 *features* são extraídas, normalizadas e submetidas ao modelo para classificação imediata. A Figura 15 mostra o caminho percorrido por cada ação até a decisão.

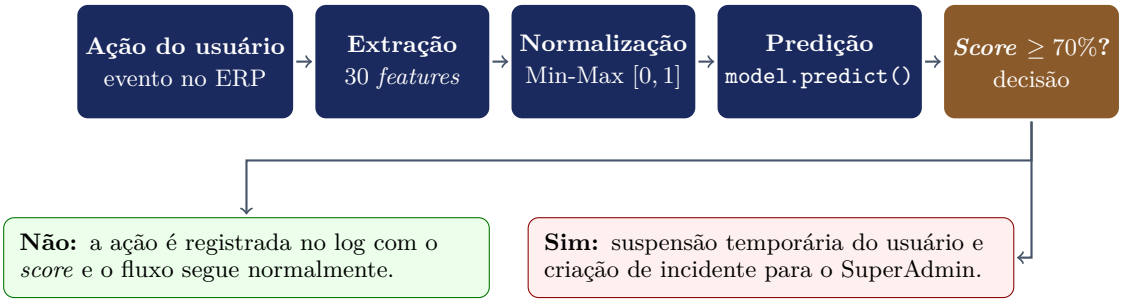

Figura 15 – Fluxo da inferência em tempo real no navegador, da ação do usuário até a decisão sobre o limiar.

```
const features = extractFeatures(userAction);
const normalized = normalizeFeatures(features, statsMin, statsMax
  );
const tensor = tf.tensor2d([normalized]);
const prediction = model.predict(tensor);
const riskScore = prediction.dataSync()[0];

```

```
if (riskScore >= RISK_THRESHOLD) {
  suspendUser(userId, 180);
  createIncident(userId, userAction, riskScore);
}
```

Listing 3.2 – Inferência em tempo real no frontend.

Quando o *score* de risco passa do limiar configurado, o usuário é suspenso automaticamente e um incidente é criado para análise do SuperAdmin, fechando o ciclo de *Human-in-the-Loop* descrito na Seção 3.3.2.

As estatísticas de normalização usadas no *frontend* são as mesmas calculadas no treinamento e são exportadas junto com cada versão do modelo. Sem essa sincronia, os dados chegariam ao modelo em uma escala diferente da que ele viu durante o treino, e a predição perderia sentido.

3.5 Considerações sobre a Implementação

A implementação priorizou três aspectos. O primeiro é a **transparência**: todas as decisões do modelo podem ser auditadas pelo painel de logs e pelo histórico de incidentes, o que dá ao administrador visibilidade completa sobre o comportamento do sistema de segurança. O segundo é a **reversibilidade**: nenhuma ação automatizada é definitiva, já que o bloqueio temporário de 3 minutos permite reverter a decisão em caso de falso positivo. O terceiro é a **adaptabilidade**: o re-treinamento com *feedback* do administrador deixa o modelo evoluir conforme o contexto da organização, sem precisar mexer no código-fonte.

O código-fonte do sistema está disponível no repositório <https://github.com/KAYOKG/PROJETO-TCC-UFU>.

Capítulo 4

Experimentos e Análise dos Resultados

Este capítulo descreve a avaliação do modelo de Aprendizado de Máquina apresentado no Capítulo 3. São detalhados a geração do conjunto de dados, a engenharia de *features*, o *baseline* de comparação e os resultados. A última seção traz uma análise crítica dos acertos, das limitações e do alcance das evidências obtidas.

4.1 Método para a Avaliação

Esta seção descreve as decisões metodológicas que estruturam os experimentos. São apresentados a geração e a organização do conjunto de dados sintéticos, a engenharia de *features* usada para representar o comportamento dos usuários e o sistema de regras estáticas adotado como *baseline*. As métricas empregadas na avaliação quantitativa, acurácia, precisão, revocação, F1-Score, AUC-ROC e matriz de confusão, foram definidas na Seção 2.4 e são retomadas aqui apenas no momento da leitura dos resultados.

4.1.1 Conjunto de Dados

Como não havia dados reais de operação do ERP de corretagem de café, foi desenvolvido um gerador de dados sintéticos capaz de simular perfis realistas de comportamento. Foram geradas 10.000 amostras no total, divididas igualmente entre oito perfis comportamentais, o que resulta em 1.250 amostras por perfil, organizados em duas classes balanceadas:

Perfis normais (50% das amostras):

- *Normal Worker*: ações CRUD regulares, horário comercial, mesmo IP e dispositivo, intervalos normais entre ações;
- *Normal Manager*: acessa mais módulos, inclui configurações, horário estendido;

- *Normal Analyst*: analista de mercado e cotações de café, com predominância de leituras em módulos de relatórios, sessões longas de análise, poucos módulos acessados e baixa taxa de erros;
- *Normal Intern*: estagiário ou funcionário recente, com poucas ações por sessão, acesso restrito a módulos básicos (cadastro e consultas), intervalos maiores entre ações e taxa de erros um pouco maior por inexperiência (erros de validação, não tentativas de acesso indevido).

Perfis suspeitos (50% das amostras):

- *Data Exfiltration*: muitas leituras consecutivas, acesso rápido a vários registros, horários incomuns. O perfil reflete os padrões de acesso anômalo identificados por Ranjan e Kumar (2022), que mostraram como leituras excessivas e sequenciais indicam tentativa de extração de dados;
- *Privilege Escalation*: tentativas de acesso a módulos não autorizados, erros frequentes, mudanças de configuração. Inspira-se nos padrões de comportamento malicioso em logs de auditoria analisados por Berlin, Slater e Saxe (2015), onde tentativas de ações fora das permissões sinalizam escalação de privilégios;
- *Account Compromise*: login de IP e localização diferentes, navegador diferente, padrões de ação fora do histórico. Baseia-se na abordagem de Jiang et al. (2018), que usou dados de comportamento do usuário para identificar mudanças de contexto de acesso como indício de conta comprometida;
- *Insider Threat*: acesso a dados sensíveis fora do padrão e atividade em horários incomuns. O perfil combina a análise de logs de sistema de Du et al. (2017) com a perspectiva de uso malicioso de ML discutida por Danziger e Henriques (2017), representando um colaborador legítimo que age de forma maliciosa.

A geração é automatizada e roda no *backend*. Para cada amostra, o gerador sorteia um dos oito perfis e produz os valores das *features* a partir das faixas características daquele comportamento. O perfil *Data Exfiltration*, por exemplo, recebe alta frequência de leituras e horários noturnos, enquanto o *Normal Worker* recebe ações em horário comercial e baixa taxa de erros. Cada amostra é então rotulada conforme a classe do perfil de origem, normal ou suspeita.

Os quatro perfis por classe representam a diversidade de comportamentos, legítimos e maliciosos, de um ambiente corporativo. Cada classe recebeu metade das amostras, decisão tomada na própria geração dos dados. Conforme discutido na Seção 2.1.1, o desbalanceamento é a situação natural em segurança, e um conjunto muito assimétrico levaria o modelo a favorecer a classe majoritária, com acurácia alta e detecção baixa. Jiang et al. (2018) enfrentaram esse cenário em dados reais, com proporção de 315.693

para 1 entre amostras normais e anômalas, e recorreram à técnica SMOTE para equilibrar o *dataset*. Como neste trabalho os dados são gerados de forma controlada, o equilíbrio foi obtido na origem, sem necessidade de *oversampling*. A contrapartida dessa escolha está registrada entre as limitações (Seção 4.3.3): em operação real a prevalência de ameaças é muito menor, e a precisão tende a cair nesse cenário.

O conjunto foi dividido em três partições: 7.000 amostras para treinamento (70%), 1.500 para validação (15%) e 1.500 para teste (15%). A proporção segue a prática descrita na Seção 2.1.1, que destina a maior parte dos dados ao ajuste dos parâmetros e reparte o restante entre o acompanhamento do *overfitting* durante o treino e a estimativa final de desempenho. As três partições são disjuntas, de modo que nenhuma amostra usada no teste participou do treinamento ou da validação. Com 1.500 amostras reservadas, cada ponto percentual das métricas corresponde a cerca de 15 classificações, resolução suficiente para distinguir o desempenho das duas abordagens. Durante a geração, ruído gaussiano foi somado aos valores das *features* para aumentar a variabilidade das amostras e evitar padrões determinísticos demais, aproximando os dados sintéticos da variação encontrada em operação real. O ruído perturba os valores dentro da faixa de cada perfil, e o rótulo continua vindo do perfil de origem, então a variabilidade aumenta sem troca de classe. O conjunto de dados completo, com as 10.000 amostras, está disponível no repositório do projeto[1] para reprodução dos experimentos.

4.1.2 Engenharia de *Features*

A partir dos logs capturados pelo sistema, foram extraídas e computadas 30 *features* numéricas, organizadas nas três categorias definidas na fundamentação teórica:

***Features* de Ambiente** (extraídas direto dos logs): hora do dia (0–23), dia da semana (0–6), nível de acesso codificado (0–3), tipo de ação codificado (*one-hot* para Create, Read, Update, Delete, Login, Config), módulo codificado, resultado da ação (0/1), duração da sessão em minutos, dispositivo e navegador codificados.

***Features* de Comportamento** (computadas por janela de tempo por usuário): frequência de ações nos últimos N minutos, variedade de tipos de ação distintos na sessão, entropia de Shannon da sequência de ações (que mede a previsibilidade do comportamento), quantidade de módulos distintos acessados, contagem de acessos a dados sensíveis, taxa de erros, tempo médio entre ações consecutivas e *burst score* (detecção de rajadas de ações em curto período).

***Features* de Contexto** (extraídas dos logs e computadas): latência de rede, tipo de rede codificado, distância geográfica da localização habitual do usuário, *flag* de mudança de IP, tentativas de login, tempo de inatividade e *flag* de dispositivo novo.

A definição dessas categorias se apoiou nos trabalhos correlatos. As *features* de ambiente contextualizam cada ação, à semelhança do DeepLog de Du et al. (2017), que

[1] <https://github.com/KAYOKG/PROJETO-TCC-UFU>

usa *timestamp* e tipo de ação para detectar anomalias em logs de sistema. As *features* de comportamento, como frequência de ações e entropia, vieram da abordagem de Bolt, de Leoni e van der Aalst (2018), que analisou variantes de processo a partir de padrões comportamentais em logs de eventos. As *features* de contexto, como mudança de IP e geolocalização, correspondem aos indicadores de comprometimento de conta identificados por Jiang et al. (2018).

Todas as *features* foram normalizadas com *Min-Max Scaling* para o intervalo $[0, 1]$, de modo que nenhuma variável domine o treinamento por diferença de escala. As estatísticas de normalização foram salvas e reutilizadas no *backend* (treinamento) e no *frontend* (inferência), o que garante consistência entre os dois ambientes.

4.1.3 Baseline de Regras Estáticas

Para confrontar a hipótese do TCC, de que a classificação supervisionada com TensorFlow.js supera os métodos tradicionais baseados em regras fixas, foi implementado um sistema de 8 regras estáticas como *baseline*:

1. Mais de 50 ações na janela de atividade;
2. Acesso fora do horário comercial (22h–6h);
3. Mais de 3 tentativas de login;
4. IP diferente do habitual;
5. Taxa de erros superior a 20%;
6. *Burst score* elevado (mais de 5 ações em 1 minuto);
7. Mais de 20 acessos a dados sensíveis;
8. Geolocalização a mais de 100 km da habitual.

A ação é classificada como suspeita quando duas ou mais regras são acionadas ao mesmo tempo. O sistema reproduz a abordagem tradicional, comum em sistemas corporativos, contra a qual o modelo de ML é comparado.

Cabe registrar a origem dessas regras, porque ela delimita o alcance da comparação. As oito heurísticas foram definidas pelo próprio autor, a partir de dois insumos: os indicadores de comprometimento apontados nos trabalhos relacionados, como as tentativas de acesso fora da permissão em Berlin, Slater e Saxe (2015) e as mudanças de contexto de acesso em Jiang et al. (2018), e os valores operacionais praticados no ERP desenvolvido. O *baseline* não é, portanto, um produto comercial consolidado nem uma implementação de referência publicada na literatura, e sim uma reconstrução das heurísticas típicas dessa abordagem.

Duas consequências decorrem disso. A primeira é que a comparação mede a distância entre o modelo e um conjunto plausível de regras fixas, não entre o modelo e o estado da arte em sistemas baseados em regras. A segunda é que os limiares poderiam ser recalibrados e alterariam o resultado do *baseline*, ainda que a natureza rígida das regras permaneça. Para reduzir o risco de favorecimento, os limiares foram fixados antes da avaliação e as duas abordagens foram submetidas exatamente ao mesmo conjunto de teste, com os mesmos rótulos. O alcance dessa evidência é retomado na Seção 4.3.1.

4.2 Experimentos

Esta seção apresenta os resultados do processo experimental: os dados de convergência do treinamento, as matrizes de confusão que detalham os erros de classificação de cada abordagem e a comparação entre o modelo de ML e o *baseline* de regras no conjunto de teste.

4.2.1 Resultados do Treinamento

O modelo, com a arquitetura descrita na Seção 3.4.1, foi treinado por 50 épocas com *batch size* de 64 sobre as 7.000 amostras de treinamento. A convergência foi consistente: a perda (*loss*) no treino caiu de 0,1192 (época 5) para 0,0271 (época 50), e a perda na validação caiu de 0,0504 para 0,0135. O treinamento completo levou cerca de 114,2 segundos. A Tabela 4 registra a evolução das métricas a cada 5 épocas, e a Figura 16 mostra as duas curvas de perda.

Tabela 4 – Evolução do treinamento ao longo das épocas.

Época	**Loss (treino)**	**Acurácia (treino)**	**Loss (validação)**	**Acurácia (validação)**
5	0,1192	97,47%	0,0504	98,27%
10	0,0751	98,49%	0,0277	99,13%
15	0,0613	98,73%	0,0271	99,00%
20	0,0496	99,00%	0,0207	99,33%
25	0,0401	99,37%	0,0162	99,60%
30	0,0412	99,23%	0,0155	99,47%
35	0,0351	99,39%	0,0162	99,53%
40	0,0335	99,40%	0,0192	99,40%
45	0,0293	99,51%	0,0155	99,53%
50	0,0271	99,59%	0,0135	99,53%

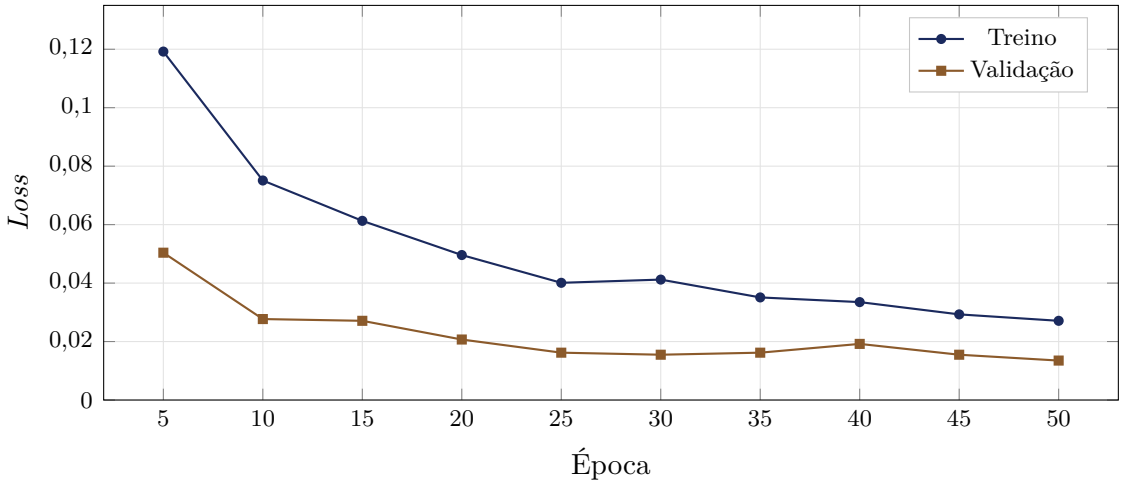

Figura 16 – Curvas de perda no treino e na validação ao longo das 50 épocas.

A acurácia de validação ficou acima de 99,33% a partir da época 20, sinal de que o modelo aprendeu os padrões relevantes sem *overfitting* severo. A pequena oscilação na perda de validação entre as épocas 25 e 50 (de 0,0135 a 0,0192) é esperada e não indica instabilidade. O modelo convergiu bem apesar da diversidade dos quatro perfis normais, o que mostra capacidade de generalizar diante de comportamentos legítimos variados.

Na Figura 16, a curva de validação aparece abaixo da curva de treino em toda a extensão. O comportamento decorre do *Dropout* e da *Batch Normalization*, que atuam durante o treino e são desligados na validação, o que reduz a perda medida nessa partição.

4.2.2 Matrizes de Confusão

As Tabelas 5 e 6 mostram as matrizes do modelo de ML e do *baseline* de regras, com a distribuição das classificações corretas e incorretas.

Tabela 5 – Matriz de confusão do modelo ML no conjunto de teste (1.500 amostras).

	Predito Normal	**Predito Suspeito**	**Total**
Real Normal	728 (VN)	1 (FP)	729
Real Suspeito	4 (FN)	767 (VP)	771

Tabela 6 – Matriz de confusão do baseline de regras estáticas no conjunto de teste (1.500 amostras).

	Predito Normal	**Predito Suspeito**	**Total**
Real Normal	704 (VN)	25 (FP)	729
Real Suspeito	216 (FN)	555 (VP)	771

As matrizes mostram que o modelo de ML produziu apenas 1 falso positivo e 4 falsos negativos, 5 erros em 1.500 amostras. O *baseline* de regras produziu 25 falsos positivos e 216 falsos negativos, 241 erros no total. A diferença mais importante está nos falsos

negativos. O modelo deixou escapar 4 ações suspeitas, enquanto as regras deixaram passar 216 ameaças sem detecção.

Na comparação com os correlatos, o desempenho das regras estáticas confirma o que Berlin, Slater e Saxe (2015) apontaram sobre o risco de falsos positivos e negativos em sistemas baseados em regras para detectar comportamento malicioso em logs de auditoria. O modelo de ML, por sua vez, teve desempenho próximo ao relatado por Jiang et al. (2018), cujo classificador XGBoost também alcançou alta precisão na detecção de ameaças internas, ainda que em um contexto de dados diferente (CMU-CERT *dataset*).

Os 25 falsos positivos do *baseline* confirmam que as regras estáticas não separam comportamentos legítimos atípicos, como os erros do estagiário, de comportamentos de fato maliciosos. O modelo de ML ficou em 1 falso positivo, o que mostra que a rede aprendeu a diferença entre os erros do perfil *Intern* e os erros ligados a *Privilege Escalation*.

4.2.3 Avaliação no Conjunto de Teste

A Tabela 7 apresenta a comparação entre o modelo de ML e o *baseline* de regras, avaliados sobre o conjunto de teste com 1.500 amostras não vistas durante o treinamento, e a Figura 17 traz os mesmos valores em forma de gráfico.

Tabela 7 – Comparativo de métricas: Modelo ML vs. Baseline de Regras Estáticas.

Métrica	**Modelo ML**	**Regras Estáticas**	**Diferença**
Acurácia	99,67%	83,93%	+15,74 p.p.
Precisão	99,87%	95,69%	+4,18 p.p.
Revocação	99,48%	71,98%	+27,50 p.p.
F1-Score	99,68%	82,16%	+17,52 p.p.
AUC-ROC	0,9999	-	-

Nota: As métricas do *baseline* foram obtidas aplicando as 8 regras estáticas (Seção 4.1.3) sobre o mesmo conjunto de teste de 1.500 amostras. O F1-Score do modelo foi extraído da saída do `model.evaluate()` do TensorFlow.js no *backend*. O do *baseline* foi calculado a partir das contagens de VP, FP e FN da Tabela 6, usando os rótulos reais do *dataset* sintético como referência.

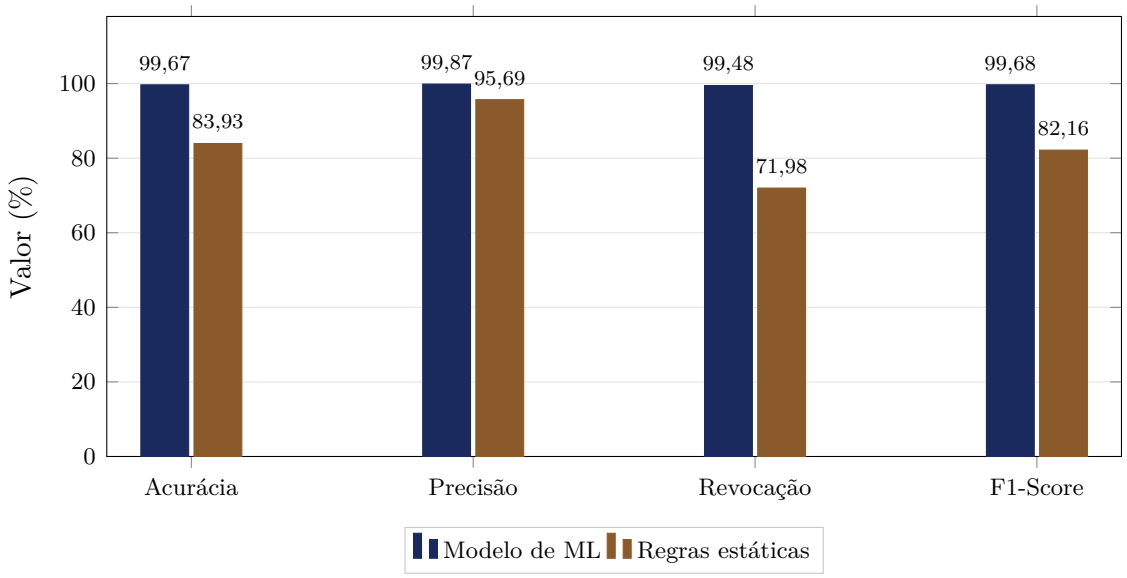

Figura 17 – Comparativo das métricas do modelo de ML e do *baseline* de regras estáticas no conjunto de teste.

O modelo de ML obteve F1-Score de 99,68% contra 82,16% do *baseline*. Isso representa uma melhoria relativa de **21,3%** sobre o valor do *baseline*, calculada da seguinte forma:

$$\frac{99{,}68 - 82{,}16}{82{,}16} \times 100 \approx 21{,}3\%$$

A diferença absoluta entre os dois valores é de 17,52 pontos percentuais, registrada na última coluna da Tabela 7. A maior diferença entre os dois métodos está na revocação, de 99,48% contra 71,98%. Na prática, as regras estáticas não detectaram cerca de 28% das ameaças do conjunto de teste.

A Acurácia do *baseline* ficou em 83,93%, 15,74 pontos percentuais abaixo da acurácia do modelo, e há um motivo específico para essa diferença. O perfil *Normal Intern* tem taxa de erros mais alta por inexperiência, o que dispara a regra de "taxa de erros superior a 20%" mesmo sem nenhuma ação maliciosa. O modelo aprendeu a separar esse padrão do *Privilege Escalation*, mas as regras fixas não conseguem fazer essa distinção. Danziger e Henriques (2017) aponta justamente esse ponto como o limite central das abordagens baseadas em regras estáticas em ambientes corporativos.

4.3 Avaliação dos Resultados

Esta seção apresenta a interpretação dos resultados sob três perspectivas. Primeiro, verifica se os dados sustentam a hipótese formulada no Capítulo 1. Depois, discute os acertos da abordagem, com as decisões de projeto que contribuíram para o desempenho. Por fim, reconhece as limitações do estudo, necessárias para contextualizar as métricas e orientar os trabalhos futuros.

4.3.1 Validação da Hipótese

A hipótese principal afirma que a classificação supervisionada com TensorFlow.js em sistemas de corretagem de café melhora a detecção de comportamentos suspeitos frente a métodos convencionais baseados em regras fixas. Os resultados obtidos são compatíveis com essa hipótese e permitem rejeitar a hipótese nula de que as duas abordagens teriam desempenho equivalente no conjunto avaliado. A diferença observada é ampla e consistente em todas as métricas, o que torna implausível atribuí-la ao acaso da partição de teste.

Sustentar a hipótese em sentido amplo, porém, exigiria condições que este trabalho não reúne. O *baseline* foi construído pelo próprio autor (Seção 4.1.3), e não representa um sistema de regras validado externamente. Os dados são sintéticos e separáveis por construção (Seção 4.3.3). A evidência produzida aqui, portanto, é a de que a abordagem supera um conjunto plausível de regras fixas sob condições controladas, e não a de que o resultado se transfere para operação real.

O modelo de ML superou o *baseline* em todas as métricas avaliadas. A melhoria relativa de 21,3% no F1-Score, cujo cálculo está detalhado na Seção 4.2.3, mostra que a classificação supervisionada é mais eficaz na detecção de comportamentos suspeitos no cenário avaliado. A diferença mais relevante aparece na revocação, de 99,48% contra 71,98%. O modelo identifica quase todas as ameaças, enquanto as regras deixam escapar cerca de 28% das ações suspeitas.

As matrizes de confusão apontam na mesma direção. Foram apenas 4 falsos negativos no modelo, contra 216 do *baseline*. Em segurança corporativa, uma ameaça não detectada pode causar prejuízo financeiro, perda de dados confidenciais ou dano à reputação, o que torna essa diferença um critério de peso na escolha entre as duas abordagens.

Por fim, o AUC-ROC de 0,9999 mostra que o modelo separa as duas classes quase perfeitamente no *dataset* avaliado e mantém alta taxa de detecção sob diferentes limiares de decisão. O valor tão próximo de 1,0 reflete a separabilidade dos perfis sintéticos e deve ser lido com a mesma ressalva das demais métricas.

4.3.2 Análise dos Acertos

As 30 *features* em três categorias (ambiente, comportamento e contexto) deram ao modelo informação suficiente para capturar padrões que regras estáticas não expressam, como a correlação entre hora do dia, frequência de ações e tipo de módulo acessado.

Os perfis *Normal Analyst* e *Normal Intern* foram um desafio a mais, porque ampliaram a diversidade de comportamentos legítimos a distinguir dos suspeitos. O *Intern*, em especial, tem características que uma abordagem ingênua confundiria com *Privilege Escalation*, como taxa de erros alta e acesso a poucos módulos. O fato de o modelo ter mantido apenas 1 falso positivo mostra que a rede aprendeu a diferença entre erro por

inexperiência e tentativa maliciosa de acesso. Essa distinção depende de correlações entre várias *features* que regras isoladas não alcançam.

A inferência em tempo real no *frontend*, descrita na Seção 3.4.2, classifica cada ação na hora, sem precisar consultar o servidor. Isso resulta em baixa latência na detecção e independência do estado da rede, características desejáveis em ambientes corporativos.

4.3.3 Limitações

A principal limitação é o uso de dados sintéticos no treinamento e na avaliação. Os perfis gerados têm padrões bem definidos e separáveis por *design*, o que facilita a classificação e produz métricas otimistas. Em produção, espera-se que comportamentos legítimos e maliciosos se sobreponham de forma mais sutil, com métricas menores do que as deste estudo. As métricas aqui devem ser lidas como indicadores do potencial da abordagem, não como estimativa de desempenho em produção.

A distribuição balanceada entre as classes é outra fonte de otimismo. Em operação real a prevalência de ameaças é baixa, e a precisão cai quando os eventos de interesse são raros, ainda que o AUC-ROC seja menos sensível a essa mudança por medir a separação entre as classes. Uma implantação real exigiria recalibrar o limiar para o novo ponto de operação.

Há ainda o limite imposto pelo próprio *baseline*, pelas razões discutidas na Seção 4.1.3: as regras foram definidas neste trabalho e não constituem uma referência externa validada.

Outra limitação é a falta de validação com usuários reais do sistema de corretagem de café. O *Human-in-the-Loop* foi implementado e testado em ambiente controlado (Seção 3.3.2), mas sua eficácia em operação real ainda precisa ser verificada, considerando volume de dados, diversidade de comportamentos e a resistência dos usuários ao monitoramento.

Por fim, o modelo não faz análise temporal de sequências, como as redes LSTM do DeepLog de Du et al. (2017), o que poderia capturar padrões sequenciais mais complexos. A arquitetura densa trata cada ação de forma relativamente independente e não modela a dependência temporal entre ações consecutivas de um mesmo usuário.

Capítulo 5

Conclusão

Este trabalho propôs e implementou um sistema de análise comportamental com Aprendizado de Máquina para detecção de atividades suspeitas em um software de gerenciamento de corretagem de café. O desenvolvimento, detalhado no Capítulo 3, foi da coleta automatizada de logs de eventos até um ciclo completo de *Human-in-the-Loop* com inferência em tempo real, passando pela engenharia de *features*, pelo treinamento do modelo e pela construção de um *dashboard* analítico.

Cada objetivo específico definido no Capítulo 1 foi alcançado ao longo do desenvolvimento:

(a) **Estudo das características comportamentais dos usuários:** as características foram investigadas e documentadas, resultando em 30 *features* numéricas em três categorias (ambiente, comportamento e contexto), conforme a fundamentação teórica e o *pipeline* de engenharia de *features* do sistema.

(b) **Análise de logs de eventos:** os logs foram examinados, estruturados e persistidos em banco SQLite, servindo de base tanto para o treinamento quanto para a inferência em tempo real. O sistema captura *timestamp*, identificação do usuário, nível de acesso, tipo de ação, detalhes, dispositivo, navegador, rede, geolocalização, IP e dados de sessão.

(c) **Desenvolvimento de um modelo de Aprendizado de Máquina:** foi construído e treinado um modelo de classificação supervisionada com TensorFlow.js, uma rede neural densa com 4.993 parâmetros, *BatchNormalization* e *Dropout*. O modelo alcançou F1-Score de 99,68% no conjunto de teste.

(d) **Validação e teste do modelo:** o modelo foi avaliado com acurácia, precisão, revocação, F1-Score e AUC-ROC, e comparado a um *baseline* de 8 regras estáticas. O F1-Score ficou 21,3% acima do *baseline* em termos relativos (99,68% contra 82,16%), diferença que permite rejeitar a hipótese nula de equivalência entre as duas abordagens no conjunto avaliado.

(e) **Proposta de medidas preventivas:** as medidas preventivas vieram na forma de bloqueio automático de usuários com comportamento suspeito (suspensão temporária de 3 minutos) e de gestão de incidentes pelo SuperAdmin, com as opções de confirmar a ameaça (encerrando a sessão do usuário) ou liberar como legítimo.

(f) **Discussão sobre implicações éticas e legais:** as implicações éticas foram consideradas no projeto. O *Human-in-the-Loop* garante que nenhum usuário seja bloqueado de forma permanente sem a análise de um administrador humano, o que preserva o direito ao contraditório e evita decisões automatizadas irreversíveis.

(g) **Recomendação de práticas de segurança:** o sistema incorporou um conjunto de práticas de segurança, como monitoramento em tempo real das ações, níveis de ameaça por usuário atualizados de forma dinâmica, painel de gestão de incidentes com histórico auditável e re-treinamento incremental do modelo com *feedback* do administrador.

5.1 Principais Contribuições

A principal contribuição é a evidência empírica, no cenário avaliado, de que modelos de classificação supervisionada com TensorFlow.js superam abordagens baseadas em regras estáticas na detecção de comportamentos suspeitos em sistemas de corretagem de café. O modelo de ML alcançou revocação de 99,48% contra 71,98% das regras estáticas, uma diferença de 27,50 pontos percentuais na capacidade de identificar ameaças. A matriz de confusão do modelo teve apenas 4 falsos negativos (ameaças não detectadas), contra 216 do *baseline*, evidência de que as regras deixam escapar cerca de 28% das ações suspeitas. O alcance dessa contribuição tem dois limites já registrados no Capítulo 4: os dados são sintéticos e separáveis por construção, e o *baseline* foi definido neste próprio trabalho, o que o distingue de uma referência externa validada.

Outra contribuição é o mecanismo de *Human-in-the-Loop*, que permite re-treinar o modelo com base nas decisões do administrador. Esse mecanismo amplia a proposta inicial da hipótese e mostra a viabilidade de um sistema de segurança que melhora com o uso e se adapta ao contexto de cada organização. A cada decisão do SuperAdmin (confirmar ameaça ou liberar como legítimo), uma nova amostra rotulada entra no *dataset* de treinamento, o que faz o modelo aprender com o contexto real da empresa.

Do ponto de vista metodológico, o trabalho apresentou um *framework* reprodutível para sistemas de análise comportamental com ML, que inclui a geração de dados sintéticos com oito perfis realistas (quatro normais e quatro suspeitos), o *pipeline* de engenharia de *features* com 30 variáveis em três categorias, a arquitetura de rede neural com *Batch-Normalization* e *Dropout*, a avaliação comparativa com *baseline* de regras e o sistema de bloqueio automático com gestão administrativa de incidentes.

No plano tecnológico, o uso de TensorFlow.js permitiu rodar a inferência direto no navegador do usuário, sem consultar o servidor a cada predição. Essa decisão dá baixa latência na detecção e independência do estado da rede, características desejáveis em ambientes corporativos.

5.2 Trabalhos Futuros

A partir das limitações da Seção 4.3.3, os trabalhos futuros a seguir podem aprimorar e expandir a proposta:

Validação com dados reais: a principal limitação é o uso de dados sintéticos. Como trabalho futuro, vale coletar e usar dados de operação real do ERP de corretagem de café no treinamento e na avaliação, o que daria métricas mais representativas do desempenho em produção. A validação poderia ser feita em parceria com corretoras de café, usando dados anonimizados.

Comparação com um *baseline* externo: confrontar o modelo com uma ferramenta de detecção já consolidada no mercado, ou com um conjunto de regras publicado na literatura, no lugar do *baseline* construído neste trabalho. A comparação daria à evidência um alcance que o *baseline* próprio não alcança.

Modelagem temporal com LSTM: o modelo atual trata cada ação de forma relativamente independente. Incorporar redes recorrentes (*Long Short-Term Memory*) para capturar padrões sequenciais nas ações dos usuários, na linha do DeepLog (DU et al., 2017), poderia melhorar a detecção de ameaças que aparecem como sequências de ações que pareceriam normais se analisadas isoladamente.

Detecção de anomalias não supervisionada: complementar o modelo supervisionado com técnicas não supervisionadas (*autoencoders*, *Isolation Forest*) para detectar comportamentos anômalos ainda não rotulados, ampliando a detecção de ameaças desconhecidas que não constam no *dataset* de treinamento.

Estudo de usabilidade do *Human-in-the-Loop*: avaliar com administradores reais a eficácia e a usabilidade do painel de gestão de incidentes, identificando melhorias na interface e no fluxo de trabalho que aumentem a produtividade do operador e a qualidade das decisões.

Análise de explicabilidade do modelo: aplicar técnicas como SHAP (*SHapley Additive exPlanations*) para dar ao administrador explicações mais detalhadas sobre por que cada ação foi classificada como suspeita, aumentando a transparência e a confiança nas decisões. Hoje o sistema mostra as *features* de maior peso na classificação, mas uma análise formal de explicabilidade traria interpretações mais robustas.

Integração com sistemas de autenticação corporativos: integrar o sistema a protocolos como LDAP ou *Single Sign-On* (SSO) para operar em ambientes corporativos reais, no lugar do sistema simplificado de credenciais fixas usado na demonstração.

Referências

AHMED, M.; MAHMOOD, A. N.; HU, J. A survey of network anomaly detection techniques. **Journal of Network and Computer Applications**, Elsevier, v. 60, p. 19–31, 2016. Citado na página 23.

BERLIN, K.; SLATER, D.; SAXE, J. Malicious behavior detection using windows audit logs. In: **Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security**. [S.l.: s.n.], 2015. p. 35–44. Citado 11 vezes nas páginas 12, 13, 19, 22, 24, 27, 28, 29, 44, 46 e 49.

BOLT, A.; de Leoni, M.; van der Aalst, W. M. Process variant comparison: Using event logs to detect differences in behavior and business rules. **Information Systems**, v. 74, p. 53–66, 2018. ISSN 0306-4379. Information Systems Engineering: selected papers from CAiSE 2016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306437916305257>. Citado 5 vezes nas páginas 13, 26, 28, 29 e 46.

BREIMAN, L. Random forests. **Machine Learning**, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 19.

BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 2, p. 1153–1176, 2016. Citado 2 vezes nas páginas 13 e 23.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM Computing Surveys**, ACM, v. 41, n. 3, p. 1–58, 2009. Citado 2 vezes nas páginas 13 e 22.

CHAWLA, N. V. et al. Smote: Synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, v. 16, p. 321–357, 2002. Citado na página 20.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: ACM. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.], 2016. p. 785–794. Citado na página 19.

DANZIGER, M.; HENRIQUES, M. A. A. Uma análise do uso de machine learning contra a segurança da informação. In: **Anais do X Encontro dos Alunos e Docentes do Departamento de Engenharia de Computação e Automação Industrial (EADCA)**. Campinas, SP: Faculdade de Engenharia Elétrica e de Computação, UNICAMP, 2017. Citado 9 vezes nas páginas 16, 19, 20, 22, 25, 27, 28, 44 e 50.

DU, M. et al. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: **Proceedings of the 2017 ACM SIGSAC conference on computer and communications security**. [S.l.: s.n.], 2017. p. 1285–1298. Citado 16 vezes nas páginas 12, 13, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 44, 45, 52 e 55.

HOMOLIAK, I. et al. Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. **ACM Computing Surveys**, ACM, v. 52, n. 2, p. 1–40, 2019. Citado 2 vezes nas páginas 12 e 23.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. **Proceedings of the 32nd International Conference on Machine Learning**. [S.l.], 2015. p. 448–456. Citado na página 21.

JIANG, W. et al. An insider threat detection method based on user behavior analysis. In: SPRINGER. **Intelligent Information Processing IX: 10th IFIP TC 12 International Conference, IIP 2018, Nanning, China, October 19-22, 2018, Proceedings 10**. [S.l.], 2018. p. 421–429. Citado 10 vezes nas páginas 13, 18, 19, 22, 27, 28, 29, 44, 46 e 49.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: **3rd International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2015. Citado na página 21.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015. Citado na página 21.

LIU, L. et al. Detecting and preventing cyber insider threats: A survey. **IEEE Communications Surveys & Tutorials**, IEEE, v. 20, n. 2, p. 1397–1417, 2018. Citado 2 vezes nas páginas 12 e 23.

RANJAN, R.; KUMAR, S. S. User behaviour analysis using data analytics and machine learning to predict malicious user versus legitimate user. **High-Confidence Computing**, v. 2, n. 1, p. 100034, 2022. ISSN 2667-2952. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2667295221000246>. Citado 10 vezes nas páginas 12, 13, 18, 19, 20, 22, 26, 28, 29 e 44.

SHANNON, C. E. A mathematical theory of communication. **The Bell System Technical Journal**, Nokia Bell Labs, v. 27, n. 3, p. 379–423, 1948. Citado na página 24.

SMILKOV, D. et al. Tensorflow.js: Machine learning for the web and beyond. **Proceedings of Machine Learning and Systems**, v. 1, p. 309–321, 2019. Citado na página 32.

SOMMER, R.; PAXSON, V. Outside the closed world: On using machine learning for network intrusion detection. **Proceedings of the IEEE Symposium on Security and Privacy**, IEEE, p. 305–316, 2010. Citado 2 vezes nas páginas 16 e 23.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR.org, v. 15, n. 1, p. 1929–1958, 2014. Citado na página 21.

YUAN, S.; WU, X. Deep learning for insider threat detection: Review, challenges and opportunities. **Computers & Security**, Elsevier, v. 104, p. 102221, 2021. Citado na página 21.