DEPARTAMENTO DE ENGENHARIAS E CIÊNCIAS DA COMPUTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA E DE TELECOMUNICAÇÕES

UNIVERSIDADE AUTÓNOMA DE LISBOA

"LUÍS DE CAMÕES"

APLICAÇÃO DE INTELIGÊNCIA ARTIFICIAL E BLOCKCHAIN PARA A CONDUÇÃO AUTÓNOMA

Dissertação para a obtenção do grau de Mestre em Engenharia Informática e de Telecomunicações

Autor: Gonçalo Filipe Tendeiro Rodrigues de Lemos

Orientador: Professor Doutor Héctor Dave Orrillo Ascama

Número do candidato: 30007523

Janeiro de 2026

Lisboa

Dedicatória

Esta Dissertação é dedicada, de forma especial, à minha família, que sempre me ajudou durante a realização deste trabalho.

Dedico-a, em particular, aos meus pais e alguns amigos próximos, que sempre me apoiaram, guiaram e encorajaram.

A todos dedico este trabalho.

Agradecimentos

Em primeiro lugar, agradece-se, de forma incondicional, todo o apoio prestado pelo Professor Orientador, Héctor Dave Orrillo Ascama, ao longo do desenvolvimento desta dissertação.

Destaca-se, em particular, a disponibilidade constante, o acompanhamento rigoroso e a orientação metodológica fornecida em todas as etapas do trabalho, desde a clarificação dos conceitos fundamentais até à definição e validação das opções técnicas adotadas.

Salienta-se, igualmente, a capacidade de enquadrar criticamente os objetivos e resultados obtidos, contribuindo para uma evolução sustentada do projeto e para a consolidação do raciocínio científico subjacente.

Para além disso, reconhece-se o contributo decisivo do Professor Orientador na estruturação e revisão da componente escrita, através de recomendações objetivas e exigentes, que permitiram melhorar a coerência, a precisão terminológica e a qualidade global do documento.

Agradece-se também o apoio prestado pelos meus pais e familiares próximos durante o percurso no Mestrado.

Sem o apoio todo recebido, não teria sido possível realizar este trabalho.

Epígrafe

"As cousas árduas e lustrosas se alcançam com trabalho e com fadiga."

– Luís de Camões

Resumo

Esta dissertação propõe uma arquitetura de defesa contra ataques em *VANETs*, utilizando a *Federated Learning* (Aprendizagem Federada – *FL*) e a *Blockchain*. Apesar dos benefícios proporcionados pelas *VANETs*, é necessário desenvolver soluções que permitam que o sistema seja robusto e íntegro.

A arquitetura proposta integra o *LSTM* (*Long Short-Term Memory*), que é treinado localmente nos *Workers* (veículos), para realizar previsões de velocidade e latência.

A Aprendizagem Federada realiza o treino colaborativo sem a necessidade de partilha de dados brutos, e a *Blockchain* garante maior privacidade e controlo de acesso.

As duas estratégias de defesa, "Distância Euclidiana" e "Divergência *KL*", são implementadas para detetar ataques, como o ataque estático, o ataque de pretensão estático, o ataque de pretensão aleatória e o ataque progressivo.

A arquitetura, que inclui *Roadside Units* (Unidades de Apoio Rodoviário – *RSUs*), nós *Workers* e um nó Central (*Chief*), é testada em cenários de simulação implementados com ferramentas como o *SUMO, o NS-3* (módulo *LTE/EPC*, com parametrização *5G-like*) e o *TensorFlow*.

Os resultados obtidos avaliam o desempenho das estratégias de defesa (Distância Euclidiana e Divergência *KL*), com e sem a *Blockchain*.

Palavras-chave: *VANET*, Aprendizagem Federada, *LSTM*, *Blockchain*.

Abstract

This dissertation proposes a defense architecture against attacks on VANETs, using Federated Learning (FL) and Blockchain. Despite the benefits provided by VANETs, it is necessary to develop solutions that allow the system to be robust and reliable.

The proposed architecture integrates LSTM (Long Short-Term Memory), which is trained locally on Workers (vehicles), to perform speed and latency predictions.

Federated Learning enables collaborative training without the need to share raw data, and Blockchain ensures greater privacy and access control.

The two defense strategies, "Euclidean Distance" and "KL Divergence", are implemented to detect attacks such as static attacks, static pretension attacks, random pretension attacks, and forward attacks.

The architecture, which includes Roadside Units (RSUs), Worker nodes, and a Central (Chief) node, is tested in simulation scenarios implemented with tools such as SUMO, NS-3 (LTE/EPC module, with 5G-like parameterization), and TensorFlow.

The results obtained evaluate the performance of the defense strategies (Euclidean Distance and KL Divergence), with and without Blockchain.

***Keywords**: VANET, Federated Learning, LSTM, Blockchain.*

Resumen

Esta tesis propone una arquitectura de defensa contra ataques a VANET mediante Aprendizaje Federado (FL) y Blockchain. A pesar de los beneficios que ofrecen las VANET, es necesario desarrollar soluciones que permitan que el sistema sea robusto y confiable.

La arquitectura propuesta integra LSTM (Memoria a Largo Plazo y Corto Plazo), que se entrena localmente en los nodos trabajadores (vehículos), para realizar predicciones de velocidad y latencia.

El Aprendizaje Federado realiza entrenamiento colaborativo sin necesidad de compartir datos sin procesar, y Blockchain garantiza una mayor privacidad y control de acceso.

Las dos estrategias de defensa, "Distancia Euclidiana" y "Divergencia KL", se implementan para detectar ataques como ataques estáticos, ataques de pretensión estática, ataques de pretensión aleatoria y ataques hacia adelante.

La arquitectura, que incluye Unidades de Carretera (RSU), nodos trabajadores y un nodo central (principal), se prueba en escenarios de simulación implementados con herramientas como SUMO, NS-3 (módulo LTE/EPC con parametrización similar a la de 5G) y TensorFlow.

Los resultados obtenidos evalúan el desempeño de las estrategias de defensa ("Distancia Euclidiana" y "Divergencia KL"), con y sin Blockchain.

***Palabras clave**: VANET, Aprendizaje Federado, LSTM, blockchain.*

Índice

Índice de Figuras

Índice de tabelas

Lista de Siglas e Acrónimos

IA	Artificial Intelligence
BL	Blockchain
DID	Decentralized Identifier
FL	Federated Learning
FedAvg	Federated Averaging
Indy	Hyperledger Indy
ITS	Intelligent Transport System
JSON	JavaScript Object Notation
KL	Kullback-Leibler
LSTM	Long Short-Term Memory
MANET	Mobile Ad hoc Network
NS-3	Network Simulator 3
OBU	On-Board Unit
RSU	Roadside Unit
SUMO	Simulation of Urban Mobility
TA	Trusted Authority
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VANET	Vehicular Ad hoc Network
5G	Fifth Generation
LTE	Long-Term Evolution
RNN	Recurrent Neural Network
DNN	Deep Neural Network

GPS	Global Positioning System
URL	Uniform Resource Locator

1 Introdução

Apesar das vantagens disponibilizadas pela Aprendizagem Federada, a natureza distribuída destes sistemas torna-os vulneráveis a ataques de envenenamento de modelo.

Entre estes ataques, incluem-se os ataques estáticos, os ataques de pretensão e os ataques progressivos, que podem comprometer a integridade dos dados.

Esta dissertação propõe uma arquitetura integrada que combina Aprendizagem Federada, *LSTM*, *BL* e duas estratégias de defesa (Distância Euclidiana e Divergência *KL*), com o objetivo de reforçar a segurança e a confiabilidade das redes veiculares.

Estas tecnologias contribuem para o desenvolvimento de sistemas de transporte mais robustos.

No contexto da arquitetura proposta, colocam-se os seguintes problemas de investigação:

- Como garantir integridade e segurança em *VANETs* face a ataques estáticos, ataques de pretensão e ataques progressivos, que degradam o comportamento dos Workers e a consistência das atualizações?
- Como detetar e mitigar estes ataques numa arquitetura que integra Aprendizagem Federada, *LSTM* e *BL*?

1.1 Problema científico

Esta dissertação aborda o problema da robustez da Aprendizagem Federada, no contexto das *VANETs,* perante ataques de envenenamento de modelo.

Neste trabalho, considera-se um sistema federado composto por um conjunto de vinte *Workers*, onde um número específico de *Workers,* em cada cenário, se comporta de forma maliciosa ao enviar atualizações adulteradas dos parâmetros do modelo global.

O problema científico pode ser formulado da seguinte forma:

Dado um processo de agregação federada baseado em *FedAvg*, é possível detetar e mitigar, com eficácia, atualizações maliciosas, utilizando métricas como a Distância Euclidiana e a Divergência *KL*, sem comprometer, de forma significativa, a convergência do modelo global?

Deste modo, estabelecem-se as seguintes hipóteses:

Hipótese 1: A utilização conjunta da Distância Euclidiana e da Divergência *KL* permite identificar comportamentos maliciosos, em diversos cenários de envenenamento de modelo, com maior precisão do que a utilização de uma única métrica.

Hipótese 2: A introdução de uma camada de *BL* para controlo de identidade e auditoria não afeta, de forma significativa, o desempenho do treino federado.

1.2 Objetivos

Neste capítulo, são apresentados os objetivos da presente dissertação.

O objetivo principal consiste em criar e testar uma arquitetura para *VANETs* que utilize a Aprendizagem Federada, o *LSTM* e a *BL,* bem como desenvolver duas estratégias de defesa para reforçar a segurança e a integridade da arquitetura e detetar *Workers* maliciosos durante a execução da simulação.

De forma mais específica, esta dissertação pretende:

- Implementar um modelo *LSTM* em contexto de Aprendizagem Federada, de modo a permitir previsões da velocidade e da latência, sem que os *Workers* enviem dados brutos.
- Integrar a *BL* no sistema, com o objetivo de reforçar a privacidade e suportar o controlo de acesso.
- Testar, durante as simulações, os cenários com e sem *BL*.
- Realizar ataques estáticos, ataques de pretensão e ataques progressivos em ambiente controlado.

  Para construir este ambiente, é utilizado o *Simulation of Urban MObility*(Simulação de Mobilidade Urbana - *SUMO*) e o *Network Simulator 3* (Simulador de rede 3 - *NS-3*), com o módulo *Long-Term Evolution* (Evolução de Longo Prazo - *LTE*)/*Evolved Packet Core* (O Núcleo de Pacote Evoluído - *EPC*), com parametrização "*5G-like"*.
- Utilizar as duas estratégias de defesa (Distância Euclidiana e Divergência *KL*) para detetar *Workers* maliciosos.
- Avaliar, no final da simulação, o desempenho das defesas com e sem *BL*.

1.3 Justificação

Para permitir comunicações entre veículos e infraestruturas, são necessários sistemas inteligentes capazes de operar de forma distribuída, segura e resiliente.

A Aprendizagem Federada promove descentralização e privacidade, mas permanece vulnerável a ataques maliciosos que podem comprometer a integridade do treino e a qualidade do modelo global.

Neste contexto, este trabalho propõe uma solução integrada que combina *LSTM* e Aprendizagem Federada com *BL* e estratégias de defesa, com o objetivo de detetar e mitigar diferentes padrões de ataque.

Esta dissertação tem relevância por contribuir para o avanço do conhecimento da segurança em sistemas distribuídos, bem como por apoiar o desenvolvimento de sistemas de transporte mais fiáveis e resistentes a ataques.

1.4 Estrutura da Tese

Esta dissertação está organizada em cinco capítulos:

- Capítulo 1 – Introdução: Neste capítulo, são introduzidos o tema e o enquadramento do problema, definem-se os objetivos do trabalho, apresenta-se a justificação da sua realização e descrevem-se a metodologia adotada e a estrutura global do documento.
- Capítulo 2 – Fundamentação Teórica: Neste capítulo, são apresentadas a fundamentação teórica e a revisão da literatura, abordando-se as redes veiculares e as suas arquiteturas, a aplicação de Inteligência Artificial nas *VANETs*, o papel da *BL* nas *VANETs*, as vulnerabilidades presentes na Aprendizagem Federada e as defesas baseadas na Distância Euclidiana e na Divergência *KL*.
- Capítulo 3 – Metodologia: Neste capítulo, é descrito o modelo desenvolvido, detalhando-se a arquitetura e os mecanismos de privacidade e segurança suportados por *BL*, bem como as estratégias de deteção de ataques baseadas em comportamento.
- Capítulo 4 – Resultados obtidos e Análise Experimental: Neste capítulo, são apresentados os resultados obtidos após a execução da simulação completa, incluindo-se a análise dos relatórios e dos gráficos gerados e a interpretação do impacto das defesas e da *BL* nos diferentes cenários.
- Por último, Capítulo 5 – Conclusões: Neste capítulo, é apresentada a síntese das conclusões do trabalho, identificam-se as limitações encontradas e propõem-se sugestões para trabalhos futuros.

1.5 Ferramentas e tecnologias utilizadas

A abordagem adotada nesta investigação baseou-se na execução de simulações, de modo a avaliar o comportamento do sistema em quatro cenários, com e sem *BL*.

O desenvolvimento e a execução foram realizados predominantemente, em *Python*, em conjunto com ferramentas de simulação, incluindo a implementação de componentes no *NS-3,* desenvolvidos em *C++*.

No ambiente computacional, as simulações foram executadas num equipamento com *Microsoft Windows 11 Home, 64 bits* (compilação 26200), processador *Intel Core i7-10510U @ 1.80 GHz* (4 núcleos, 8 *threads*), placa gráfica *NVIDIA GeForce MX250,* com 2 GB de *VRAM* dedicada, 16 *GB* de *RAM* e 1 *TB* de armazenamento.

Para permitir a utilização de determinadas ferramentas e garantir compatibilidade de dependências, recorreu-se à virtualização. A máquina virtual utilizada foi a *Oracle VirtualBox,* com o *Ubuntu 20.04.6 LTS* (*focal*), configurada com 14 *GB* de *RAM*, 3 *vCPUs* e 90 *GB* de armazenamento em disco virtual.

As principais tecnologias utilizadas na execução das simulações foram:

• *SUMO,* para simular a mobilidade veicular, gerando trajetórias, velocidades e padrões de circulação realistas.

• *NS-3,* para simular as comunicações em rede, modelando tráfego, latências e condições de transmissão.

• *TraCI,* para integrar o *SUMO* com o *NS-3*, permitindo a troca de dados e o controlo da simulação em tempo real.

• *Flower,* para realizar a Aprendizagem Federada, coordenando as rondas de treino e a agregação entre *Workers* e o servidor.

• *TensorFlow/Keras,* para treinar o modelo, disponibilizando a implementação e a otimização de redes *LSTM*.

• *Pandas/OpenPyXL,* para gerar os relatórios em formato *Excel*, organizando e exportando métricas e os resultados de forma automática.

• *Matplotlib,* para gerar os gráficos, visualizando a evolução de métricas e as comparações entre cenários.

• *Hyperledger Indy,* para gerir as identidades de forma descentralizada, suportando *Decentralized Identifier* (Identificador Descentralizado - *DIDs)* e credenciais verificáveis.

• *Hyperledger Fabric,* para efetuar registo e auditoria de atualizações, garantindo a rastreabilidade e a integridade das operações.

A execução das simulações foi automatizada através de scripts de orquestração em *Python*, responsáveis por executar os cenários com e sem *BL* e, no final, gerar relatórios e gráficos com base nos dados obtidos.

2 Fundamentação Teórica

Neste capítulo, são apresentados conceitos teóricos sobre *VANETs*, Aprendizagem Federada, *LSTM* e *BL,* de modo a fornecer uma compreensão sólida dos diferentes componentes presentes na arquitetura, bem como das soluções propostas nesta dissertação.

2.1 Redes Veiculares: Conceitos e Arquiteturas

As *Vehicular Ad Hoc Networks* (Redes Ad-Hoc Veiculares - *VANETs*) são uma tecnologia relevante no contexto dos *Intelligent Transport Systems* (Sistemas de Transporte Inteligentes - *ITS*)[1], [2]. Estas redes constituem uma subclasse das *Mobile Ad Hoc Networks* (Rede *Ad Hoc* Móvel - *MANETs*)[1].

Os três principais componentes das *VANETs* são a *Trusted Authority* (Autoridade confiável - *TA*), as *Roadside Units* (Unidades de Beira de Estrada - *RSUs*) e as *On-Board Units* (Unidades de bordo - *OBUs*)[1], [3], [4], [5], [6].

Estes três componentes podem ser descritos da seguinte forma:

1. *TA*: Responsável pela gestão de todo o sistema.
2. *RSU*: Estação base instalada nas vias públicas, que permite aos veículos comunicar com a infraestrutura.
3. *OBU*: Tecnologia instalada nos veículos, que permite a comunicação entre veículos e com a infraestrutura.

2.1.1 Tipos de comunicações nas *VANETs*

Nas *VANETs,* existem dois tipos principais de comunicação, referenciados na literatura [1], [7], [8], [9]:

1. *Vehicle-to-Vehicle* (Veículo para Veículo - *V2V*): Nesta comunicação, os veículos trocam mensagens entre si. O *V2V* permite a partilha de informação relevante em situações inesperadas, como acidentes rodoviários.

2. *Vehicle-to-Infrastructure* (Veículo para Infraestrutura - *V2I*): Nesta comunicação, os veículos e as infraestruturas trocam mensagens entre si.

Esta comunicação é utilizada, sobretudo, para prevenir e mitigar acidentes, bem como para melhorar a segurança e a eficiência dos trajetos.

Nas *VANETs*, existe ainda outro tipo de comunicação, denominado *Vehicle-to-Everything* (Veículo-para-Tudo – *V2X*), que possibilita a troca de informação entre os veículos e quaisquer entidades relevantes no ambiente rodoviário, tais como outros veículos, infraestruturas, peões e dispositivos.

Este tipo de comunicação engloba o *V2V* e o *V2I*[10], [11].

2.1.2 Características e desafios presentes nas *VANETs*

As *VANETs* apresentam características próprias que as distinguem de outros tipos de rede, tais como:

- Elevada Mobilidade: Os nós nas *VANETs* alteram frequentemente a posição, pelo que a topologia e a localização da rede variam com regularidade[1], [7], [8], [9].
- Potência de transmissão limitada: O *Wireless Access in Vehicular Environments* (Acesso sem fios em ambientes veiculares - *WAVE)* apresenta potência de transmissão com cobertura entre 0 e 28,8 *dBm* (*decibel-miliwatt*) e um alcance máximo de 1 km, o que restringe significativamente a capacidade de cobertura[1], [12].
- Comunicação sem fios: Os nós nas *VANETs* podem trocar informação e estabelecer ligações, sendo, contudo, necessário garantir mecanismos que assegurem comunicação segura durante a transmissão[1], [13].

As características das *VANETs* conferem elevado potencial e capacidade de inovação, mas introduzem desafios que necessitam de ser superados, nomeadamente:

- Redes Veiculares heterogéneas: O desenvolvimento de tecnologias de comunicação sem fios conduziu à coexistência de múltiplas soluções, muitas vezes incompatíveis.
  Esta heterogeneidade dificulta a garantia de comunicação robusta, sendo necessário desenvolver mecanismos que assegurem interoperabilidade entre tecnologias[1], [14], [15].
- Gestão e armazenamento de dados: O elevado número de veículos origina sobrecarga na gestão e no armazenamento, tornando necessária a adoção de soluções de armazenamento distribuído e seguro[1], [16].

- Fragmentação da Rede: A fragmentação pode tornar alguns nós inacessíveis, sendo este problema particularmente evidente em zonas com semáforos ou em zonas rurais, quando a densidade de nós é reduzida[1].
- Incompatibilidade com Protocolos de Encaminhamento Tradicionais: A elevada mobilidade e as alterações frequentes na topologia tornam os protocolos tradicionais ineficientes, pelo que se torna necessária a conceção de algoritmos e protocolos de encaminhamento resilientes, de modo a melhorar o desempenho e as taxas de entrega de pacotes[1], [17].

2.2 Inteligência Artificial em Sistemas Veiculares

Os sistemas veiculares, como as *VANETs,* podem auxiliar significativamente os condutores durante os trajetos, contribuindo para uma melhor gestão do combustível, para trajetos mais eficientes e para uma redução significativa de acidentes[18], [19].

As *VANETs* produzem uma grande quantidade de dados e são vulneráveis a diferentes tipos de ataques.

Para responder a estes desafios, diversos investigadores propuseram a utilização de técnicas de Inteligência Artificial.

A integração de Inteligência Artificial com *VANETs* apresenta benefícios relevantes, tais como[18]:

- Melhoria da capacidade de deteção e prevenção de ataques maliciosos[18], [20], [21], [22], [23].
- Melhoria da análise e classificação de *malware*[18], [24], [25].
- Aumento da privacidade e da integridade dos dados[18], [26], [27], [28], [29].

2.2.1 Aprendizagem Federada

Atualmente, uma das principais preocupações consiste em prevenir a violação de dados sensíveis.

Os governos têm vindo a estabelecer regulamentos para proteção dos dados dos utilizadores, aplicando multas significativas em caso de incumprimento[30].

Neste contexto, a Aprendizagem Federada tem-se destacado como uma solução para mitigar problemas de privacidade.

A Aprendizagem Federada permite que múltiplos participantes treinem, de forma colaborativa, um modelo de aprendizagem automática, sem que seja necessário enviar dados locais para uma entidade central[30].

Esta abordagem tem sido aplicada em diversas áreas, nomeadamente em sistemas distribuídos e aprendizagem automática, contribuindo para o reforço da privacidade[30], [31], [32].

Na Aprendizagem Federada, existem três componentes principais[30], [31]:

1. Clientes/*Workers*: Entidades detentoras dos dados, responsáveis pelo treino local e beneficiárias do modelo resultante.
2. Servidor: Entidade responsável por manter e atualizar o modelo global e por gerir a comunicação com os *Workers*.
3. Estrutura de comunicação e computação: Infraestrutura que suporta a troca de parâmetros do modelo e a execução do treino local.

O processo típico de Aprendizagem Federada ocorre da seguinte forma[31]:

- O servidor seleciona os clientes que cumprem requisitos de elegibilidade.
- O servidor envia o modelo global mais recente aos clientes selecionados.
- Os clientes treinam com os seus próprios dados e enviam apenas as atualizações do modelo.
- O servidor agrega as atualizações recebidas e atualiza o modelo global.
- O processo repete-se ao longo de múltiplas iterações.

2.2.2 Agregação Federada

A *Federated Averaging* (Agregação Federada - *FedAvg*) é um dos algoritmos mais utilizados em Aprendizagem Federada.

Neste algoritmo, os clientes treinam localmente o mesmo modelo e um servidor central agrega as atualizações através de média ponderada dos parâmetros.

O servidor não recebe dados brutos[33], [34], [35].

O objetivo principal consiste em preservar a privacidade.

Em cada ronda, são executados os seguintes processos[33], [34], [36], [37], [38]:

1. Seleção de clientes: O servidor escolhe um subconjunto de clientes.
2. Envio do modelo global: O servidor envia os pesos atuais aos clientes.
3. Treino local: Cada cliente treina o modelo localmente.
4. Agregação por média ponderada: O servidor atualiza o modelo global com base nas atualizações recebidas.

2.2.3 Vantagens do *FedAvg* nas *VANETs*

A integração do *FedAvg* em *VANETs* apresenta, sobretudo, as seguintes vantagens[39], [40], [41], [42], [43], [44]:

- Maior privacidade nos veículos: Dados sensíveis, como a trajetória e o estado do veículo, permanecem localmente, sendo enviados apenas parâmetros do modelo. Assim, reduz-se o risco de fuga direta de dados.
- Simplicidade e escalabilidade: O *FedAvg* é de implementação relativamente simples em arquiteturas com *RSUs* e *OBUs* e mantém escalabilidade, mesmo com um elevado número de veículos.
- Compatibilidade com arquiteturas típicas de *VANETs*: O *FedAvg* é compatível com diferentes modelos de *VANETs*, promovendo utilização mais eficiente de recursos e facilitando integração com camadas adicionais de segurança, como criptografia, assinatura em lote e agregação segura.

2.2.4 Aplicação da Aprendizagem Federada na dissertação

Nesta dissertação, a Aprendizagem Federada é utilizada para manter os dados de treino nos *Workers*, que enviam ao *Chief* apenas as atualizações do modelo resultantes do treino local. Este processo decorre ao longo de várias rondas:

- Em cada ronda, o *Chief* distribui o modelo global mais recente pelos *Workers*.
- Após a receção do modelo, os *Workers* realizam treino local com os seus dados e enviam apenas as atualizações dos pesos.
- Por fim, o *Chief* agrega as atualizações recebidas através do *FedAvg* e atualiza o modelo global.
- O processo repete-se até se verificar convergência.

2.2.5 *LSTM*

O *LSTM* é uma arquitetura de *Recurrent Neural Network* (Rede Neuronal Recorrente - *RNN*) que integra células de memória, concebidas para reter informação ao longo do tempo. Cada célula utiliza três portas[45]:

1. Porta de entrada: Controla a quantidade de informação nova introduzida na memória[46], [47].
2. Porta de esquecimento: Determina que parte do estado anterior deve ser mantida ou eliminada.
    - Se o valor da porta se aproxima de 1, a célula preserva a informação.
    - Se se aproxima de 0, a informação é descartada e deixa de influenciar os passos temporais seguintes[46], [47], [48].
3. Porta de saída: Define que parte do conteúdo da memória interna é utilizada para produzir a saída no respetivo passo temporal[46], [47], [48].

Em função da relevância da informação, as portas regulam o que é bloqueado e o que é transmitido.

Adicionalmente, a informação é ponderada pelo conjunto de pesos associado às células durante a sua propagação[45].

A arquitetura *LSTM* permite retropropagação no tempo, ajustando pesos, de modo a reduzir o erro de previsão[45].

A Figura 1 apresenta o funcionamento de uma célula *LSTM*.

A célula recebe três entradas:

1) Os dados de entrada no instante atual.
2) O estado oculto anterior.
3) O estado da célula anterior.

Com base nestas entradas, são calculadas três "portas":

- A primeira determina a informação a esquecer.
- A segunda define a informação nova a acrescentar.
- A terceira controla a parcela do conteúdo final que é exposta na saída.

É igualmente gerado um conteúdo candidato (com ativação *tanh*), que pode ser incorporado no estado da célula.

O novo estado da célula resulta da combinação entre o que é preservado do estado anterior e o que é adicionado a partir do conteúdo candidato, através de operações ponto a ponto (soma e multiplicação).

Por fim, o novo estado oculto é obtido ao aplicar *tanh* ao novo estado da célula e ao filtrar o resultado pela porta de saída.

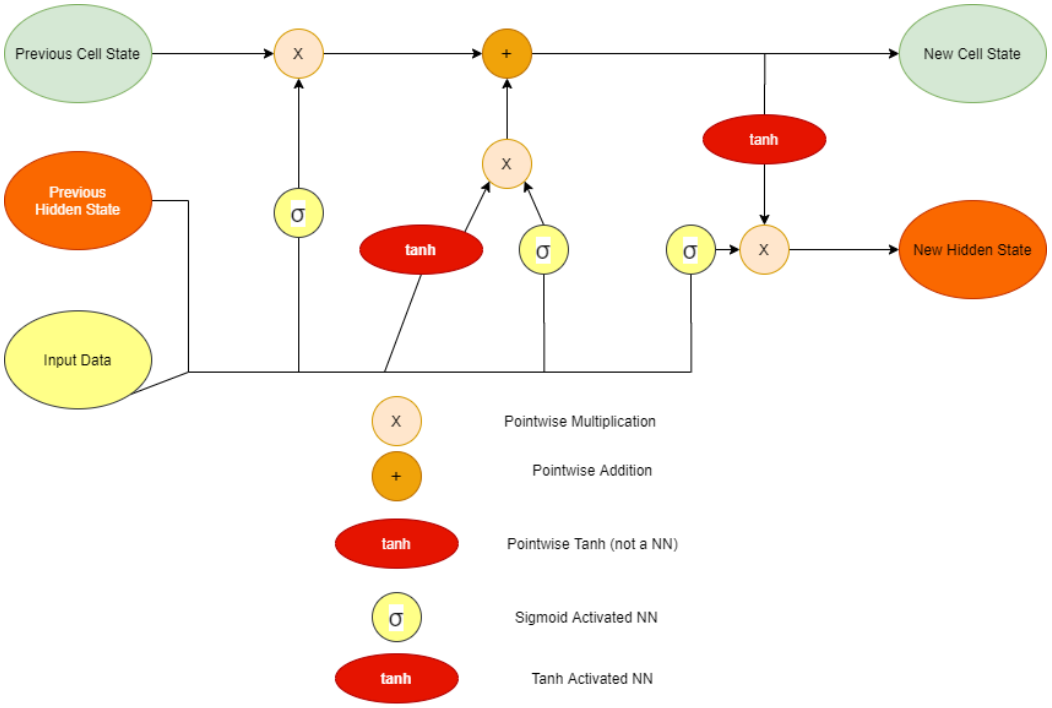

Figura 1 - Funcionamento do LSTM (Fonte: Elaboração Própria)

2.2.6 Vantagens do *LSTM* nas *VANETs*

O *LSTM* é amplamente utilizado em *VANETs*. A integração destas tecnologias apresenta as seguintes vantagens[49], [50], [51], [52], [53], [54], [55]:

- Captura de dependências temporais longas e não lineares: O *LSTM* permite modelar padrões temporais complexos, contribuindo para previsões mais precisas do tráfego e da densidade.
- Melhoria da precisão do tráfego e do fluxo de rede: O *LSTM* permite estimar tráfego e fluxo de rede, podendo apoiar a localização de veículos, mesmo em situações de indisponibilidade de *Global Positioning System* (Sistema de Posicionamento Global - *GPS*).

- Previsão de trajetórias e mobilidade: O *LSTM* permite prever trajetórias, revelando-se útil para encaminhamento e gestão de mobilidade.
- Deteção de intrusão e comportamentos maliciosos: O *LSTM* pode contribuir para a deteção de comportamentos maliciosos em *VANETs*, explorando padrões espaço-temporais nas comunicações veiculares.

2.2.7 Aplicação do *LSTM* na dissertação

Nesta dissertação, o *LSTM* foi selecionado como a arquitetura de rede neuronal utilizada no contexto da Aprendizagem Federada:

- Todos os *Workers* treinam localmente um modelo *LSTM* idêntico, utilizando dados de tráfego, nomeadamente velocidade e latência.
- Os dados são organizados em sequências temporais compostas por pares [velocidade, latência], permitindo ao *LSTM* prever os valores no instante seguinte.
- No final de cada ronda, cada *Worker* envia ao *Chief* apenas as atualizações dos pesos, e não os dados brutos.
- O *Chief* agrega as atualizações recebidas e atualiza o modelo global.

2.3 Definição da *BL* e respetivas características

A *BL* pode ser definida como um livro-razão distribuído e imutável de transações, replicado e distribuído pelos nós participantes da rede[18], [56], [57].

Cada bloco da cadeia contém, tipicamente, um *hash* criptográfico do bloco anterior, um *timestamp* e um conjunto de transações, frequentemente representados através de uma árvore de *Merkle*[18], [58].

A *BL* integra ainda um algoritmo de consenso, que permite aos nós participantes chegarem a acordo quanto à validação e à ordem das transações registadas[18], [57].

As principais características da *BL* incluem[18], [56], [59]:

- Integridade.
- Imutabilidade.
- Descentralização.
- Irreversibilidade.

- Persistência.
- Transparência.
- Anonimato.

2.3.1 Vantagens da *BL* nas *VANETs*

A integração de *BL* em *VANETs* apresenta as seguintes vantagens[18], [60], [61], [62], [63]:

- Partilha de informação distribuída: A distribuição da informação reduz a dependência de uma entidade central.
- Resistência à adulteração (imutabilidade): A *BL* reduz o risco de manipulação de registos e de dados.
- Armazenamento mais seguro e transparente: A *BL* permite armazenar dados de forma segura e transparente.
- Canais de comunicação mais fiáveis: Ao disponibilizar validação e registo distribuído, a *BL* contribui para canais de comunicação mais fiáveis.
- Maior segurança nas comunicações: A *BL pode* mitigar ataques e modificações não autorizadas.

2.3.2 Aplicação da *BL* na dissertação

Nesta dissertação, a *BL* é utilizada como uma camada de controlo de acesso e de auditoria durante a Aprendizagem Federada.

1) Quando o sistema executa a simulação "com *BL*", é inicializado um cliente *Hyperledger Fabric* através do ficheiro *network.json*, é importada a identidade *Admin* e esse cliente é associado ao *RSU*.

   Deste modo, o *RSU* pode invocar o *chaincode* no canal e utilizar o resultado como critério de autorização, aceitando ou rejeitando a participação e a submissão de atualizações de pesos pelos *Workers*.
2) Para além do *Hyperledger Fabric*, é também utilizado o *Hyperledger Indy,* com o objetivo de suportar identificação e prova de autoria. Nesta arquitetura, cada *Worker* cria um *DID* e mantém uma *wallet* local.

A ligação à rede *Indy* é estabelecida através da abertura do *pool* a partir de um ficheiro *genesis* (transações iniciais), que contém a configuração dos nós do *ledger*.

Este ficheiro é descarregado através de um *Uniform Resource Locator (*Localizador Uniforme de Recursos - *URL)* e guardado localmente, de modo a inicializar a comunicação com o *pool*.

Para demonstrar a posse do *DID*, o *Worker* assina um *nonce* (desafio), e o *RSU* valida criptograficamente essa assinatura.

Em conjunto, estas duas tecnologias permitem garantir que apenas entidades autenticadas e autorizadas interagem com a infraestrutura.

2.4 Ataques e Vulnerabilidades no sistema

Nesta dissertação, foram desenvolvidos cenários de ataque nos quais certos *Workers* participam legitimamente na execução, mas procuram degradar o modelo global, através do envio de atualizações manipuladas.

Em termos práticos, foram implementados quatro cenários de ataque (T1 – Ataque estático, T2 – Ataque de pretensão estático, T3 – Ataque de pretensão aleatória e T4 – Ataque progressivo), executados por vinte *Workers*.

Nos cenários T1 e T2, existe um Worker malicioso e dezanove *Workers* fidedignos, enquanto, nos cenários T3 e T4, existem cinco *Workers* maliciosos e quinze *Workers* fidedignos.

Do ponto de vista da vulnerabilidade explorada, os ataques correspondem a envenenamento de modelo.

Após o treino local, o *Worker* malicioso envia ao *Chief* uma versão adulterada dos pesos do modelo, isto é, a atualização local é degradada antes de ser integrada.

Esta alteração prejudica a agregação no *Chief*, uma vez que a atualização global passa a refletir contributos incoerentes com a tendência média do treino fidedigno.

Como consequência, podem ocorrer instabilidade, degradação da capacidade de generalização e redução do desempenho ao longo das rondas.

Importa referir que o sistema implementado nesta dissertação agrega os vetores de pesos resultantes do treino local.

Assim, manter a integridade e a consistência estatística destas atualizações constitui um requisito de elevada importância para a robustez do processo.

2.4.1 Ataque estático (T1)

No cenário T1, o *Worker* malicioso comporta-se de forma maliciosa durante todo o treino federado.

No código, o ataque T1 é implementado através da introdução sistemática de ruído nos tensores de pesos gerados após o treino local, produzindo atualizações recorrentemente desviantes face ao modelo global.

Uma vez que o ataque é contínuo, observa-se um padrão de anomalia relativamente estável.

As métricas de discrepância mantêm valores elevados, permitindo avaliar a capacidade das defesas para isolar o *Worker* malicioso.

2.4.2 Ataque de pretensão estático (T2)

No cenário T2, o *Worker* malicioso apresenta comportamento fidedigno nas primeiras rondas e, após um determinado período, passa a enviar atualizações maliciosas.

No código, a fase inicial corresponde às rondas em que o *Worker* devolve pesos não manipulados e, a partir de um ponto predefinido, as atualizações passam a ser adulteradas com uma perturbação mais intensa do que no cenário T1.

O ataque T2 é particularmente relevante em sistemas federados, por explorar a fragilidade de mecanismos de confiança que assumem que bom comportamento inicial implica comportamento fidedigno no futuro.

Ao iniciar de forma benigna, o *Worker* cria uma perceção de normalidade (*baseline* operacional) e, subsequentemente, altera o comportamento, para maximizar o impacto quando o sistema já se encontra estabilizado e dependente do histórico do *Worker*.

2.4.3 Ataque de pretensão aleatória (T3)

No cenário T3, o comportamento dos *Workers* maliciosos alterna, de forma aleatória, entre benigno e malicioso.

No código, em cada ronda, o *Worker* pode adulterar os pesos ou enviar a atualização normal resultante do treino local.

Este cenário é relevante por dificultar a deteção baseada exclusivamente em regras de limiar simples.

Ao intercalar rondas benignas, o *Worker* reduz a evidência acumulada e procura evitar padrões persistentes de violação.

O objetivo do ataque T3 consiste em introduzir ruído intermitente na agregação global, provocando degradação gradual, sem gerar um sinal anómalo constante.

2.4.4 Ataque progressivo (T4)

No cenário T4, o *Worker* ataca de forma subtil e cumulativa.

Em vez de introduzir um desvio elevado de forma imediata, cada *Worker* malicioso inicia o ataque de forma gradual e aumenta a intensidade ao longo das rondas.

Adicionalmente, a ativação do comportamento malicioso é faseada: apenas os *Workers* com id < *max_attackers* são considerados atacantes e cada um passa a atuar maliciosamente a partir da ronda em que *round_num* >= *worker.id*, o que faz com que os atacantes "entrem" gradualmente ao longo da execução.

No código, a perturbação aplicada aos pesos aumenta com o número das rondas, sendo limitada por um valor máximo.

O ataque T4 também é relevante por representar um cenário em que *Workers* maliciosos procuram evitar deteções reativas, explorando a tendência dos sistemas federados para tolerarem pequenas variações naturais entre atualizações.

O objetivo deste ataque T4 consiste em não gerar um *outlier* imediato, mas induzir uma tendência de degradação por ronda que aparente ser fidedigna.

As configurações dos cenários de ataque são apresentadas na tabela 1.

Tabela 1 - Configurações dos cenários de ataque

Cenário de ataque	Designação	N.º de Workers maliciosos	N. º de Workers fidedignos
T1	Ataque estático	1	19
T2	Ataque de pretensão estático	1	19
T3	Ataque de pretensão aleatória	5	15
T4	Ataque progressivo	5	15

A tabela 1 descreve as configurações e a informação considerada para a execução das simulações realizadas neste trabalho.

2.5 Estratégias de Defesa (Distância Euclidiana e Divergência *KL*)

Para mitigar o efeito das atualizações maliciosas, foram implementadas, nesta dissertação, duas estratégias complementares no *Chief* (Distância Euclidiana e Divergência *KL*).

Estas defesas procuram identificar atualizações inconsistentes com o estado atual do treino federado, assumindo-se que as atualizações fidedignas apresentam discrepâncias relativamente estáveis, enquanto as atualizações maliciosas tendem a exibir desvios superiores e/ou alterações abruptas no comportamento estatístico ao longo das rondas.

No código, as atualizações recebidas são convertidas num vetor achatado de pesos, permitindo o cálculo uniforme de métricas de discrepância.

O *Chief* mantém, para cada *Worker,* um histórico das métricas e do estado (fidedigno/suspeito/isolado), e a decisão baseia-se não apenas no valor de uma ronda, mas também no comportamento observado ao longo do tempo.

A Distância Euclidiana é amplamente utilizada em *VANETs,* para medir proximidade espacial entre veículos, sendo fundamental para a formação de *clusters* estáveis e eficientes, bem como para melhorar o encaminhamento e a disseminação de dados.

Esta métrica é simples e intuitiva, reduzindo a complexidade do processo de seleção de nós e contribuindo para a estabilidade da rede[64], [65], [66].

Por sua vez, a Divergência *KL* é útil na comparação de distribuições de probabilidade, podendo ser aplicada na deteção de anomalias e ataques cibernéticos em *VANETs*[67], [68].

A utilização conjunta destas duas defesas (Distância Euclidiana e Divergência *KL*) permite explorar vantagens complementares.

A Distância Euclidiana quantifica discrepâncias diretas entre vetores de pesos, enquanto a Divergência *KL* avalia diferenças sob uma perspetiva estatística.

Em conjunto, estas métricas contribuem para uma maior robustez na deteção de anomalias e para a otimização do desempenho da rede[67].

2.5.1 Distância Euclidiana

A Distância Euclidiana mede a discrepância entre o vetor de pesos do modelo global e o vetor de pesos enviado pelo *Worker,* em cada ronda.

Quando um *Worker* executa o treino local de forma fidedigna, espera-se que o seu modelo local permaneça relativamente próximo do modelo global, no início da ronda.

Em contraste, quando um *Worker* atua de forma maliciosa, os pesos podem ser adulterados para produzir atualizações com deslocação anómala, resultando numa distância significativamente superior entre o modelo global e o modelo submetido.

Para reduzir falsos positivos (situações em que o sistema assinala um Worker como malicioso, sendo este fidedigno), o *Chief* não se baseia num único valor isolado, uma vez que mantém um histórico por *Worker* e avalia o comportamento ao longo das rondas.

Quando se verificam violações repetidas dos limiares definidos para o cenário, o *Worker* é marcado como suspeito e, caso acumule um número predefinido de violações, é isolado.

2.5.2 Divergência *KL*

A Divergência *KL* avalia as diferenças entre a atualização do *Worker* e o modelo global, através de uma perspetiva estatística.

Para esse efeito, os vetores de pesos são transformados em distribuições normalizadas, sendo calculada a Divergência *KL* entre essas distribuições.

Tal como na Distância Euclidiana, a decisão não depende de um único valor num determinado instante.

O *Chief* mantém um histórico da Divergência *KL* por *Worker* e compara o valor atual com as tendências recentes, permitindo sinalizar comportamentos persistentemente anómalos e transições entre comportamento fidedigno e malicioso.

Esta capacidade é particularmente relevante nos cenários T2 e T3.

2.5.3 Decisão do *Chief*

As duas defesas (Distância Euclidiana e Divergência *KL*) são aplicadas de forma coordenada no *Chief*.

Para cada atualização recebida, são calculadas as métricas e é atualizado o registo do *Worker*, com base no histórico e no contador de violações.

O sistema suporta estados operacionais, como fidedigno, suspeito e isolado.

Quando um *Worker* excede critérios definidos, tais como violações repetidas, picos súbitos, persistência de valores elevados ou tendência cumulativa de degradação, o *Worker* é isolado e as suas atualizações passam a ser excluídas da agregação em rondas futuras.

O isolamento corresponde, assim, à interrupção da contribuição desse *Worker* para o cálculo do modelo global, preservando o treino federado face a atualizações maliciosas.

2.5.4 Relevância das defesas em relação aos quatro cenários

A utilização conjunta das duas defesas (Distância Euclidiana e Divergência *KL*) permite detetar diferentes tipos de ataque:

- Ataques contínuos e com elevada discrepância (T1).
- Ataques com mudança abrupta após uma fase inicial benigna (T2).
- Ataques com alternância aleatória entre comportamento fidedigno e malicioso (T3).
- Ataques com degradação lenta e progressiva (T4).

Importa salientar que, nos cenários T2 e T4, que apresentam maior sofisticação, o *Chief* analisa o histórico dos *Workers,* para detetar mudanças de regime e tendências que podem não ultrapassar, de imediato, limiares simples numa única ronda.

As defesas implementadas procuram, assim, equilibrar robustez na deteção de *Workers* maliciosos e prudência, de modo a reduzir falsos positivos.

2.6 Revisão de literatura

As *VANETs* têm sido uma tecnologia inovadora no contexto da mobilidade inteligente, devido à possibilidade de suportar comunicações *V2V* e *V2I*.

Estas comunicações podem reduzir o risco de acidentes, tornar as viagens mais económicas e eficientes, entre outras vantagens.

Contudo, apesar dos benefícios associados a esta tecnologia, persistem vulnerabilidades, nomeadamente:

- Elevada heterogeneidade da rede.
- Dificuldade de armazenamento dos dados produzidos.
- Problemas de privacidade que necessitam de ser resolvidos.

Entre as várias soluções propostas na literatura, a Inteligência Artificial e a *BL* têm assumido particular relevância.

Nesta revisão de literatura, apresentam-se as vantagens da integração destas duas tecnologias, bem como alguns estudos representativos.

2.6.1 Vantagens da Integração da Inteligência Artificial com a *BL*

A combinação de *IA* com a *BL*, no contexto das *VANETs*, apresenta vantagens que podem ser agrupadas em quatro categorias principais: comunicação segura**,** privacidade e integridade de dados**,** aprendizagem colaborativa e deteção colaborativa de intrusões.

1. Comunicação segura

   A *BL*, devido às suas características de imutabilidade, integridade e descentralização, constitui uma abordagem adequada para mitigar problemas de segurança em *VANETs*.

   Ao integrar técnicas de *IA*, torna-se possível otimizar processos na *BL* e melhorar a eficiência dos mecanismos de deteção, contribuindo para a criação de canais de comunicação seguros[27], [60].

Exemplos de estudos:

- Num estudo, os autores utilizaram *Deep Neural Networks* (Rede Neuronal Profunda - *DNNs*) e a *BL* para responder a desafios de segurança em *VANETs*.

  A *BL* foi utilizada para garantir comunicação segura entre as *VANETs* e as *RSUs*, enquanto as *DNNs* foram aplicadas para detetar comportamentos anómalos.

  Com base nessas deteções, o estado dos componentes é atualizado para "inválido" na *BL*, impedindo que outros componentes se associem a dispositivos comprometidos[69].
- Noutro estudo, foi integrada Inteligência Artificial e *BL* numa única rede, com o objetivo de abordar questões de segurança durante a condução e proteger dados sensíveis em *VANETs*.

  Foi utilizado um *LSTM* para suportar condução autónoma segura, através da análise de dados veiculares temporais.

  Ao nível de clusters de *RSUs*, foi implementada uma *BL*.

  Os autores reportaram elevada precisão de previsão, com valores entre 92,5% e 93,8%, superiores aos obtidos em alguns estudos baseados em abordagens centralizadas mais tradicionais[70].

2. Privacidade e Integridade dos dados

A integração de *BL* e *IA* permite proteger os dados gerados pelas *VANETs* e suportar o seu acesso, de forma descentralizada[71], [72].

Exemplos de estudos:

- Num estudo, foram integradas Inteligência Artificial e *BL* com o objetivo de preservar a privacidade dos dados trocados entre veículos e *RSUs*.

  A *BL* foi utilizada para criar canais de partilha de informações transparentes e fiáveis entre veículos e *RSUs*, enquanto a Inteligência Artificial foi aplicada para compreender necessidades dos veículos e prever requisitos de conteúdo relevante para *RSUs* ou veículos[72].

3. Aprendizagem Colaborativa

A literatura apresenta diversos estudos que exploram a *BL* para processar dados e conhecimento resultante de aprendizagem automática, de forma distribuída, reduzindo a dependência de conjuntos de dados centralizados e de entidades únicas[27], [73], [74].

Exemplos de estudos:

- Num estudo, os autores combinaram a *BL* com um algoritmo de Aprendizagem Federada, para reforçar segurança e privacidade durante a partilha de conhecimento.

  Os veículos aprendem informação ambiental através de métodos de aprendizagem automática e partilham o conhecimento adquirido através da *BL*.

  Adicionalmente, a *BL* foi utilizada para mitigar ataques maliciosos.

  Os autores indicaram redução do consumo computacional face a sistemas de *BL* convencionais, bem como melhoria na precisão da aprendizagem[75].

- Outro estudo propôs uma solução de aprendizagem coletiva baseada em *BL*, na qual cada veículo partilha o modelo local aprendido como conhecimento, melhorando eficiência e precisão.

  Nesta abordagem, a *BL* substitui o servidor central, tornando o processo de aprendizagem coletiva mais seguro, automático e transparente[76].

4. Deteção Colaborativa de Intrusões

A Aprendizagem Federada e a *BL* podem ser utilizadas em conjunto, para criar sistemas distribuídos e colaborativos de deteção de intrusões em *VANETs*.

A Aprendizagem Federada evita a partilha direta de dados sensíveis, ao permitir treino local dos modelos.

A *BL,* por sua vez, reduz a dependência de um servidor central e reforça a confiança no processo.

Exemplos de estudos:

- Num estudo, foi proposto um mecanismo distribuído de deteção de intrusões, baseado em *Deep Learning* (Aprendizagem Profunda - *DL*), que distribui o treino do modelo por dispositivos de ponta (veículos conectados e *RSUs*), utilizando Aprendizagem Federada e *BL*[77].
- Noutro estudo, foi introduzido um sistema de deteção de intrusões, baseado em *Deep Learning*, para detetar ciberataques em *VANETs*.

  O sistema proposto foi integrado numa estrutura de inteligência *edge* gerida por *BL*, na qual os nós de ponta (veículos e *RSUs*) participam, de forma colaborativa e fiável, no treino federado do mecanismo de deteção[78].

2.6.2 Limitações da Inteligência Artificial e da *BL*

Apesar de a *IA* e a *BL* constituírem soluções inovadoras em *VANETs*, ambas apresentam limitações que necessitam de ser consideradas, para assegurar eficiência e robustez.

Estas limitações podem ser agrupadas em duas categorias: escalabilidade e armazenamento, e vulnerabilidades de segurança.

1. Escalabilidade e Armazenamento

A descentralização da *BL* contribui para elevados níveis de segurança. No entanto, essa mesma descentralização pode limitar a escalabilidade.

Exemplos de estudos:

- Num estudo, verificou-se que o aumento do número de nós e de transações tende a aumentar a necessidade de armazenamento.

Adicionalmente, fatores como taxa de transações, tamanho de blocos, sobrecarga da comunicação e latência podem restringir a escalabilidade do sistema[79].

2. Vulnerabilidades de segurança

Alguns estudos indicam que a *BL* e a Inteligência Artificial permanecem vulneráveis a determinados tipos de ataques cibernéticos, sobretudo no que diz respeito ao processamento de dados sensíveis[18], [57], [80], [81].

Exemplos de estudos:

- Três estudos analisaram, de forma aprofundada, vulnerabilidades da *BL* no contexto da condução autónoma[27], [57], [61].
- Outros estudos detalharam vulnerabilidades associadas à *IA* em *VANETs*[82], [83].

Em síntese:

A literatura indica que a integração de *IA* com *BL* em *VANETs* reforça a segurança das comunicações e a proteção de dados, ao combinar mecanismos de integridade, rastreabilidade e validação distribuída com técnicas de deteção e aprendizagem distribuída, incluindo Aprendizagem Federada.

Os estudos analisados exemplificam estas vantagens, através de modelos baseados em *DNN/LSTM* e de abordagens colaborativas, suportadas por infraestruturas de *BL* que aumentam a confiança operacional entre veículos e *RSUs*.

Contudo, estes benefícios coexistem com limitações relevantes, sobretudo ao nível da escalabilidade e do *overhead* introduzido por requisitos de armazenamento, comunicação e latência, que tendem a agravar-se com o crescimento do número de nós e de transações.

Adicionalmente, apesar de a *BL* reforçar a auditabilidade e mitigar manipulações específicas, persistem vulnerabilidades e vetores de ataque que podem afetar tanto componentes de *BL* como componentes de *IA*, tornando necessário um desenho arquitetural que equilibre robustez, privacidade e desempenho, suportado por mecanismos explícitos de mitigação e controlo de custos operacionais.

3 Metodologia

Neste capítulo, descreve-se o modelo proposto e o respetivo funcionamento. A arquitetura desenvolvida nesta dissertação tem como objetivos:

- Gerar dados realistas num contexto de redes veiculares.
- Treinar um modelo preditivo através de Aprendizagem Federada.
- Mitigar o impacto de contribuições maliciosas, através de mecanismos de deteção e isolamento.

Em termos operacionais, a arquitetura integra um conjunto de *Workers* (veículos), responsáveis pelo treino local, e um *Chief,* que centraliza a agregação e aplica as defesas.

Em cada cenário de ataque, é comparado o modo com e sem *BL*, permitindo comparações diretas entre configurações.

Para executar o sistema de forma integrada, optou-se por automatizar o processo através de um *runner* em Python (*run_all_scenarios.py*), que invoca, para cada cenário, o *pipeline* principal (*run_simulation.py*).

O (*run_simulation.py*) recebe como argumentos o cenário, o número de *Workers* e o número de rondas (vinte *Workers* e trinta rondas).

Em cada execução, é criada uma diretoria com o formato *simulation_runs/<cenário>_<timestamp>/*, onde são armazenados os *logs* do servidor *Flower*, dos clientes e da simulação *SUMO/NS-3*, bem como os ficheiros de resultados.

Esta automatização facilita a reprodutibilidade, assegura separação clara entre cada execução e evita a necessidade de múltiplos comandos por parte do utilizador.

A arquitetura pode ser descrita como um *pipeline* coordenado pelo *runner*.

O servidor *Flower* é iniciado no início do processo e permanece ativo, de seguida, ocorre a recolha de dados por simulação e, por fim, o treino federado nos clientes é iniciado apenas após estarem concluídos os *datasets* locais.

Na fase de recolha de dados, procede-se à geração e recolha de dados através de uma simulação que utiliza o *SUMO* (geração da mobilidade) e o *NS-3* (rede), integrados por um módulo *bridge* em *Python* (*sumo_bridge_data_collector.py*).

Durante a simulação:

- O *bridge* atua como servidor local *Transmission Control Protocol (*Protocolo de Controlo de Transmissão - *TCP)*, em 127.0.0.1:5555.
- Recolhe a velocidade diretamente no *SUMO,* via *TraCI*, e uma estimativa de latência devolvida pelo *NS-3,* em cada iteração (valor agregado por passo).

As variáveis recolhidas (velocidade e latência) são agregadas e convertidas em séries temporais, a partir das quais se constroem janelas deslizantes com *SEQ_LEN=2* e duas *features* (velocidade e latência).

Deste modo, cada amostra de entrada corresponde a dois instantes consecutivos do par [velocidade, latência], e o objetivo do *LSTM* consiste em prever o par no instante seguinte.

Após a recolha, os dados são particionados e guardados num ficheiro associado a cada *Worker,* com o formato *worker_<id>.pkl,* na diretoria *simulation_runs/<cenário>_<timestamp>/worker_data/*.

Esta diretoria contém conjuntos de treino e validação *(hold-out)*, guardados como *X, y, X_test, y_test,* utilizando tamanhos fixos *(N_TRAIN=256, N_VAL=64)*.

Quando a exportação é concluída, o *bridge* cria um ficheiro *DATA_READY.json* na diretoria da execução, sinalizando que os dados locais necessários ao treino federado estão disponíveis.

Na fase federada, o *runner* inicia, em primeiro lugar, o servidor *Flower* num processo separado, que fica a escutar em 0.0.0.0:8080.

A disponibilidade do servidor é verificada pelo *runner* através de uma ligação *TCP* a 127.0.0.1:8080.

Após a conclusão da recolha *SUMO/NS-3* e apenas depois de detetado o ficheiro *DATA_READY.json,* bem como a existência dos *ficheiros .pkl,* o *runner* inicia os clientes *Flower*.

Cada cliente carrega o ficheiro *worker_<id>.pkl* correspondente (da execução mais recente em *simulation_runs*), inicializa o modelo *LSTM* e executa o treino local por rondas.

Em cada ronda, após o treino local, o *Worker* envia ao servidor apenas as atualizações dos pesos e o número de exemplos utilizados.

No lado do servidor, o *Chief* recebe as contribuições, aplica as defesas (Distância Euclidiana e Divergência *KL*) e calcula um *trust score*.

Com base nessas informações, decide quais são as atualizações elegíveis para a agregação.

Este procedimento evita que o *FedAvg* seja executado de forma indiferenciada, incorporando mecanismos de segurança que reduzem a influência de atualizações maliciosas.

Adicionalmente, o *Chief* pode marcar e isolar *Workers* quando se verificam comportamentos maliciosos persistentes.

3.1 Visão Geral da Arquitetura Integrada

A arquitetura desenvolvida nesta dissertação foi concebida para cumprir três objetivos complementares:

1) Gerar dados realistas, num contexto de redes veiculares, através de simulação de mobilidade e de rede, utilizando o *SUMO* e o *NS-3*.
2) Utilizar a Aprendizagem Federada para realizar o treino distribuído de um modelo preditivo baseado em *LSTM*.
3) Detetar e eliminar contribuições maliciosas, de modo a reduzir o impacto das atualizações maliciosas no modelo global.

Para facilitar a reprodutibilidade e a execução experimental, foi desenvolvido um *runner* (*run_all_scenarios.py*) que executa automaticamente os quatro cenários de ataque (T1 a T4), sendo que, em cada cenário, são consideradas duas configurações (sem *BL* e com *BL*).

A configuração com *BL* é ativada através do argumento --blockchain no ficheiro (*run_simulation.py*).

Em cada execução, é criada uma diretoria dedicada, com o formato *simulation_runs/<cenário>_<timestamp>/*, onde são armazenados os *logs*, os dados gerados e os artefactos necessários para a Aprendizagem Federada.

A execução completa do sistema é coordenada pelo ficheiro (*run_simulation.py*), que organiza o processo em sete etapas sequenciais:

1. Preparação do ambiente e do *logging*.
2. Arranque do servidor federado.

3. Recolha de dados por simulação no *SUMO* e no NS-3, através do *bridge* em *Python*.
4. Validação da disponibilidade dos ficheiros de dados, através do *DATA_READY.json*.
5. Lançamento dos clientes *Flower,* como processos independentes.
6. Espera pela conclusão do treino federado.
7. Recolha e análise dos resultados.

Esta separação por etapas é vantajosa, uma vez que garante que os *Workers* iniciam o treino apenas quando os respetivos dados locais se encontram completos e consistentes.

Importa referir que, embora existam diversos estudos que combinam Aprendizagem Federada e *BL* em *VANETs*, a maioria desses trabalhos foca-se na proteção da privacidade ou na substituição do servidor central por uma infraestrutura distribuída.

O contributo desta dissertação não consiste na criação de um novo algoritmo de agregação, mas na proposta de uma arquitetura integrada e funcional que combina os seguintes elementos:

1 Desenvolvimento de quatro cenários distintos de envenenamento de modelo (estático (T1), pretensão estático (T2), pretensão aleatória (T3) e progressivo (T4)).
2 Aplicação conjunta de duas métricas de defesa (Distância Euclidiana e Divergência *KL*).
3 Implementação de tecnologias como *SUMO*, NS-3 e *Flower*.
4 Camada de identidade e auditoria baseada em *Hyperledger Indy* e *Hyperledger Fabric*.
5 Avaliação comparativa dos cenários com e sem *BL*.

O fator distintivo desta dissertação, em comparação com outros trabalhos, reside na avaliação sistemática da robustez comportamental ao longo das rondas, através de duas métricas de defesa (Distância Euclidiana e Divergência *KL*), bem como na possibilidade de comparar, de forma direta, os diferentes cenários com e sem *BL*.

3.1.1 Funções dos diferentes componentes presentes na arquitetura

A arquitetura proposta é constituída por três componentes principais: *Workers*, *Chief* e *RSU*.

De seguida, descrevem-se as funções desempenhadas por cada um destes componentes no contexto da Aprendizagem Federada e da simulação considerada.

1) *Workers*

Os *Workers* correspondem às entidades que participam na Aprendizagem Federada.

A cada *Worker* é associado um identificador (*worker_id*) e um conjunto de dados local, exportado na fase de simulação para um ficheiro com o formato *worker_<id>.pkl*.

Na fase do treino federado, cada *Worker* carrega localmente o respetivo ficheiro *.pkl*, inicializa um modelo *LSTM,* idêntico para todos os participantes, e executa o treino local ao longo de trinta rondas.

Em cada ronda, após o treino local, o *Worker* devolve ao *Chief* apenas as atualizações dos pesos e o número de exemplos utilizados. Uma vez que os dados brutos permanecem do lado do cliente, obtém-se maior privacidade.

Nesta arquitetura, são considerados quatro cenários de ataque:

- T1 (Ataque Estático).
- T2 (Ataque de Pretensão Estático).
- T3 (Ataque de Pretensão Aleatória).
- T4 (Ataque Progressivo).

Nos cenários T1 e T2, existe um *Worker* malicioso, enquanto nos cenários T3 e T4, existem cinco *Workers* maliciosos.

As configurações dos ataques, bem como o número máximo de atacantes por cenário, são parametrizadas e transmitidas aos *Workers*.

Quando um *Worker* é marcado como atacante, introduz ruído e/ou alterações nas atualizações, com o objetivo de degradar a precisão do modelo global.

A introdução de *Workers* maliciosos é efetuada de forma controlada e reprodutível, permitindo avaliar o desempenho das duas defesas implementadas perante diferentes padrões de ataque.

2) Chief

O *Chief* corresponde ao nó central responsável por coordenar o treino federado e aplicar as duas estratégias de defesa.

Nesta arquitetura, o *Chief* recebe as atualizações dos pesos, calcula métricas de discrepância relativamente ao modelo global (Distância Euclidiana e Divergência *KL*) e determina um *trust score* para cada *Worker*.

Adicionalmente, o *Chief* mantém um histórico por ronda.

Com base nesta informação, o *Chief* decide quais as contribuições elegíveis para a agregação, podendo marcar *Workers* como suspeitos e, quando necessário, isolar os *Workers* que apresentem comportamento malicioso persistente.

Estes mecanismos de defesa asseguram que o *FedAvg* é aplicado de forma criteriosa, evitando a aceitação indiscriminada de contribuições, o que reforça a robustez do modelo global face a atualizações maliciosas.

3) RSU

Nesta dissertação, existem dois níveis de *RSUs,* com funções distintas:

1. *RSUs* na simulação *SUMO/NS-3 (V2I)*:

   Durante a recolha dos dados, existem dois *RSUs* definidos no *SUMO (sumo/rsu.add.xml)*, que representam pontos de infraestrutura *V2I*.

   A configuração destes *RSUs* é transmitida ao *NS-3* no início da simulação, e a latência devolvida pelo *NS-3,* para cada veículo, reflete o comportamento do canal e da rede na presença dessa infraestrutura.

2. *RSU* na camada de autorização (com *BL*):

   Existe também um componente lógico *RSU* (*src/components/rsu.py*), que integra funcionalidades de autorização e validação quando a *BL* está ativa.

   No *(run_simulation.py*), este *RSU* pode ser associado a um cliente *Fabric* e utilizado como interface para verificação de autorização através de *chaincode_query*.

   Nesta dissertação, no *pipeline* federado, os clientes *Flower* comunicam diretamente com o servidor *Flower*, assim, a *BL* atua como mecanismo de suporte (autorização e identidade) e não como substituto do protocolo federado.

   Nesta versão, a *BL* não altera o fluxo base de treino, nem intermedeia o envio efetivo das atualizações no *Flower*.

A Figura 2 representa um cruzamento simulado em *SUMO*, no qual os veículos são representados pelos *Workers*. Existem dois *RSUs* (*RSU1* e *RSU2* (*Chief*)) que comunicam entre si, e as setas representam a comunicação *V2I* e *V2V*.

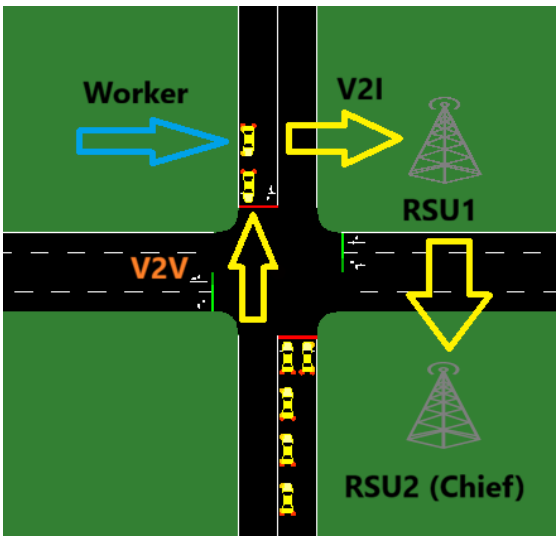

Figura 2 - Cruzamento simulado em SUMO com o RSU1 e RSU2 (Chief) e os Workers (Fonte: Elaboração Própria)

3.1.2 Cenário de simulação e recolha dos dados

A geração de dados é realizada através de uma simulação que utiliza o *SUMO* (mobilidade) e o *NS-3* (rede), sendo coordenada pelo módulo (*sumo_bridge_data_collector.py*).

Este módulo assegura a integração entre as duas ferramentas, através de duas interfaces complementares:

1. *TraCI (SUMO),* para obter os valores de velocidade dos veículos.
2. *Socket TCP* local, para a comunicação com o *NS-3*, com o *bridge* a atuar como servidor em 127.0.0.1:5555.

   Nesta ligação, o *bridge* envia ao *NS-3* as atualizações dos veículos, e o *NS-3* devolve um valor de latência por iteração, que é registado pelo *bridge*.

Durante a simulação, o *bridge* itera por múltiplos passos e mantém *buffers* por entidade, agregando pares [velocidade, latência] com carimbo temporal.

Este procedimento garante que o *dataset* reflete uma evolução temporal coerente e sincronizada entre a mobilidade gerada no *SUMO* e o desempenho de rede simulado no *NS-3*.

3.1.3 Construção de séries temporais

Após a recolha dos valores de velocidade e de latência, os dados são reorganizados em séries temporais e são aplicadas janelas deslizantes, de modo a produzir amostras adequadas para o *LSTM*.

Nesta dissertação, a configuração utilizada define *SEQ_LEN=2*, com duas *features* por *timestep* (velocidade e latência).

Com este procedimento, cada amostra de entrada corresponde a dois instantes consecutivos do par [velocidade, latência], enquanto o alvo corresponde ao par no instante seguinte.

Após a preparação dos dados, o *bridge* exporta os dados de cada *Worker* para *simulation_runs/<cenário>_<timestamp>/worker_data/* e cria ficheiros com o formato *worker_<id>.pkl*, contendo os conjuntos de treino e validação (*hold-out*), guardados como *X, y, X_test, y_test*. São utilizados limites máximos (até *N_TRAIN=256 e até N_VAL=64*), em função do volume de dados disponível.

Quando necessário, é aplicado um *split* 90/10 para obtenção do *hold-out*.

Para efetuar a transição para a fase federada, é criado um ficheiro *DATA_READY.json* na diretoria da execução.

O (*run_simulation.py*) utiliza este ficheiro *JSON* como condição para iniciar os clientes federados, evitando o treino federado com dados incompletos ou ausentes.

3.1.4 Aprendizagem Federada (*Flower*)

Após a deteção do *DATA_READY.json* e dos *ficheiros .pkl*, é iniciada a fase de Aprendizagem Federada com o *Flower,* com o arranque dos clientes.

O servidor é executado no *script* (*flbl_server.py*) e fica a escutar em 0.0.0.0:8080, sendo a disponibilidade verificada pelo *runner,* através de uma ligação *TCP* a 127.0.0.1:8080.

Os *Workers* são lançados como processos independentes através do ficheiro (*flbl_client.py*), e cada *Worker* estabelece ligação ao servidor no endereço local.

Cada *Worker* carrega o *dataset* local, inicializa o modelo *LSTM* e executa o treino federado por rondas.

Em cada ronda, devolve as atualizações dos pesos e o número de exemplos utilizados.

Com base nesta informação, o servidor agrega as contribuições fidedignas, aplicando simultaneamente as estratégias de defesa no *Chief,* de modo a impedir que contribuições maliciosas afetem o modelo global.

3.1.5 Execuções com e sem *BL*

A arquitetura considerada nesta dissertação suporta duas configurações (sem *BL* e com *BL*). Quando o argumento --blockchain está ativo, o (*run_simulation.py*) tenta inicializar um cliente *Fabric* e associá-lo a um componente *RSU*, permitindo realizar verificações de autorização através de chamadas *chaincode_query,* sendo a autorização inferida pelo sucesso da chamada.

Em caso de falha na inicialização do *Fabric*, a execução prossegue sem este componente.

Adicionalmente, no código, os *Workers* integram uma base de identidade criptográfica suportada por *Indy* (*wallet* e *DID*), que é inicializada no lado do cliente, independentemente da configuração com ou sem *BL*.

Importa salientar que a arquitetura não substitui o protocolo federado do *Flower* por uma rede *BL*, uma vez que a *BL* atua como mecanismo de suporte (identidade, autorização e possibilidade de auditoria). O treino federado e a comunicação cliente-servidor no *Flower* mantêm-se inalterados.

3.1.6 Fluxo completo do sistema

A execução completa do sistema pode ser resumida nas seguintes etapas:

- O (*run_all_scenarios.py*) invoca o (*run_simulation.py*) para os quatro cenários, em modo sem *BL* e em modo com *BL* (--blockchain).
- O (*run_simulation.py*) inicia o servidor *Flower* e executa a recolha dos dados *SUMO/NS-3,* através do *bridge*.

  O *bridge* recolhe a velocidade no *SUMO* via *TraCI* e a *latência* do *NS-3* através de *TCP* em 127.0.0.1:5555.

- O *bridge* constrói séries temporais, cria janelas com *SEQ_LEN = 2*, exporta os ficheiros *worker_<id>.pkl para simulation_runs/<cenário>_<timestamp>/worker_data/* e cria o ficheiro *DATA_READY.json*.
- Após a deteção do *DATA_READY.json*, o *runner* inicia os clientes *Flower* (*flbl_client.py*), que treinam localmente e enviam as atualizações dos pesos em cada ronda.
- O *Chief* aplica as defesas (Distância Euclidiana e Divergência *KL*), rejeita contribuições maliciosas e pode isolar *Workers* antes da agregação final do modelo global.

  Por fim, o sistema armazena *logs* e os artefactos de execução, permitindo análise posterior.

3.1.7 Arquitetura geral do sistema

A Figura 3 – Arquitetura geral do sistema sintetiza o fluxo operacional implementado nesta dissertação, desde a execução automatizada dos cenários até à geração de resultados.

A Figura organiza o processo em etapas sequenciais: *Runner*, *Simulação (SUMO+NS-3)*, *Dados*, *Aprendizagem Federada* (*Chief* ↔ *Workers*), Defesas e Relatórios/Gráficos.

Adicionalmente, indica o componente *RSU*/Blockchain (opcional), que apenas intervém quando a configuração com *BL* está ativa.

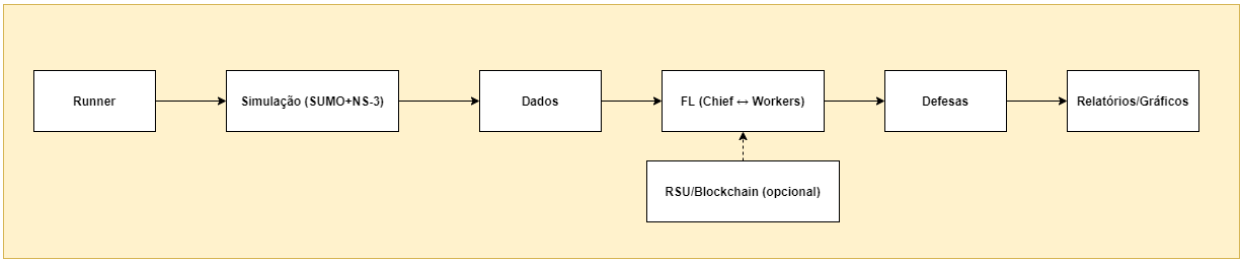

Figura 3 - Arquitetura geral do sistema

- Na etapa *Runner*, a execução é orquestrada de forma automatizada, permitindo executar os quatro cenários de ataque (T1 a T4) em duas configurações (com e sem *BL*).

  Esta orquestração invoca o *pipeline* de cada cenário, assegurando a passagem dos parâmetros necessários à execução (cenário, número de *Workers* e número de rondas).
- Na etapa Simulação *(SUMO+NS-3)*, ocorre a recolha e preparação dos dados, a partir da integração entre o *SUMO* (mobilidade) e o *NS-3* (comunicações/latência).

Como resultado, ficam disponíveis os dados necessários para o treino local dos participantes, sendo a passagem à etapa seguinte condicionada pela conclusão da recolha.

- Na etapa *FL (Chief ↔ Workers)*, o servidor federado instância o *Chief*, que coordena as rondas, agrega o modelo global e interage com os *Workers*, os quais executam o treino local e enviam as atualizações ao servidor.

  Quando a configuração com *BL* está ativa, o bloco *RSU/*Blockchain (opcional) é introduzido como suporte a mecanismos de autorização e auditoria, complementando o processo federado, sem alterar o fluxo principal apresentado na Figura.
- Na etapa Defesas, o *Chief* aplica os mecanismos implementados de Distância Euclidiana, de Divergência *KL* e de confiança, com o objetivo de detetar e isolar contribuições maliciosas, impedindo a sua influência em agregações futuras.
- Por fim, na etapa Relatórios/Gráficos, são produzidos os resultados (registos, métricas, relatórios e gráficos), organizados por execução, permitindo a análise comparativa dos cenários e do desempenho do sistema.

3.2 Mecanismos de Privacidade e Segurança com *BL*

Nesta dissertação, a Aprendizagem Federada reforça a privacidade, ao permitir que os *Workers* mantenham os dados de treino localmente e enviem apenas as atualizações dos pesos para o servidor.

Apesar deste benefício, subsistem riscos de segurança que necessitam de ser mitigados, nomeadamente a participação não autorizada de *Workers*, o envio de atualizações maliciosas e a necessidade de auditoria das interações e contribuições.

Para responder a estes desafios, são utilizados dois mecanismos baseados em *BL* (*Hyperledger Indy* e *Hyperledger Fabric*), suportados por um componente *RSU,* que atua como ponto de validação e controlo.

3.2.1 *Hyperledger Indy*

Neste sistema, cada *Worker* possui uma *wallet Indy* e um *DID*.

A ligação ao *pool Indy* é estabelecida através de um ficheiro genesis, descarregado via *URL*, e cada *Worker* cria e armazena localmente o respetivo *DID* na *wallet*.

Pool a ser iniciado e criação da *wallet Indy*:

utils/indy_setup.py

```
POOL_NAME = os.environ.get("POOL_NAME", "pool_name")
GENESIS_URL = os.environ.get("GENESIS_URL", "http://localhost:9000/genesis")
```

Figura 4 - Constantes de configuração Indy

```
async def _ensure_pool_open():
    """Ensure pool is opened (singleton pattern)"""
    global _pool_handle
    async with _pool_lock:
        if _pool_handle is not None:
            return _pool_handle

        # Set protocol version
        await pool.set_protocol_version(2)

        # Download genesis file
        genesis_path = _download_genesis(GENESIS_URL)
        pool_config = json.dumps({"genesis_txn": genesis_path})

        try:
            # Create pool config
            await pool.create_pool_ledger_config(POOL_NAME, pool_config)
        except PoolLedgerConfigAlreadyExistsError:
            log("Pool config already exists")

        # Open pool
        _pool_handle = await asyncio.wait_for(
            pool.open_pool_ledger(POOL_NAME, None),
            timeout=OPEN_TIMEOUT
        )
        log(f"Pool opened. handle={_pool_handle}")
        return _pool_handle
```

Figura 5 - Inicialização da pool Indy

```python
async def setup_indy(wallet_id: str, wallet_key: str = "wallet_key") -> int:
    """
    Ensures pool is open once. Returns wallet_handle (int) for the given wallet_id.

    Args:
        wallet_id: Unique wallet identifier
        wallet_key: Wallet encryption key

    Returns:
        int: Wallet handle
    """
    # Ensure pool is opened
    await _ensure_pool_open()

    wc = json.dumps({"id": wallet_id})
    wk = json.dumps({"key": wallet_key})

    try:
        # Try to create wallet
        await wallet.create_wallet(wc, wk)
    except WalletAlreadyExistsError:
        # Wallet already exists, this is fine
        pass

    # Open wallet
    wh = await wallet.open_wallet(wc, wk)
    return wh
```

Figura 6 - Configuração do Indy

Criação dos *Workers* com a *wallet* e o *DID*:

components/worker.py

```python
async def create_workers(num_workers: int, local_model_size: int) -> List[Worker]:
    workers: List[Worker] = []
    for i in range(num_workers):
        wh = await setup_indy(wallet_id=f"wallet_worker_{i}", wallet_key="wallet_key")
        did_str, _ = await indy_did.create_and_store_my_did(wh, "{}")
        workers.append(Worker(i, did_str, wh, local_model_size))
        await asyncio.sleep(0)
    return workers
```

Figura 7 - Criação dos Workers com a wallet e o DID

Para um *Worker* demonstrar a posse do *DID*, são assinados desafios (*nonces*), utilizando as chaves armazenadas na *wallet*.

Este mecanismo permite realizar uma autenticação criptográfica, sem divulgar chaves privadas ou dados sensíveis.

Assinatura de *nonce* no Worker:

components/worker.py

```python
async def sign_nonce(self, nonce: str) -> str:
    try:
        verkey = await self._local_verkey()
        sig_bytes = await indy_crypto.crypto_sign(self.wallet_handle, verkey, nonce.encode("utf-8"))
        return base64.b64encode(sig_bytes).decode("ascii")
    except Exception as e:
        log_worker(f"ERROR signing nonce: {e}")
        raise e
```

Figura 8 - Assinatura de nonce no Worker

3.2.2 *Hyperledger Fabric*

A autorização é verificada através de um cliente *Fabric*, que carrega o perfil da rede e, de seguida, associa o *channel*, o *peer* e o *orderer* necessários para invocar o *chaincode*.

Cliente Fabric a ser inicializado e importação do *Admin* via *PEMs*:

utils/fabric_setup.py

```python
def init_fabric_client(network_json: str) -> Client:
    c = Client(net_profile=network_json)
    # Load/create channel
    ch_name = "mychannel"
    # If your profile defines the channel, load it; else create it
    try:
        c.new_channel(ch_name)  # idempotent in practice
    except Exception:
        pass

    # Attach peer and orderer objects from profile so queries work
    peer_name = "peer0.org1.example.com"
    orderer_name = "orderer.example.com"  # adjust if different in your profile

    peer = c.get_peer(peer_name)
    if peer is None:
        raise RuntimeError(f"Peer not found in profile: {peer_name}")
    orderer = c.get_orderer(orderer_name)
    if orderer is None:
        raise RuntimeError(f"Orderer not found in profile: {orderer_name}")

    ch = c.get_channel(ch_name) or c.new_channel(ch_name)
    # Safe to re-add; SDK guards duplicates
    ch.add_peer(peer)
    ch.add_orderer(orderer)

    return c
```

Figura 9 - Inicialização do cliente Fabric

```python
def import_admin_from_pems(client: Client, org: str, msp_id: str, cert_path: str, key_path: str):
    from hfc.fabric.user import create_user
    from hfc.util.keyvaluestore import FileKeyValueStore
    if not (os.path.exists(cert_path) and os.path.exists(key_path)):
        raise FileNotFoundError("Admin PEMs not found")
    with open(cert_path, "rb") as f: cert = f.read()
    with open(key_path, "rb") as f: key = f.read()
    state_store = FileKeyValueStore("./hfc-key-store")
    admin = create_user("Admin", org, state_store, msp_id, key, cert)
    client._users.setdefault(org, {})["Admin"] = admin
    return admin
```

Figura 10 - importação do Admin via PEMs

Quando a simulação é executada com *BL*, o *runner* ativa essa configuração e tenta associar o cliente *Fabric* ao *RSU.* Em caso de falha, a execução prossegue sem o componente *Fabric*.

Ativação do Fabric no *runner* quando a *BL* está ativa

run_simulation.py

```python
async def prepare_workers(self):
    """Create workers with RSU - needed for simulation"""
    local_model_size = 10
    workers = await create_workers(self.num_workers, local_model_size)

    rsu = RSU(
        id=1,
        fabric_client=None,
        channel="mychannel",
        cc_name="mycc",
        org="org1.example.com",
        peer="peer0.org1.example.com",
        strict=False,
        min_score=0,
    )

    if self.use_blockchain:
        try:
            fabric = init_fabric_client("./config/network.json")
            import_admin_from_pems(
                fabric,
                org="org1.example.com",
                msp_id="Org1MSP",
                cert_path="/home/goncalolemos/fabric-test/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem",
                key_path="/home/goncalolemos/fabric-test/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/f3571e2d86c3ed1182fd372b230c71a9b23d0cd5338cb9c64b03ca6536fe17c0_sk",
            )
            rsu.set_fabric(fabric)
        except Exception as e:
            log(f"[RUNNER] Blockchain setup failed: {e}")

    for w in workers:
        await rsu.add_worker(w)
        w.rsu = rsu

    attack_type = ATTACK_CONFIG.get(self.scenario, {}).get("type")
    max_attackers = ATTACK_CONFIG.get(self.scenario, {}).get("max_attackers", 0)
    for w in workers:
        w.attacker = is_attacker(w.id, max_attackers)
        w.attack_type = attack_type
        w.max_attackers = max_attackers

    return workers, rsu
```

Figura 11 - Ativação do Fabric no runner quando a BL está ativa

3.2.3 *RSU*

Quando o *Fabric* não está ativo, ou ocorre uma falha na inicialização, e uma vez que o *RSU* está configurado com *strict=False*, a autorização é aceite por defeito.

Quando o *Fabric* está ativo, o *RSU* tenta comprovar a posse do *DID* através de um *nonce* assinado. Contudo, na configuração atual (*indy_pool_handle=None*), as rotinas de verificação de assinatura devolvem verdadeiro por defeito, pelo que a prova criptográfica não é efetivamente validada nesta execução.

O *RSU* suporta validação criptográfica baseada em *Indy*.

No entanto, nesta simulação, o *RSU* é inicializado sem *indy_pool_handle (indy_pool_handle=None)*, o que faz com que as rotinas *verify_signature()* e *_verify_payload_signature()* devolvam verdadeiro, desativando, na prática, a validação criptográfica via *Indy* nesta execução.

Autorização de um *Worker* no *RSU*

components/rsu.py

```python
async def is_worker_authorized(
    self,
    did_str: str,
    verify_sig: Callable[[str], Awaitable[str]],
) -> bool:
    """
    FIXED: Use mycc's 'query' function
    """
    if self.fabric is None:
        return not self.strict

    admin = self._get_admin()
    if admin is None:
        return not self.strict

    try:
        ch = self.fabric.get_channel(self.channel)
        if ch is None:
            return not self.strict

        peer_obj = self.fabric.get_peer(self.peer)
        if peer_obj is None:
            return not self.strict

        try:
            ch.add_peer(peer_obj)
        except Exception:
            pass

        # Prove DID ownership
        nonce = os.urandom(16).hex()
        sig = await verify_sig(nonce)
        if not sig:
            return False

        if not await self.verify_signature(did_str, nonce, sig):
            return False

        # Call mycc's query function
        resp = await self.fabric.chaincode_query(
            requestor=admin,
            channel_name=self.channel,
            peers=[peer_obj],
            cc_name=self.cc_name,
            fcn="query",
            args=["a"]
        )

        # If query succeeds, authorize
        return True

    except Exception as e:
        log_rsu(f"[RSU] authorization check failed: {e}")
        return not self.strict
```

Figura 12 - Autorização de um Worker no RSU

Para integrar o *Worker* no *RSU,* é utilizado o método *add_worker*, que apenas regista o *Worker* caso este cumpra a validação anterior:

components/rsu.py

```
async def add_worker(self, worker: Any) -> None:
    did_str = getattr(worker, "did", None)
    if not did_str:
        raise ValueError(f"Worker {getattr(worker, 'id', '?')} missing DID")
    signer = getattr(worker, "sign_nonce", None)
    if signer is None or not callable(signer):
        raise ValueError(f"Worker {getattr(worker, 'id', '?')} missing sign_nonce()")
    if await self.is_worker_authorized(did_str, signer):
        self.workers.append(worker)
    else:
        raise ValueError(f"Worker {getattr(worker, 'id', '?')} not authorized.")
```

Figura 13 - Integração do Worker no RSU

O *RSU* suporta uma verificação de integridade das atualizações, antes de as reencaminhar para o *Chief*.

O *RSU* descodifica o *payload* e tenta validar a assinatura através do *Indy*. Contudo, na configuração atual (indy_pool_handle=None), esta validação criptográfica encontra-se, na prática, desativada e é aceite por defeito.

De seguida, o *RSU* valida a autorização via *Fabric,* quando este está ativo, e reencaminha a atualização para o *Chief*.

components/rsu.py

```python
async def submit_and_authorize(self, worker_id: int, payload_b64: str,
                               sig_b64: str, did: str, num_examples: int = 0) -> bool:
    """Check blockchain authorization BEFORE forwarding to Chief"""

    # Decode payload
    try:
        raw = base64.b64decode(payload_b64.encode("ascii"))
        flat = np.frombuffer(raw, dtype=np.float32)
    except Exception as e:
        log_rsu(f"[BLOCKCHAIN] Worker {worker_id}: Failed to decode payload: {e}")
        return False

    # 1. Verify signature
    try:
        msg = flat.tobytes()
        sig_valid = await self._verify_payload_signature(did, msg, sig_b64)
        if not sig_valid:
            log_rsu(f"[BLOCKCHAIN] Worker {worker_id}: Invalid signature")
            return False
    except Exception as e:
        log_rsu(f"[BLOCKCHAIN] Worker {worker_id}: Signature verification error: {e}")
        return False

    # 2. Check blockchain authorization
    if self.fabric:
        authorized = await self._check_blockchain_authorization_safe(worker_id, did)
        if not authorized:
            return False
    else:
        if self.strict:
            return False

    # 3. Forward to Chief
    if self._submit_upstream:
        self._submit_upstream(worker_id, flat, num_examples)
        return True
    else:
        log_rsu(f"[RSU-ERROR] Worker {worker_id}: No upstream configured!")
        return False
```

Figura 14 - verificação de integridade das atualizações antes de enviar para o Chief

Contudo, apesar de estar implementado, este método não é invocado no fluxo principal do *Flower* nesta versão, uma vez que os clientes *Flower* comunicam diretamente com o servidor (*start_numpy_client*("127.0.0.1:8080")).

3.2.4 Registo e validação dos modelos no *Hyperledger Fabric*

Para permitir auditoria e imutabilidade das versões do modelo, existe uma camada utilitária que armazena e valida modelos num *chaincode* dedicado.

Esta camada inclui funções para armazenar um modelo (*storeModel*), obter o modelo (*getModel*) e validar o modelo e a respetiva assinatura (*validateModel*).

Contudo, esta camada não se encontra integrada no fluxo principal da simulação e apresenta parâmetros que podem divergir da configuração utilizada no *RSU*.

utils/blockchain_utils.py

```python
import os
import json
import numpy as np

# Permite escolher o chaincode por env; default mantém o atual
CC_NAME = os.getenv("CC_NAME", "modelcontract")

def _as_list(a):
    # garante lista JSON mesmo que venha np.ndarray
    return a.tolist() if isinstance(a, np.ndarray) else a

def _admin(worker):
    # Usa "Org1", não "org1.example.com"
    return worker.fabric_client.get_user("Org1", "Admin")

def store_model_on_blockchain(worker, model, signature):
    model_json = json.dumps(_as_list(model))
    args = ["storeModel", str(worker.id), model_json, signature]
    try:
        return worker.fabric_client.chaincode_invoke(
            requestor=_admin(worker),
            channel_name="mychannel",
            peers=["peer0.org1.example.com"],
            fcn="storeModel",
            args=args,
            cc_name=CC_NAME,
            transient_map=None,
            wait_for_event=True,
        )
    except Exception as e:
        print(f"Error storing model on blockchain: {e}")
        return None

def get_model_from_blockchain(worker, model_key):
    args = ["getModel", str(model_key)]
    try:
        resp = worker.fabric_client.chaincode_query(
            requestor=_admin(worker),
            channel_name="mychannel",
            peers=["peer0.org1.example.com"],
            fcn="getModel",
            args=args,
            cc_name=CC_NAME,
        )
        return json.loads(resp)
    except Exception as e:
        print(f"Error getting model from blockchain: {e}")
        return None

def validate_model_on_blockchain(worker, model, signature):
    model_json = json.dumps(_as_list(model))
    args = ["validateModel", model_json, signature, str(worker.id)]
    try:
        return worker.fabric_client.chaincode_query(
            requestor=_admin(worker),
            channel_name="mychannel",
            peers=["peer0.org1.example.com"],
            fcn="validateModel",
            args=args,
            cc_name=CC_NAME,
        )
    except Exception as e:
        print(f"Error validating model on blockchain: {e}")
        return None
```

Figura 15 - Registo e validação dos modelos no Hyperledger Fabric

3.2.5 Ativação da *BL* na simulação

Para executar a *BL* na simulação, é utilizado o argumento --blockchain, que ativa o caminho de inicialização correspondente no *runner*.

run_simulation.py

```
def main():
    parser = argparse.ArgumentParser(description="Run FLBL simulation")
    parser.add_argument("--scenario", type=str, default="T3",
                        choices=["T1", "T2", "T3", "T4"])
    parser.add_argument("--workers", type=int, default=20)
    parser.add_argument("--rounds", type=int, default=30)
    parser.add_argument("--blockchain", action="store_true")

    args = parser.parse_args()

    runner = SimulationRunner(
        scenario=args.scenario,
        num_workers=args.workers,
        num_rounds=args.rounds,
        use_blockchain=args.blockchain
    )

    runner.run()
```

Figura 16 - Ativação da BL na simulação

3.3 Estratégia de Deteção de Ataques baseada em comportamento

Neste trabalho, a *Behavior Attestation* é implementada no *Chief*, que avalia o comportamento dos *Workers* através das atualizações dos pesos enviadas ao longo das rondas.

Um *Worker* fidedigno tende a produzir atualizações compatíveis com o modelo global atual, enquanto um *Worker* malicioso tende a produzir atualizações divergentes relativamente ao modelo global.

Nesta dissertação, são implementados dois mecanismos de defesa (Distância Euclidiana e Divergência *KL*).

Estas duas defesas são aplicadas nos quatro cenários da simulação.

Para além destas duas métricas, o *Chief* calcula também um *trust score*, utilizado para marcar suspeitas, acumular evidência e isolar *Workers,* quando necessário.

3.3.1 Definição básica das duas defesas

Distância Euclidiana

Nesta defesa, o *Chief* calcula a norma (L2) entre o vetor global e o vetor recebido do *Worker*:

src/components/chief.py

```
def _compute_raw_distance(self, flat_vec: np.ndarray) -> float:
    if self.global_model is None:
        return 0.0
    return float(np.linalg.norm(self.global_model - flat_vec))
```

Figura 17 - Distância Euclidiana

Divergência *KL* (*Kullback-Leibler*)

Nesta defesa, o *Chief* calcula a Divergência *KL* entre o vetor global e o vetor recebido do *Worker*.

Para esse efeito, é utilizada a função *kl_divergence*:

src/components/chief.py

```
def _compute_raw_kl(self, flat_vec: np.ndarray) -> float:
    if self.global_model is None:
        return 0.0
    return kl_divergence(self.global_model, flat_vec)
```

Figura 18 - Divergência KL (Kullback-Leibler)

3.3.2 Implementação no código das duas defesas

Distância Euclidiana

A Distância Euclidiana utiliza o histórico para detetar desvios persistentes ou desvios instantâneos.

No código, existe um contador de violações (*record.euclidean_violations*), que permite identificar qual a defesa que detetou o comportamento malicioso.

src/components/chief.py

```
def _evaluate_per_defense(self, worker_id: int, record: WorkerRecord,
                          raw_distance: float, raw_kl: float, flat_vec: np.ndarray):
    """Evaluate each defense mechanism separately"""
    euclidean_params = self.config.get("euclidean", {})
    kl_params = self.config.get("kl_divergence", {})

    # EUCLIDEAN DEFENSE
    max_mean_dist = euclidean_params.get("max_mean_distance", 2.5)
    max_last_dist = euclidean_params.get("max_last_distance", 3.0)

    euclidean_violation = False
    if len(record.distance_history) >= 3:
        mean_dist = np.mean(record.distance_history[-5:])
        if mean_dist > max_mean_dist or raw_distance > max_last_dist:
            record.euclidean_violations += 1
            record.detected_by.add("euclidean")
            euclidean_violation = True
```

Figura 19 - Implementação no código da Distância Euclidiana

Neste código, se *mean_dist* for superior a *max_mean_dist,* o desvio é considerado persistente.

Se *raw_distance* for superior a *max_last_dist,* ocorre um *spike* instantâneo.

Divergência *KL*

A Divergência *KL* mede a incompatibilidade estatística entre vetores.

Quando ocorre uma deteção, o contador *record.kl_divergence_violations* é incrementado e o campo *detected_by* é marcado como *kl_divergence*.

src/components/chief.py

```
# KL DIVERGENCE DEFENSE - Use config thresholds
max_mean_kl = kl_params.get("max_mean_kl", 0.25)
max_last_kl = kl_params.get("max_last_kl", 0.35)

kl_violation = False
if len(record.kl_history) >= 3:
    mean_kl = np.mean(record.kl_history[-5:])
    if mean_kl > max_mean_kl or raw_kl > max_last_kl:
        record.kl_divergence_violations += 1
        record.detected_by.add("kl_divergence")
        kl_violation = True
```

Figura 20 - Implementação no código da Divergência KL

No código implementado, existe uma verificação adicional: caso a Distância Euclidiana sinalize uma violação e a Divergência *KL* não sinalize, o *Chief* pode, ainda assim, incrementar o *record.kl_divergence_violations,* se o *raw_kl* estiver acima do valor esperado pelo histórico.

Esta verificação permite que, perante um desvio forte já detetado pela Distância Euclidiana, a Divergência *KL* também sinalize quando houver evidência estatística suficiente.

3.3.3 *Trust Score*

Para além destas duas defesas, o *Chief* calcula também um *trust score,* de modo a controlar o comportamento dos *Workers* ao longo das rondas.

Nesta dissertação, o *trust score* depende das duas defesas (Distância Euclidiana e Divergência *KL*) e considera o pior caso (*min*).

src/components/chief.py

```
def _compute_trust_score(self, raw_distance: float, raw_kl: float) -> float:
    dist_score = max(0.0, 1.0 - (raw_distance / self.distance_threshold))
    kl_score = max(0.0, 1.0 - (raw_kl / self.kl_threshold))
    return min(dist_score, kl_score)
```

Figura 21 - Cálculo do Trust Score

Se uma das métricas indicar comportamento anómalo, o *trust score* diminui de imediato.

Deste modo, um *Worker* apenas é considerado altamente fiável quando ambas as métricas se mantêm elevadas.

3.3.4 Critérios e limiares das defesas por cenário (T1-T4)

Para as defesas baseadas em Distância Euclidiana e Divergência *KL*, apresentam-se o tamanho da janela de observação (número de rondas considerado na avaliação) e os limiares de decisão aplicados ao valor médio e ao último valor calculado nessa janela.

Estes parâmetros suportam a identificação de comportamentos anómalos nas atualizações dos *Workers*.

Adicionalmente, inclui-se o mecanismo de *trust score*, responsável pela decisão de isolamento do *Worker*, indicando-se o limiar de confiança e o número de rondas necessárias para o isolamento e para a eliminação definitiva do nó malicioso.

Em conjunto, estes valores refletem o ajuste específico por cenário, assegurando consistência entre os processos de deteção e mitigação ao longo das rondas da Aprendizagem Federada.

A Tabela 2 apresenta os parâmetros de configuração adotados nos mecanismos de defesa, para cada cenário de ataque (T1–T4).

Tabela 2 - Critérios e limiares das defesas por cenário (T1–T4)

Cenário	Mecanismo	Janela	Limiar médio	Limiar último	Trust (isolamento)	Rondas p/ isolar	Rondas p/ eliminar
T1	Euclidiana	5	3.5	4.0	—	—	—
T1	KL	5	0.25	0.35	—	—	—
T1	Trust	—	—	—	0.45	4	5
T2	Euclidiana	3	1.5	2.2	—	—	—
T2	KL	3	0.10	0.15	—	—	—
T2	Trust	—	—	—	0.53	2	3
T3	Euclidiana	6	2.7	3.2	—	—	—
T3	KL	6	0.25	0.35	—	—	—
T3	Trust	—	—	—	0.40	3	5
T4	Euclidiana	4	1.05	1.60	—	—	—
T4	KL	4	0.08	0.12	—	—	—
T4	Trust	—	—	—	0.48	5	6

3.3.5 *Baseline* comportamental

A arquitetura desenvolvida estabelece uma *baseline* inicial durante as primeiras rondas, com o objetivo de servir de referência para a comparação de desvios futuros.

Esta *baseline* inicial é particularmente relevante na deteção de ataques de pretensão estático (T2), uma vez que, nesse cenário, o *Worker* inicia o processo com atualizações fidedignas e, posteriormente, altera o comportamento para malicioso.

src/components/chief.py (WorkerRecord)

```python
def add_metrics(self, trust: float, distance: float, kl: float, round_num: int):
    self.trust_history.append(trust)
    self.distance_history.append(distance)
    self.kl_history.append(kl)
    self.rounds_participated += 1

    if round_num <= 3 and not self.baseline_established:
        if len(self.trust_history) >= 3:
            self.baseline_trust = np.mean(self.trust_history[:3])
            self.baseline_distance = np.mean(self.distance_history[:3])
            self.baseline_kl = np.mean(self.kl_history[:3])
            self.baseline_established = True

    if trust < 0.50:
        self.consecutive_low_trust += 1
    else:
        self.consecutive_low_trust = 0

    max_history = 20
    if len(self.trust_history) > max_history:
        self.trust_history = self.trust_history[-max_history:]
        self.distance_history = self.distance_history[-max_history:]
        self.kl_history = self.kl_history[-max_history:]
```

Figura 22 - Métricas da Baseline comportamental

3.3.6 Isolamento/Eliminação dos *Workers*

Para decidir quais os *Workers* que devem ser isolados ou eliminados, o *Chief* percorre as atualizações recebidas, calcula as métricas, atualiza os históricos e aplica as duas defesas.

src/components/chief.py

```python
def aggregate_pending(self) -> Optional[np.ndarray]:
    if not self._pending_updates:
        return self.global_model

    self.round_num += 1

    updates = [
        (wid, flat_vec)
        for wid, (flat_vec, _) in self._pending_updates.items()
        if wid not in self.eliminated_workers
    ]

    if not updates:
        return self.global_model

    for worker_id, flat_vec in updates:
        if worker_id in self.isolated_workers or worker_id in self.eliminated_workers:
            continue

        raw_distance = self._compute_raw_distance(flat_vec)
        raw_kl = self._compute_raw_kl(flat_vec)
        trust_score = self._compute_trust_score(raw_distance, raw_kl)

        record = self.worker_registry[worker_id]
        record.add_metrics(trust_score, raw_distance, raw_kl, self.round_num)

        # PER-DEFENSE EVALUATION
        self._evaluate_per_defense(worker_id, record, raw_distance, raw_kl, flat_vec)

        # T2 DETECTION (specialized)
        if self.scenario == "T2":
            should_isolate, reason = self._evaluate_t2_worker(worker_id, record, raw_distance, raw_kl)
            if should_isolate:
                self._isolate_worker(worker_id, reason)
                continue

        # T4 DETECTION (specialized)
        elif self.scenario == "T4":
            should_isolate, reason = self._evaluate_t4_worker(worker_id, record, raw_distance, raw_kl)
            if should_isolate:
                self._isolate_worker(worker_id, reason)
                continue

        # PER-DEFENSE ISOLATION (only for T1/T3 without specialized detection)
        elif len(record.detected_by) > 0:
            total_violations = (record.euclidean_violations +
                                record.kl_divergence_violations)

            # Isolate if sufficient violations from any defense
            if total_violations >= self.isolation_threshold:
                defenses_str = ", ".join(sorted(record.detected_by))
                self._isolate_worker(worker_id, f"defenses: {defenses_str}")
                continue
```

Figura 23 - Isolamento Workers parte 1

```
    # STANDARD DETECTION
    if trust_score < self.trust_threshold:
        record.suspicious_count += 1
        self._mark_suspicious(worker_id)
        if record.suspicious_count >= self.isolation_threshold:
            self._isolate_worker(worker_id, "threshold_violations")
    else:
        if worker_id in self.suspicious_workers and self.scenario not in ["T2", "T4"]:
            self._restore_worker(worker_id)

valid_updates = [
    (flat_vec, num_examples)
    for wid, (flat_vec, num_examples) in self._pending_updates.items()
    if wid not in self.isolated_workers and wid not in self.eliminated_workers
]

if not valid_updates:
    return self.global_model

total_examples = sum(num_ex for _, num_ex in valid_updates)
if total_examples == 0:
    return self.global_model

weighted_sum = sum(flat_vec * num_ex for flat_vec, num_ex in valid_updates)
self.global_model = weighted_sum / total_examples
self._pending_updates.clear()

return self.global_model
```

Figura 24 - Isolamento Workers parte 2

Isolamento por "*violations*" das defesas nos cenários T1 e T3

Nos cenários em que não existe deteção especializada, sempre que uma das defesas sinaliza deteção e o número de violações acumuladas atinge o limiar definido, o *Worker* é isolado:

src/components/chief.py (excerto)

```
# PER-DEFENSE ISOLATION (only for T1/T3 without specialized detection)
elif len(record.detected_by) > 0:
    total_violations = (record.euclidean_violations +
                        record.kl_divergence_violations)

    # Isolate if sufficient violations from any defense
    if total_violations >= self.isolation_threshold:
        defenses_str = ", ".join(sorted(record.detected_by))
        self._isolate_worker(worker_id, f"defenses: {defenses_str}")
        continue
```

Figura 25 - Isolamento por "violations" das defesas nos cenários T1 e T3

Deteção "*standard*" por *trust score*

Para além do registo das violações por defesa, existe também uma métrica adicional de segurança, baseada no *trust score*.

Se o *trust_score* for inferior ao *trust_threshold,* é incrementado no *suspicious_count* e, quando esse contador atinge o *isolation_threshold*, o *Worker* é isolado.

src/components/chief.py (excerto)

```
# STANDARD DETECTION
if trust_score < self.trust_threshold:
    record.suspicious_count += 1
    self._mark_suspicious(worker_id)
    if record.suspicious_count >= self.isolation_threshold:
        self._isolate_worker(worker_id, "threshold_violations")
else:
    if worker_id in self.suspicious_workers and self.scenario not in ["T2", "T4"]:
        self._restore_worker(worker_id)
```

Figura 26 - Deteção "standard" por trust score

Número máximo de atacantes por cenário

Neste sistema, o isolamento dos *Workers* respeita o número máximo de atacantes por cenário (T1 – 1, T2 – 1, T3 – 5, T4 – 5), através do parâmetro *max_attackers*:

src/components/chief.py

```python
def _isolate_worker(self, worker_id: int, reason: str = "unknown"):
    """
    Isola o worker, respeitando o número máximo de atacantes definido
    para o cenário (T1=1, T2=1, T3=5, T4=5).
    ALWAYS credits both defense mechanisms - they work together.
    """
    if self.max_attackers is not None:
        current_flagged = len(self.malicious_workers | self.isolated_workers)
        if current_flagged >= self.max_attackers:
            return

    self.reliable_workers.discard(worker_id)
    self.suspicious_workers.discard(worker_id)
    self.malicious_workers.add(worker_id)
    self.isolated_workers.add(worker_id)
    record = self.worker_registry[worker_id]
    record.status = WorkerStatus.ISOLATED
    record.isolation_timestamp = datetime.now()

    # ALWAYS credit BOTH defense mechanisms
    # Both Euclidean and KL contribute to trust score calculation
    # so both deserve credit for any detection
    record.detected_by.add("euclidean")
    record.detected_by.add("kl_divergence")
```

Figura 27 - Cap por cenário

Importa salientar que, no momento do isolamento do *Worker,* o código adiciona sempre "*euclidean*" e "*kl_divergence*" no *record.detected_by*.

Deste modo, o campo *detected_by* pode refletir tanto a evidência acumulada ao longo da avaliação como a marcação final aplicada no isolamento.

3.3.7 Implementação das defesas nos respetivos cenários

Cenário T1 – Ataque Estático

Neste cenário, o *Worker* introduz continuamente ruído nos pesos:

src/components/worker.py

```
def _static_attack(self, weights):
    try:
        return [w + np.random.normal(0.0, 1.0, w.shape).astype(w.dtype) for w in weights]
    except Exception:
        return weights
```

Figura 28 - Ataque Estático

Como consequência, a Distância Euclidiana e a Divergência *KL* tendem a aumentar rapidamente, enquanto o *trust score* diminui de forma contínua.

O isolamento ocorre devido à acumulação de violações e/ou por o *trust score* se manter abaixo do limiar definido.

Cenário T2 – Ataque de Pretensão Estático

Neste cenário, o *Worker* comporta-se inicialmente de forma benigna e, posteriormente, de forma maliciosa:

src/components/worker.py

```
def _static_pretension_attack(self, weights, round_num):
    if round_num < 3:
        return weights
    if not self.attack_started:
        self.attack_started = True
    try:
        return [w + np.random.normal(0.0, 2.0, w.shape).astype(w.dtype) for w in weights]
    except Exception:
        return weights
```

Figura 29 - Ataque de Pretensão Estático

O *Chief* aplica deteção especializada através do *_evaluate_t2_worker()* e procura *spikes* face às rondas recentes, desvios relativamente à *baseline* (rondas 1 a 3), valores absolutos elevados, comportamento anómalo persistente e acumulação no *t2_suspicion_score*.

src/components/chief.py

```python
def _evaluate_t2_worker(self, worker_id: int, record: WorkerRecord,
                        raw_distance: float, raw_kl: float) -> Tuple[bool, str]:
    """
    T2 pretension detection:
    - Worker começa honesto (primeiras rondas constroem baseline);
    - Depois muda de comportamento (distâncias / KL sobem de forma súbita e mantida).
    Detetor mais sensível para garantir que apanha o único atacante.
    """
    if self.round_num <= 3:
        return False, ""

    violation_score = 0
    shift_signals = 0
    reasons = []
    euclidean_triggered = False
    kl_triggered = False

    # 1) Picos súbitos na distância (últimas 3 rondas)
    if len(record.distance_history) >= 4:
        recent_mean = np.mean(record.distance_history[-4:-1])
        spike = raw_distance - recent_mean
        if spike > 0.70:
            violation_score += 3
            shift_signals += 1
            euclidean_triggered = True
            reasons.append(f"dist_spike={spike:.4f}")
        elif spike > 0.50:
            violation_score += 1
            euclidean_triggered = True

    # 2) Picos súbitos em KL
    if len(record.kl_history) >= 4:
        recent_mean = np.mean(record.kl_history[-4:-1])
        spike = raw_kl - recent_mean
        if spike > 0.05:
            violation_score += 3
            shift_signals += 1
            kl_triggered = True
            reasons.append(f"kl_spike={spike:.6f}")
        elif spike > 0.03:
            violation_score += 1
            kl_triggered = True
```

Figura 30 - Evaluate t2 Worker parte 1

```python
# 3) Desvio em relação à baseline (rondas 1-3)
if record.baseline_established:
    dist_dev = raw_distance - record.baseline_distance
    if dist_dev > 0.80:
        violation_score += 3
        shift_signals += 1
        euclidean_triggered = True
        reasons.append(f"baseline_dist={dist_dev:.4f}")
    elif dist_dev > 0.60:
        violation_score += 1
        euclidean_triggered = True

    kl_dev = raw_kl - record.baseline_kl
    if kl_dev > 0.05:
        violation_score += 3
        shift_signals += 1
        kl_triggered = True
        reasons.append(f"baseline_kl={kl_dev:.6f}")
    elif kl_dev > 0.03:
        violation_score += 1
        kl_triggered = True

# 4) Valores absolutos elevados
if raw_distance > 1.80:
    violation_score += 2
    euclidean_triggered = True
    reasons.append(f"extreme_dist={raw_distance:.4f}")
if raw_kl > 0.20:
    violation_score += 2
    kl_triggered = True
    reasons.append(f"extreme_kl={raw_kl:.6f}")

# 5) Persistência de valores elevados após baseline
if self.round_num >= 7:
    post_shift = record.distance_history[3:]
    high_count = sum(1 for d in post_shift if d > 1.00)
    if high_count >= 3:
        violation_score += 2
        euclidean_triggered = True
        reasons.append(f"persistent={high_count}")

record.t2_suspicion_score += violation_score

# Track which defenses detected this worker
if euclidean_triggered:
    record.euclidean_violations += 1
    record.detected_by.add("euclidean")
if kl_triggered:
    record.kl_divergence_violations += 1
    record.detected_by.add("kl_divergence")

# Critérios para isolar
if shift_signals >= 2:
    return True, f"T2_shift_{shift_signals}: {'; '.join(reasons)}"
if violation_score >= 6:
    return True, f"T2_score_{violation_score}: {'; '.join(reasons)}"
if record.t2_suspicion_score >= 12:
    return True, f"T2_accum_{record.t2_suspicion_score}: {'; '.join(reasons)}"

return False, ""
```

Figura 31 - Evaluate t2 Worker parte 2

Cenário T3 – Ataque de Pretensão Aleatória

Neste cenário, o *Worker* alterna, de forma aleatória, entre comportamento benigno e malicioso:

src/components/worker.py

```
def _random_alternating_attack(self, weights):
    if random.random() > 0.5:
        try:
            return [w + np.random.normal(0.0, 1.5, w.shape).astype(w.dtype) for w in weights]
        except Exception:
            return weights
    return weights
```

Figura 32 - Ataque de Pretensão Aleatória

Neste tipo de ataque, as métricas tendem a oscilar significativamente.

As rondas benignas são intercaladas com picos, e a deteção ocorre por acumulação de violações ao longo de várias rondas e por diminuição do *trust score* quando o ataque ocorre.

A decisão final de isolamento depende do histórico e do número de rondas necessário para reunir evidência suficiente.

Cenário T4 – Ataque Progressivo

Neste cenário, a intensidade do ataque aumenta à medida que as rondas decorrem:

src/components/worker.py

```
def _progressive_attack(self, weights, round_num):
    strength = min(0.1 * (round_num + 1), 2.0)
    try:
        return [w + np.random.normal(0.0, strength, w.shape).astype(np.float32) for w in weights]
    except Exception:
        return weights
```

Figura 33 - Ataque Progressivo

Tal como no T2, é aplicada deteção especializada, através do *_evaluate_t4_worker()*, que combina violações absolutas e "*minor violations*", um *score* cumulativo com análise de

tendência (regressão linear no histórico do *trust score*) e as duas defesas implementadas na arquitetura.

src/components/chief.py

```python
def _evaluate_t4_worker(self, worker_id: int, record: WorkerRecord,
                        raw_distance: float, raw_kl: float) -> Tuple[bool, str]:
    """T4 progressive detection - catch slow degradation"""
    euclidean_params = self.config.get("euclidean", {})
    kl_params = self.config.get("kl_divergence", {})
    trust_params = self.config.get("trust", {})
    cumulative_params = self.config.get("cumulative_degradation", {})

    absolute_dist_threshold = euclidean_params.get("absolute_distance_threshold", 1.20)
    absolute_kl_threshold = kl_params.get("absolute_kl_threshold", 0.25)
    suspicious_dist_threshold = euclidean_params.get("suspicious_distance_threshold", 0.80)
    suspicious_kl_threshold = kl_params.get("suspicious_kl_threshold", 0.15)

    euclidean_triggered = False
    kl_triggered = False

    # Count violations
    if raw_distance > absolute_dist_threshold:
        record.absolute_violations += 1
        euclidean_triggered = True
    elif raw_distance > suspicious_dist_threshold:
        record.minor_violations += 1
        euclidean_triggered = True

    if raw_kl > absolute_kl_threshold:
        record.absolute_violations += 1
        kl_triggered = True
    elif raw_kl > suspicious_kl_threshold:
        record.minor_violations += 1
        kl_triggered = True

    # Track which defenses detected this worker
    if euclidean_triggered:
        record.euclidean_violations += 1
        record.detected_by.add("euclidean")
    if kl_triggered:
        record.kl_divergence_violations += 1
        record.detected_by.add("kl_divergence")

    # Immediate isolation on too many absolute violations
    violations_for_isolation = trust_params.get("violations_for_immediate_isolation", 5)
    if record.absolute_violations >= violations_for_isolation:
        return True, f"absolute_viol-{record.absolute_violations}"

    # Isolation on accumulated minor violations
    minor_for_isolation = trust_params.get("minor_violations_for_isolation", 5)
    if record.minor_violations >= minor_for_isolation:
        return True, f"minor_viol-{record.minor_violations}"

    # Cumulative score (trend + distance + KL)
    if record.rounds_participated >= cumulative_params.get("min_rounds_for_detection", 5):
        score = self._calculate_t4_score(record, cumulative_params)
        isolation_threshold = cumulative_params.get("isolation_score_threshold", 0.60)
        if score >= isolation_threshold:
            return True, f"cumul_score-{score:.3f}"

    return False, ""
```

Figura 34 - Evaluate t4 Worker

No ataque progressivo, não existe um *spike* óbvio nas rondas iniciais, em vez disso, o *trust score* degrada-se de forma gradual.

O isolamento de até cinco *Workers* ocorre quando existe evidência estatística suficiente de degradação cumulativa ou quando se verifica acumulação de violações absolutas e/ou "*minor violations"*.

4 Resultados obtidos e Análise Experimental

Após a execução completa do sistema, os resultados das simulações foram armazenados e organizados na diretoria simulation_runs/.

Cada execução corresponde a um cenário de ataque (T1-T4) e foi realizada com e sem a *BL*, sendo os respetivos artefactos agrupados em subpastas.

Os artefactos produzidos incluem registos de execução, os dados locais por *Worker*, os ficheiros de resultados e gráficos gerados, bem como um relatório consolidado que reúne, de forma completa, os resultados das diferentes execuções.

Neste capítulo, apresentam-se e analisam-se os resultados obtidos, com o objetivo de avaliar o desempenho das duas estratégias de defesa (Distância Euclidiana e Divergência *KL*), bem como a influência da *BL* nos resultados e, por fim, o desempenho da Aprendizagem Federada.

A análise inicia-se com um resumo global dos resultados nos diferentes cenários.

De seguida, procede-se a uma análise detalhada de cada cenário de ataque, a uma comparação das duas estratégias de defesa, a uma avaliação da influência da *BL* e, por fim, à análise do desempenho da Aprendizagem Federada.

4.1 Resumo global dos resultados por cada cenário de ataque

A Tabela 3 apresenta um resumo consolidado dos resultados obtidos em cada cenário, comparando as configurações sem e com *BL*.

Em todos os cenários, foram utilizados 20 *Workers*, variando apenas o número de participantes maliciosos:

- Um *Worker* malicioso nos cenários T1 e T2.
- Cinco *Workers* maliciosos nos cenários T3 e T4.

Tabela 3 - Resumo dos resultados por cada cenário

Cenário	Blockchain	Nº Workers	Nº maliciosos	Maliciosos detetados	Maliciosos isolados	Taxa de deteção (%)	Falsos positivos (detetados)	Suspeitos (não isolados)
T1	Não	20	1	1	1	100	0	0
T1	Sim	20	1	1	1	100	0	0
T2	Não	20	1	1	1	100	0	3
T2	Sim	20	1	1	1	100	0	2
T3	Não	20	5	5	5	100	0	0
T3	Sim	20	5	5	5	100	0	0
T4	Não	20	5	5	5	100	0	0
T4	Sim	20	5	5	5	100	0	0

Os resultados demonstram que, independentemente da utilização da *BL*, a taxa de deteção foi de 100% em todos os cenários, uma vez que o número de maliciosos detetados coincide com o número de maliciosos existentes e todos foram isolados com sucesso.

Adicionalmente, a tabela 3 indica que não ocorreram falsos positivos em qualquer cenário, isto é, não foram detetados como maliciosos quaisquer *Workers* fidedignos.

A principal diferença relevante verifica-se no cenário T2, no qual permanecem alguns *Workers* assinalados como suspeitos, mas não isolados (3 sem *BL* e 2 com *BL*).

Este resultado indica que, embora o isolamento final tenha sido aplicado apenas aos participantes efetivamente maliciosos, o cenário T2 tende a gerar mais sinais intermédios de suspeição, sem comprometer a deteção e o isolamento dos atacantes.

A taxa de deteção de 100% e a ausência de falsos positivos resultam do uso de um perfil de referência (*baseline*) e de limiares conservadores, que só isolam *Workers* quando o desvio é consistente/acumulado ao longo das rondas.

No T2, a presença de suspeitos não isolados decorre da natureza mais subtil do ataque de pretensão, que gera desvios intermédios suficientes para sinalização, mas insuficientes para isolamento.

4.2 Análise dos resultados por cada cenário de ataque

4.2.1 Cenário T1 – Ataque Estático

No cenário T1, em que o comportamento malicioso é constante, existe apenas um *Worker* malicioso.

Os resultados demonstram que ambas as defesas implementadas detetaram e isolaram corretamente o atacante, com e sem *BL*.

- A taxa de deteção foi de 100%, não se verificando falsos positivos nem *Workers* assinalados como suspeitos.

Este cenário evidencia a elevada eficácia da Distância Euclidiana e da Divergência *KL* neste tipo de ataque.

Este desempenho resulta de o desvio ser persistente e, por isso, ultrapassar de forma consistente os limiares face ao perfil de referência (*baseline*), permitindo o isolamento sem falsos positivos.

4.2.2 Cenário T2 – Ataque de Pretensão Estático

No cenário T2, existe igualmente apenas um *Worker* malicioso, que inicia o processo com comportamento benigno e, posteriormente, passa a atuar de forma maliciosa.

Neste cenário, as duas defesas também conseguiram detetar e isolar o único *Worker* malicioso, com e sem *BL*.

Contudo, alguns *Workers* foram assinalados como suspeitos, sem serem isolados.

Esta observação reflete a natureza mais subtil do ataque, que tende a produzir sinais intermédios de suspeição antes de se confirmar uma degradação suficientemente clara para justificar o isolamento do atacante (*Worker* malicioso).

Os suspeitos não isolados surgem porque a mudança de estado é gradual e menos marcada, levando a sinais acima do limiar de suspeição, mas insuficientes para cumprir os critérios conservadores de isolamento.

4.2.3 Cenário T3 – Ataque de Pretensão Aleatória

No cenário T3, os cinco *Workers* maliciosos alternam, de forma aleatória, entre comportamentos benignos e maliciosos.

Os resultados indicam que a deteção ocorre por acumulação de violações ao longo de várias rondas e por degradação do *trust score*.

Nas execuções com e sem *BL*, os cinco *Workers* maliciosos foram corretamente detetados e isolados, sem ocorrência de falsos positivos, nem de *Workers* assinalados como suspeitos.

A alternância aleatória exige acumulação de evidência ao longo das rondas, pelo que o isolamento ocorre quando as violações repetidas degradam o *trust score* de forma consistente.

4.2.4 Cenário T4 – Ataque Progressivo

No cenário T4, a intensidade do ataque aumenta de forma gradual ao longo das rondas, sendo considerados cinco *Workers* maliciosos.

A deteção baseia-se na acumulação de violações absolutas e *minor violations*, bem como na análise da tendência do *trust score* ao longo do tempo.

Os resultados demonstram que, apesar da ausência de picos evidentes nas fases iniciais, a degradação progressiva do comportamento é corretamente identificada, conduzindo ao isolamento dos cinco *Workers* maliciosos.

Tal como nos restantes cenários, verificou-se uma taxa de deteção de 100%.

A progressividade não gera *outliers* imediatos, mas a acumulação de pequenas violações e a tendência descendente do *trust score* tornam o desvio suficientemente consistente para isolamento.

4.3 Análise comparativa das duas estratégias de defesa

Na Figura 35, apresenta-se, para cada cenário, o número de *Workers* maliciosos detetados por cada defesa (Distância Euclidiana e Divergência *KL*), bem como a comparação entre as configurações sem *BL* e com *BL*.

A linha vermelha tracejada indica o número total de *Workers* maliciosos definidos em cada cenário (T1/T2 – 1 *Worker* malicioso, T3/T4 – 5 *Workers* maliciosos).

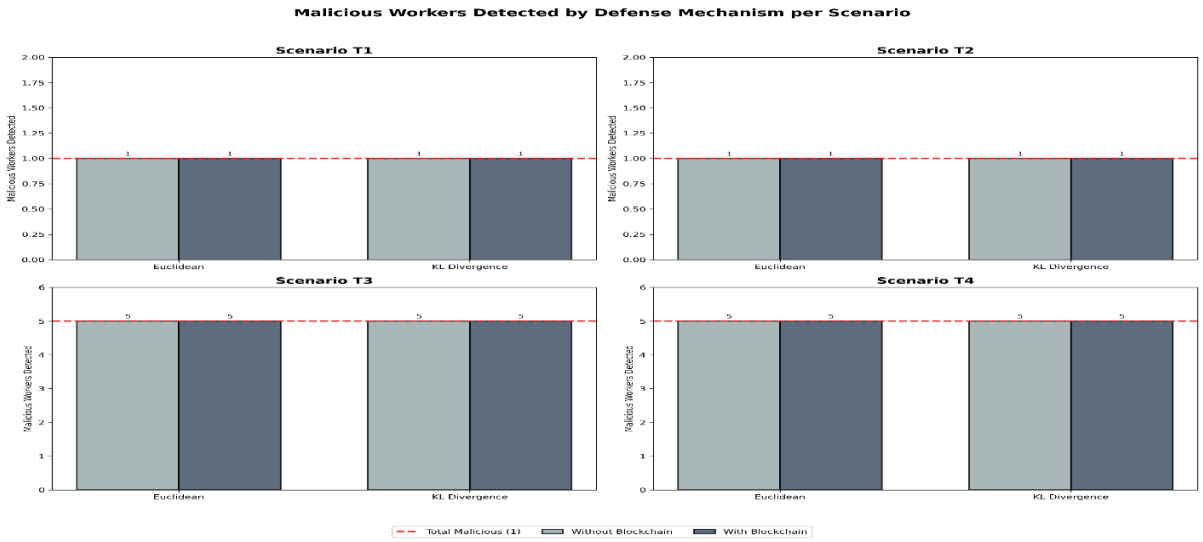

Figura 35 – Workers maliciosos detetados por cenário (com e sem BL).

Importa salientar que os resultados apresentados refletem o desempenho do mecanismo de deteção global implementado no *Chief*. Neste mecanismo, a Distância Euclidiana e a Divergência *KL* não atuam como classificadores independentes, pelo contrário, são usadas em conjunto para suportar uma decisão única de isolamento.

Por essa razão, sempre que um *Worker* é isolado, essa deteção é registada simultaneamente nas duas métricas, originando valores idênticos para a Distância Euclidiana e para a Divergência *KL*.

Adicionalmente, a deteção baseia-se num perfil de referência (*baseline)* do comportamento esperado dos *Workers* fidedignos, construído a partir do histórico observado no decorrer da simulação.

A comparação com esse perfil é efetuada através de limiares exigentes (critérios conservadores), definidos para reduzir falsos positivos, o que implica que apenas os desvios consistentes ou acumulados ao longo das rondas conduzem ao isolamento.

Esta combinação explica a semelhança dos resultados entre as duas métricas e está alinhada com a literatura[84], que reporta desempenhos próximos entre a Divergência *KL* e medidas de distância convencionais, como a Distância Euclidiana, quando aplicadas como medidas de dissimilaridade na mesma tarefa.

4.4 Influência da *BL* nos resultados

Ao comparar as simulações realizadas com e sem *BL*, conclui-se que a utilização da *BL* não altera a taxa de deteção de contribuições maliciosas.

Em todos os cenários, o número de *Workers* maliciosos detetados e isolados é idêntico.

Estes resultados são consistentes com a arquitetura do sistema, uma vez que a deteção é efetuada no *Chief*, com base no comportamento das atualizações do modelo. Por não integrar o mecanismo de deteção (apenas o reforça a nível de autorização e auditoria), a *BL* não influencia a decisão de isolamento.

A *BL* desempenha um papel complementar, ao suportar o controlo de acesso, a autorização e a auditabilidade.

Deste modo, a *BL* reforça a confiabilidade do sistema, sem interferir diretamente nos critérios de deteção.

4.5 Desempenho da Aprendizagem Federada

O desempenho da Aprendizagem Federada é analisado nas Figuras 36, 37 e 38, que apresentam, respetivamente, a latência por ronda, a velocidade de execução e o número de clientes participantes por ronda.

A Figura 36 evidencia que a latência total por ronda é dominada pelo componente de comunicação e *overhead*, uma vez que ambas as curvas são praticamente coincidentes.

Observa-se que, nas primeiras rondas, as latências são mais elevadas, seguindo-se de um regime globalmente estável, embora se verifiquem picos pontuais em determinadas rondas.

Este comportamento é expectável devido ao arranque do processo (inicialização do servidor e *Workers* e primeira sincronização do modelo), que introduz overhead adicional.

Os picos podem resultar de variações transitórias na comunicação e no processamento, uma vez que o treino local se mantém praticamente estável.

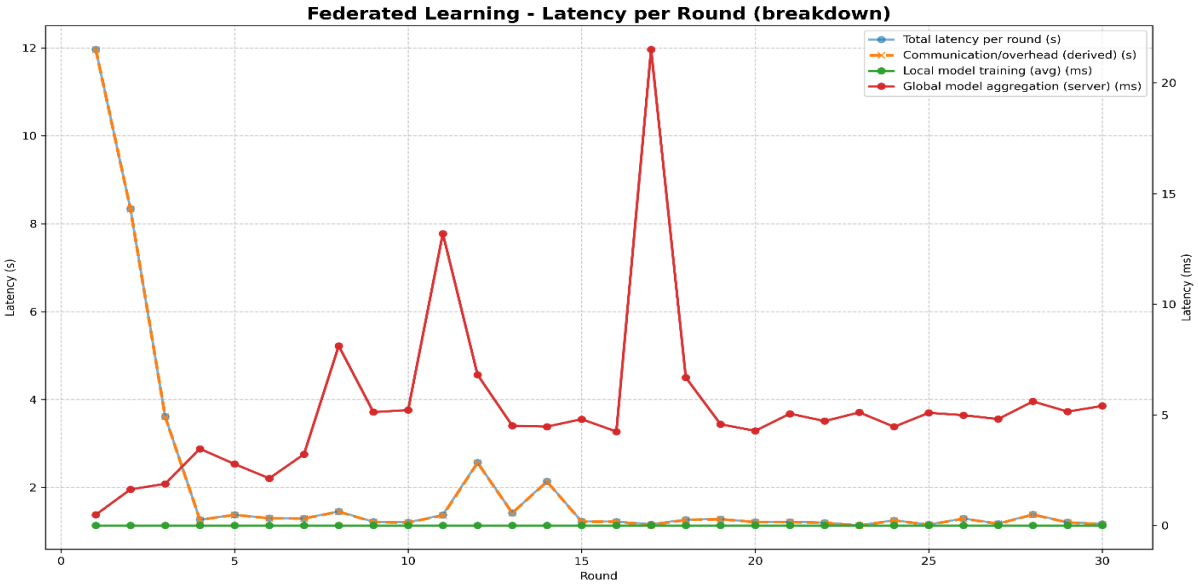

Figura 36 – Latência por ronda no treino federado.

Na Figura 37, observa-se a velocidade da Aprendizagem Federada ao longo das trinta rondas.

Os valores apresentados correspondem a taxas instantâneas (rondas por minuto), e não a uma contagem acumulada, refletindo, assim, a rapidez da execução de cada ronda. As oscilações observadas acompanham diretamente a latência por ronda: quando a latência aumenta, a velocidade (rondas por minuto) diminui e, vice-versa.

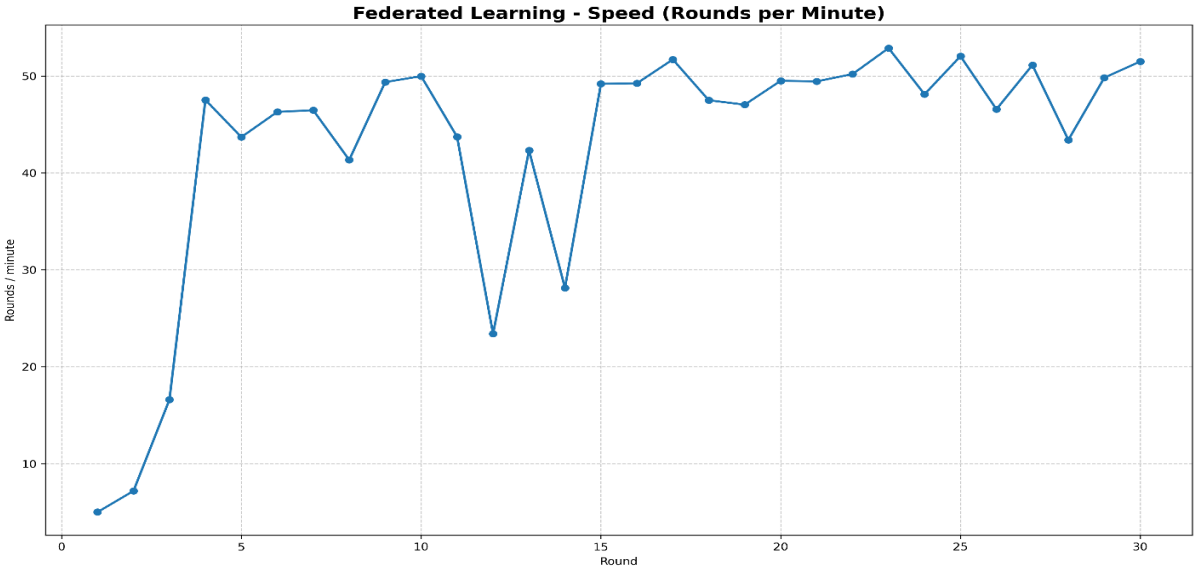

Figura 37 – Velocidade da Aprendizagem Federada ao longo das trinta rondas.

Por fim, a Figura 38 demonstra que o número de clientes participantes por ronda estabiliza em dezasseis.

Esta diferença, face aos vinte *Workers* disponíveis, resulta da configuração do servidor de Aprendizagem Federada, que utiliza uma fração de participação por ronda de 0.8.

Esta abordagem é frequentemente utilizada para reduzir o custo de comunicação e aumentar a escalabilidade.

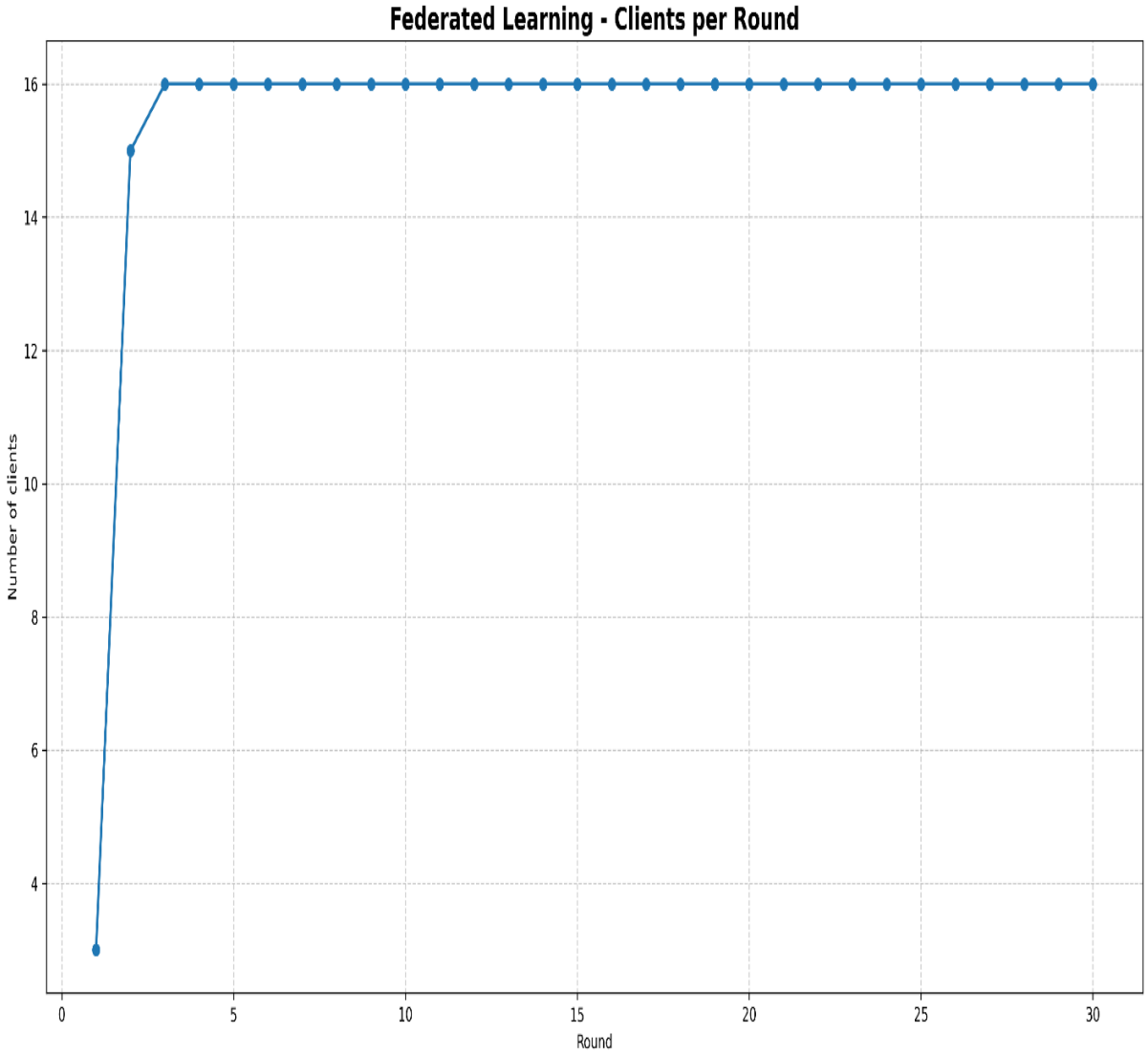

Figura 38 – Número de clientes participantes por ronda.

4.6 Relatórios gerados

Após o término das simulações, são gerados vários relatórios em formato Excel:

- Em cada subpasta, *simulation_runs/<cenário>_<timestamp>/,* é criado um ficheiro *results.xlsx*.
- Este ficheiro inclui o resumo da execução e folhas dedicadas aos resultados de classificação *(reliable/suspicious/malicious/isolated/eliminated*), bem como estatísticas por defesa.
- É ainda gerado, na pasta *simulation_runs/,* um ficheiro consolidado, designado *consolidated_report*, que agrega os resultados de todas as execuções, com e sem *BL*.
- Na pasta *simulation_runs/visualizations/,* é criado o ficheiro *fl_efficiency_summary,* que resume as métricas globais do treino federado, derivadas do *fl_metrics.pkl*.

  Entre estas métricas incluem-se a latência média por ronda, a agregação, a comunicação/sobrecarga, o número de clientes por ronda e a velocidade média.

  Estes relatórios permitem assegurar rastreabilidade e reprodutibilidade das execuções, suportando a análise comparativa entre cenários e entre configurações com e sem *BL*.

4.7 Resumo e discussão dos resultados

A análise dos resultados evidencia diferenças no comportamento das duas defesas implementadas (Distância Euclidiana e Divergência *KL*), em função do tipo de ataque realizado.

No cenário T1 (Ataque Estático), ambas as métricas apresentaram elevada eficácia, devido à persistência do desvio, o que permite a deteção por limiar acumulativo.

No cenário T2 (Ataque de Pretensão Estático), verifica-se que a utilização exclusiva de limiares instantâneos é insuficiente, sendo necessário recorrer a um histórico longitudinal, de modo a identificar a mudança de estado.

No cenário T3 (Ataque de Pretensão Aleatória), observa-se que picos isolados tendem a ser insuficientes para isolar um *Worker*, sendo necessária a acumulação de evidência ao longo das rondas.

No cenário T4 (Ataque Progressivo), pequenas perturbações sucessivas conduzem a degradação cumulativa, mas sem gerar *outliers* imediatos, tornando a deteção dependente de sinais agregados ao longo das rondas.

A utilização conjunta da Distância Euclidiana e da Divergência *KL* revelou-se bastante pertinente neste trabalho, por combinar sensibilidade a deslocações absolutas com sensibilidade a alterações na distribuição estatística dos pesos.

Contudo, importa reconhecer que a abordagem permanece dependente da definição de limiares e poderá degradar o desempenho em ambientes com elevada heterogeneidade legítima entre os *Workers*.

Adicionalmente, embora esta discussão permita interpretar os resultados obtidos nos quatro cenários de ataque, a avaliação adotada é predominantemente experimental e comparativa, centrada no desempenho do sistema sob diferentes cenários (T1-T4) e na observação do seu comportamento em ambiente simulado e controlado.

Dado que o objetivo central consiste na validação da eficácia funcional da arquitetura proposta e na avaliação da sua capacidade de resistência face aos padrões de ataque considerados, não se considerou necessária uma análise estatística dedutiva adicional nesta fase.

Ainda assim, a interpretação dos resultados poderia ser enriquecida através de uma abordagem mais analítica, incluindo testes estatísticos, análise de sensibilidade aos limiares definidos e uma comparação sistemática com métodos robustos de agregação do estado da arte.

5 Conclusões e Sugestões para Trabalhos Futuros

Esta dissertação teve como objetivo abordar o problema da garantia de integridade e segurança nas *VANETs* face a ataques estáticos, ataques de pretensão estáticos e aleatórios e ataques progressivos, recorrendo à Aprendizagem Federada para reforçar a privacidade.

Para atingir este objetivo, foi implementada uma simulação com recurso a várias ferramentas.

O *SUMO* foi utilizado para gerar a mobilidade, o *NS-3,* com *LTE/EPC* e parametrização "*5G-like*", assegurou as comunicações e o *Flower,* em conjunto com o *TensorFlow/Keras,* foi responsável pelo treino federado.

A integração destas tecnologias, via *TraCI,* e a automatização de oito execuções, com e sem *BL*, através de um *script* em *Python*, permitiram obter resultados comparáveis e reprodutíveis, possibilitando responder à questão de investigação proposta.

O objetivo principal de criar e testar uma arquitetura com *LSTM*, Aprendizagem Federada e *BL,* suportada por duas estratégias de defesa, foi alcançado com sucesso.

Adicionalmente, os objetivos específicos de executar cenários de ataque com e sem *BL* também foram cumpridos.

Em todas as simulações (T1-T4), verificou-se uma taxa de deteção de 100%, com e sem *BL,* e todos os *Workers* maliciosos foram isolados pelas defesas implementadas.

Não se registaram falsos positivos, observando-se apenas alguns *Workers* benignos assinalados como "*suspicious*" no cenário T2, embora apenas o *Worker* malicioso tenha sido efetivamente isolado.

Entre as principais contribuições desta dissertação, destaca-se a criação de uma arquitetura integrada e operacional para *VANETs,* que combina *LSTM*, treino descentralizado por Aprendizagem Federada, mecanismos de privacidade, controlo de acesso e auditoria (*Hyperledger Indy* e *Hyperledger Fabric*), bem como um processo automatizado de recolha e análise de resultados, com geração de relatórios em Excel e de gráficos.

Os resultados demonstraram, ainda, que duas métricas de defesa distintas apresentaram um desempenho equivalente na deteção de *Workers* maliciosos.

Apesar destes avanços, foram identificadas limitações. Em primeiro lugar, a avaliação baseou-se em simulação e num conjunto fixo de parâmetros (vinte *Workers* e trinta rondas). Em segundo lugar, a rede foi modelada com *LTE/EPC* "*5G-like*", e não com uma pilha completa de *5G NR*.

Relativamente a sugestões para trabalhos futuros:

- Recomenda-se a avaliação de mais tipos de cenários, a consideração de múltiplas densidades veiculares, a utilização de mapas urbanos mais heterogéneos, a incorporação de ataques mais complexos e o desenvolvimento de estratégias de defesa adicionais.
- Adicionalmente, considera-se relevante aprofundar o estudo dos mecanismos de privacidade e de segurança e otimizar o desenho da camada da *BL*, de modo a reduzir o *overhead,* mantendo a auditabilidade.

Em síntese:

Esta dissertação apresenta impacto académico ao validar uma arquitetura integrada e ao evidenciar, em ambiente controlado, que é possível alcançar deteção robusta e rastreabilidade, sem comprometer, de forma substancial, a operacionalidade do sistema.

Referências Bibliográficas

[1] M. A. Al-shareeda, M. A. Alazzawi, M. Anbar, S. Manickam, and A. K. Al-Ani, "A Comprehensive Survey on Vehicular Ad Hoc Networks (VANETs)," in 2021 International Conference on Advanced Computer Applications (ACA), Maysan, Iraq: IEEE, Jul. 2021, pp. 156–160, doi: 10.1109/ACA52198.2021.9626779.

[2] M. A. Al-Shareeda, M. Anbar, I. H. Hasbullah, and S. Manickam, "Survey of Authentication and Privacy Schemes in Vehicular ad hoc Networks," IEEE Sensors Journal, vol. 21, no. 2, pp. 2422–2433, Jan. 2021, doi: 10.1109/JSEN.2020.3021731.

[3] D. Manivannan, S. S. Moni, and S. Zeadally, "Secure authentication and privacy-preserving techniques in Vehicular Ad-hoc NETworks (VANETs)," Vehicular Communications, vol. 25, p. 100247, Oct. 2020, doi: 10.1016/j.vehcom.2020.100247.

[4] "Techniques of Early Incident Detection and Traffic Monitoring Centre in VANETs: A Review," ResearchGate, Aug. 2025, doi: 10.12720/jcm.15.12.896-904.

[5] M. A. Alazzawi, H. Lu, A. A. Yassin, and K. Chen, "Efficient Conditional Anonymity With Message Integrity and Authentication in a Vehicular Ad-Hoc Network," IEEE Access, vol. 7, pp. 71424–71435, 2019, doi: 10.1109/ACCESS.2019.2919973.

[6] M. A. Al-shareeda, M. Anbar, I. H. Hasbullah, S. Manickam, N. Abdullah, and M. M. Hamdi, "Review of Prevention schemes for Replay Attack in Vehicular Ad hoc Networks (VANETs)," in 2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP), Sep. 2020, pp. 394–398, doi: 10.1109/ICICSP50920.2020.9232047.

[7] J. Jakubiak and Y. Koucheryavy, "State of the Art and Research Challenges for VANETs," in 2008 5th IEEE Consumer Communications and Networking Conference, Jan. 2008, pp. 912–916, doi: 10.1109/ccnc08.2007.212.

[8] J. J. Blum, A. Eskandarian, and L. J. Hoffman, "Challenges of Intervehicle Ad Hoc Networks," IEEE Transactions on Intelligent Transportation Systems, vol. 5, no. 4, pp. 347–351, Dec. 2004, doi: 10.1109/TITS.2004.838218.

[9] Y. Toor, P. Muhlethaler, A. Laouiti, and A. La Fortelle, "Vehicle Ad Hoc networks: applications and related technical issues," IEEE Communications Surveys and Tutorials, vol. 10, no. 3, pp. 74–88, 2008, doi: 10.1109/COMST.2008.4625806.

[10] T. Chatterjee, R. Karmakar, G. Kaddoum, S. Chattopadhyay, and S. Chakraborty, "A Survey of VANET/V2X Routing From the Perspective of Non-Learning- and Learning-Based Approaches," IEEE Access, vol. 10, pp. 23022–23050, 2022, doi:

10.1109/ACCESS.2022.3152767.
[11] S. Chen et al., “Vehicle-to-Everything (v2x) Services Supported by LTE-Based Systems and 5G,” IEEE Communications Standards Magazine, vol. 1, no. 2, pp. 70–76, 2017, doi: 10.1109/MCOMSTD.2017.1700015.
[12] Y. L. Morgan, “Notes on DSRC and WAVE Standards Suite: Its Architecture, Design, and Characteristics,” IEEE Communications Surveys and Tutorials, vol. 12, no. 4, pp. 504–518, 2010, doi: 10.1109/SURV.2010.033010.00024.
[13] R. Mishra, A. Singh, and R. Kumar, “VANET security: Issues, challenges and solutions,” in 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Mar. 2016, pp. 1050–1055, doi: 10.1109/ICEEOT.2016.7754846.
[14] A. Mahmood, W. E. Zhang, and Q. Z. Sheng, “Software-Defined Heterogeneous Vehicular Networking: The Architectural Design and Open Challenges,” Future Internet, vol. 11, no. 3, p. 70, Mar. 2019, doi: 10.3390/fi11030070.
[15] Y. Hui, Z. Su, T. H. Luan, and J. Cai, “A Game Theoretic Scheme for Optimal Access Control in Heterogeneous Vehicular Networks,” IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 12, pp. 4590–4603, Dec. 2019, doi: 10.1109/TITS.2019.2894716.
[16] S. Ilarri, T. Delot, and R. Trillo-Lado, “A Data Management Perspective on Vehicular Networks,” IEEE Communications Surveys and Tutorials, vol. 17, no. 4, pp. 2420–2460, 2015, doi: 10.1109/COMST.2015.2472395.
[17] T. Nadia, A. Mourad, M. Hamouma, and K. Hamoudi, “A Survey on Vehicular Ad-Hoc Networks Routing Protocols: Classification and Challenges,” Journal of Digital Information Management, vol. 17, no. 4, pp. 227–244, Aug. 2019, doi: 10.6025/jdim/2019/17/4/227-244.
[18] G. Bendiab, A. Hameurlaine, G. Germanos, N. Kolokotronis, and S. Shiaeles, “Autonomous Vehicles Security: Challenges and Solutions Using Blockchain and Artificial Intelligence,” IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 4, pp. 3614–3637, Apr. 2023, doi: 10.1109/TITS.2023.3236274.
[19] G. J. M. Read, S. Shorrock, G. H. Walker, and P. M. Salmon, “State of science: evolving perspectives on human error,” Ergonomics, vol. 64, no. 9, pp. 1091–1114, Sep. 2021, doi: 10.1080/00140139.2021.1953615.
[20] K. M. Ali Alheeti and K. McDonald-Maier, “An intelligent security system for autonomous cars based on infrared sensors,” in 2017 23rd International Conference on Automation and Computing (ICAC), Sep. 2017, pp. 1–5, doi: 10.23919/IConAC.2017.8082084.

[21] I. Berger, R. Rieke, M. Kolomeets, A. Chechulin, and I. Kotenko, “Comparative Study of Machine Learning Methods for In-Vehicle Intrusion Detection,” in Computer Security, Lecture Notes in Computer Science, vol. 11387, Cham: Springer International Publishing, 2019, pp. 85–101, doi: 10.1007/978-3-030-12786-2_6.
[22] M. Jagielski, N. Jones, C.-W. Lin, C. Nita-Rotaru, and S. Shiraishi, “Threat Detection for Collaborative Adaptive Cruise Control in Connected Cars,” in Proceedings of the 11th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Stockholm, Sweden: ACM, Jun. 2018, pp. 184–189, doi: 10.1145/3212480.3212492.
[23] K. M. Ali Alheeti and K. McDonald-Maier, “Intelligent intrusion detection in external communication systems for autonomous vehicles,” Systems Science and Control Engineering, vol. 6, no. 1, pp. 48–56, Jan. 2018, doi: 10.1080/21642583.2018.1440260.
[24] C. Catal, H. Gunduz, and A. Ozcan, “Malware Detection Based on Graph Attention Networks for Intelligent Transportation Systems,” Electronics, vol. 10, no. 20, Art. no. 2534, 2021, doi: 10.3390/electronics10202534.
[25] R. Rahal, A. Amara Korba, N. Ghoualmi-Zine, Y. Challal, and M. Y. Ghamri-Doudane, “AntibotV: A Multilevel Behaviour-Based Framework for Botnets Detection in Vehicular Networks,” Journal of Network and Systems Management, vol. 30, no. 1, Art. no. 15, 2022, doi: 10.1007/s10922-021-09630-8.
[26] T. Zeng, O. Semiari, M. Chen, W. Saad, and M. Bennis, “Federated Learning on the Road Autonomous Controller Design for Connected and Autonomous Vehicles,” IEEE Transactions on Wireless Communications, vol. 21, no. 12, pp. 10407–10423, Dec. 2022, doi: 10.1109/TWC.2022.3183996.
[27] S. R. Pokhrel and J. Choi, “Federated Learning With Blockchain for Autonomous Vehicles: Analysis and Design Challenges,” IEEE Transactions on Communications, vol. 68, no. 8, pp. 4734–4746, Aug. 2020, doi: 10.1109/TCOMM.2020.2990686.
[28] A. M. Elbir, B. Soner, S. Çöleri, D. Gündüz, and M. Bennis, “Federated Learning in Vehicular Networks,” in 2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Sep. 2022, pp. 72–77, doi: 10.1109/MeditCom55741.2022.9928621.
[29] U. Majeed, L. U. Khan, A. Yousafzai, Z. Han, B. J. Park, and C. S. Hong, “ST-BFL: A Structured Transparency Empowered Cross-Silo Federated Learning on the Blockchain Framework,” IEEE Access, vol. 9, pp. 155634–155650, 2021, doi: 10.1109/ACCESS.2021.3128622.
[30] Q. Li et al., “A Survey on Federated Learning Systems: Vision, Hype and Reality for

Data Privacy and Protection," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 4, pp. 3347–3366, Apr. 2023, doi: 10.1109/TKDE.2021.3124599.
[31] P. Kairouz et al., "Advances and Open Problems in Federated Learning," arXiv, Mar. 2021, arXiv:1912.04977, doi: 10.48550/arXiv.1912.04977.
[32] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," arXiv, Feb. 2019, arXiv:1902.04885, doi: 10.48550/arXiv.1902.04885.
[33] T. Overman and D. Klabjan, "Continuous-Time Analysis of Federated Averaging," arXiv, Jan. 2025, arXiv:2501.18870, doi: 10.48550/arXiv.2501.18870.
[34] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "FedAvg with Fine Tuning: Local Updates Lead to Representation Learning," arXiv, May 2022, arXiv:2205.13692, doi: 10.48550/arXiv.2205.13692.
[35] T. Sun, D. Li, and B. Wang, "Decentralized Federated Averaging," arXiv, Apr. 2021, arXiv:2104.11375, doi: 10.48550/arXiv.2104.11375.
[36] A. Ahmad, W. Luo, and A. Robles-Kelly, "Robust federated learning under statistical heterogeneity via hessian-weighted aggregation," Machine Learning, vol. 112, no. 2, pp. 633–654, Feb. 2023, doi: 10.1007/s10994-022-06292-8.
[37] X. Wang, Z. Li, S. Jin, and J. Zhang, "Achieving Linear Speedup in Asynchronous Federated Learning with Heterogeneous Clients," arXiv, Feb. 2024, arXiv:2402.11198, doi: 10.48550/arXiv.2402.11198.
[38] K. Khan, "FedEmerge: An Entropy-Guided Federated Learning Method for Sensor Networks and Edge Intelligence," Sensors, vol. 25, no. 12, p. 3728, Jun. 2025, doi: 10.3390/s25123728.
[39] X. Feng, H. Liu, H. Yang, Q. Xie, and L. Wang, "Batch-Aggregate: Efficient Aggregation for Private Federated Learning in VANETs," IEEE Transactions on Dependable and Secure Computing, vol. 21, no. 5, pp. 4939–4952, Sep. 2024, doi: 10.1109/TDSC.2024.3364371.
[40] X. Feng, X. Wang, H. Liu, H. Yang, and L. Wang, "A Privacy-Preserving Aggregation Scheme With Continuous Authentication for Federated Learning in VANETs," IEEE Transactions on Vehicular Technology, vol. 73, no. 7, pp. 9465–9477, Jul. 2024, doi: 10.1109/TVT.2024.3369942.
[41] K. Cui, X. Feng, L. Wang, and Z. Ying, "Secure Aggregation With Logarithmic Overhead for Federated Learning in VANETs," IEEE Transactions on Consumer Electronics, vol. 71, no. 1, pp. 2072–2089, Feb. 2025, doi: 10.1109/TCE.2025.3533750.
[42] D. Ye, R. Yu, M. Pan, and Z. Han, "Federated Learning in Vehicular Edge Computing: A Selective Model Aggregation Approach," IEEE Access, vol. 8, pp. 23920–23935, 2020, doi:

10.1109/ACCESS.2020.2968399.
[43] T. S. Balaji, R. Hariharan, G. S. P, S. Balaji, and S. P. Bharathi, “Privacy-Preserving Data Aggregation in Vehicular Ad Hoc Networks Using Deep Learning Techniques,” in 2024 International Conference on Cybernation and Computation (CYBERCOM), Nov. 2024, pp. 591–596, doi: 10.1109/CYBERCOM63683.2024.10803136.
[44] W. Ali, I. U. Din, A. Almogren, and J. J. P. C. Rodrigues, “Federated Learning-Based Privacy-Aware Location Prediction Model for Internet of Vehicular Things,” IEEE Transactions on Vehicular Technology, vol. 74, no. 2, pp. 1968–1978, Feb. 2025, doi: 10.1109/TVT.2024.3368439.
[45] L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac, “Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network,” arXiv, Mar. 2017, arXiv:1703.09752, doi: 10.48550/arXiv.1703.09752.
[46] Z. Wu and S. King, “Investigating gated recurrent neural networks for speech synthesis,” arXiv, Jan. 2016, arXiv:1601.02539, doi: 10.48550/arXiv.1601.02539.
[47] Z. Li et al., “Towards Binary-Valued Gates for Robust LSTM Training,” arXiv, Jun. 2018, arXiv:1806.02988, doi: 10.48550/arXiv.1806.02988.
[48] F. Landi, L. Baraldi, M. Cornia, and R. Cucchiara, “Working Memory Connections for LSTM,” Neural Networks, vol. 144, pp. 334–341, Dec. 2021, doi: 10.1016/j.neunet.2021.08.030.
[49] N. Youness, A. Mostafa, M. A. Sobh, A. M. Bahaa, and K. Nagaty, “VeMisNet: Enhanced Feature Engineering for Deep Learning-Based Misbehavior Detection in Vehicular Ad Hoc Networks,” Journal of Sensor and Actuator Networks, vol. 14, no. 5, p. 100, Oct. 2025, doi: 10.3390/jsan14050100.
[50] A. R. H. Adamou, L. Mokdad, J. Ben Othman, and A. Benzerbadj, “Hybrid Deep Learning Optimization for Accurate Trajectory Prediction in Vehicular Networks,” in ICC 2025 - IEEE International Conference on Communications, Jun. 2025, pp. 4197–4202, doi: 10.1109/ICC52391.2025.11162119.
[51] A. H. Adamou, L. Mokdad, J. Ben Othman, and A. Benzerbadj, “VANET-DEEP-MP: A Deep Learning Model for Mobility Prediction in VANET,” in GLOBECOM 2024 - 2024 IEEE Global Communications Conference, Dec. 2024, pp. 3727–3732, doi: 10.1109/GLOBECOM52923.2024.10901633.
[52] N. Rezazadeh, M. A. Amirabadi, and M. H. Kahaei, “Deep learning-based location prediction in VANET,” IET Intelligent Transport Systems, vol. 18, no. 9, pp. 1574–1587, 2024, doi: 10.1049/itr2.12529.

[53] S. S. Sepasgozar and S. Pierre, “Fed-NTP: A Federated Learning Algorithm for Network Traffic Prediction in VANET,” IEEE Access, vol. 10, pp. 119607–119616, 2022, doi: 10.1109/ACCESS.2022.3221970.
[54] L. Mou, P. Zhao, H. Xie, and Y. Chen, “T-LSTM: A Long Short-Term Memory Neural Network Enhanced by Temporal Information for Traffic Flow Prediction,” IEEE Access, vol. 7, pp. 98053–98060, 2019, doi: 10.1109/ACCESS.2019.2929692.
[55] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, “LSTM network: a deep learning approach for short-term traffic forecast,” IET Intelligent Transport Systems, vol. 11, no. 2, pp. 68–75, 2017, doi: 10.1049/iet-its.2016.0208.
[56] R. Gupta et al., “VAHAK: A Blockchain-based Outdoor Delivery Scheme using UAV for Healthcare 4.0 Services,” in IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Jul. 2020, pp. 255–260, doi: 10.1109/INFOCOMWKSHPS50562.2020.9162738.
[57] S. Brotsis, K. Limniotis, G. Bendiab, N. Kolokotronis, and S. Shiaeles, “On the Suitability of Blockchain Platforms for IoT Applications: Architectures, Security, Privacy, and Performance,” Computer Networks, vol. 191, p. 108005, May 2021, doi: 10.1016/j.comnet.2021.108005.
[58] Z. Zhang, X. Song, L. Liu, J. Yin, Y. Wang, and D. Lan, “Recent Advances in Blockchain and Artificial Intelligence Integration: Feasibility Analysis, Research Issues, Applications, Challenges, and Future Work,” Security and Communication Networks, vol. 2021, Art. no. 9991535, 2021, doi: 10.1155/2021/9991535.
[59] M. B. Mollah et al., “Blockchain for the Internet of Vehicles Towards Intelligent Transportation Systems: A Survey,” IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4157–4185, Mar. 2021, doi: 10.1109/JIOT.2020.3028368.
[60] M. Dibaei et al., “Investigating the Prospect of Leveraging Blockchain and Machine Learning to Secure Vehicular Networks: A Survey,” IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 2, pp. 683–700, Feb. 2022, doi: 10.1109/TITS.2020.3019101.
[61] N. Kamble, R. Gala, R. Vijayaraghavan, E. Shukla, and D. Patel, “Using Blockchain in Autonomous Vehicles,” in Artificial Intelligence and Blockchain for Future Cybersecurity Applications, pp. 285–305, 2021, doi: 10.1007/978-3-030-74575-2_15. (ResearchGate)
[62] P. K. Sharma, N. Kumar, and J. H. Park, “Blockchain-Based Distributed Framework for Automotive Industry in a Smart City,” IEEE Transactions on Industrial Informatics, vol. 15, no. 7, pp. 4197–4205, Jul. 2019, doi: 10.1109/TII.2018.2887101.

[63] C. Oham, R. Michelin, S. S. Kanhere, R. Jurdak, and S. Jha, “B-FERL: Blockchain based Framework for Securing Smart Vehicles,” arXiv, Jul. 2020, arXiv:2007.10528, doi: 10.48550/arXiv.2007.10528.
[64] N. Phull, P. Singh, M. Shabaz, and F. Sammy, “Performance Enhancement of Cluster-Based Ad Hoc On-Demand Distance Vector Routing in Vehicular Ad Hoc Networks,” Scientific Programming, vol. 2022, Art. no. 7423989, 2022, doi: 10.1155/2022/7423989.
[65] A. Bijalwan, I. Hussain, K. C. Purohit, and M. A. Kumar, “Enhanced Ant Colony Optimization for Vehicular Ad Hoc Networks Using Fittest Node Clustering,” Sustainability, vol. 15, no. 22, p. 15903, Nov. 2023, doi: 10.3390/su152215903.
[66] D. Gutiérrez-Reina, V. Sharma, I. You, and S. Toral, “Dissimilarity Metric Based on Local Neighboring Information and Genetic Programming for Data Dissemination in Vehicular Ad Hoc Networks (VANETs),” Sensors, vol. 18, no. 7, p. 2320, Jul. 2018, doi: 10.3390/s18072320.
[67] N. Nissar, N. Naja, and A. Jamali, “Securing VANETs: Multi-Objective Intrusion Detection With Variational Autoencoders,” IEEE Transactions on Consumer Electronics, vol. 70, no. 1, pp. 3867–3874, Feb. 2024, doi: 10.1109/TCE.2024.3372691.
[68] S. Borade and L. Zheng, “Euclidean Information Theory,” in 2008 IEEE International Zurich Seminar on Communications, Mar. 2008, pp. 14–17, doi: 10.1109/IZS.2008.4497265.
[69] E. Rabieinejad, A. Yazdinejad, A. Dehghantanha, R. M. Parizi, and G. Srivastava, “Secure AI and Blockchain-enabled Framework in Smart Vehicular Networks,” in 2021 IEEE Globecom Workshops (GC Wkshps), Dec. 2021, pp. 1–6, doi: 10.1109/GCWkshps52748.2021.9682140.
[70] L. Xia, Y. Sun, R. Swash, L. Mohjazi, L. Zhang, and M. A. Imran, “Smart and Secure CAV Networks Empowered by AI-Enabled Blockchain: The Next Frontier for Intelligent Safe Driving Assessment,” IEEE Network, vol. 36, no. 1, pp. 197–204, Jan. 2022, doi: 10.1109/MNET.101.2100387.
[71] A. Hammoud, H. Sami, A. Mourad, H. Otrok, R. Mizouni, and J. Bentahar, “AI, Blockchain, and Vehicular Edge Computing for Smart and Secure IoV: Challenges and Directions,” IEEE Internet of Things Magazine, vol. 3, no. 2, pp. 68–73, Jun. 2020, doi: 10.1109/IOTM.0001.1900109.
[72] Y. Qian, Y. Jiang, L. Hu, M. S. Hossain, M. Alrashoud, and M. Al-Hammadi, “Blockchain-Based Privacy-Aware Content Caching in Cognitive Internet of Vehicles,” IEEE Network, vol. 34, no. 2, pp. 46–51, Mar. 2020, doi: 10.1109/MNET.001.1900161.
[73] S. R. Pokhrel and J. Choi, “A Decentralized Federated Learning Approach for Connected

Autonomous Vehicles," in 2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Apr. 2020, pp. 1–6, doi: 10.1109/WCNCW48565.2020.9124733.
[74] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain Empowered Asynchronous Federated Learning for Secure Data Sharing in Internet of Vehicles," IEEE Transactions on Vehicular Technology, vol. 69, no. 4, pp. 4298–4311, Apr. 2020, doi: 10.1109/TVT.2020.2973651.
[75] H. Chai, S. Leng, Y. Chen, and K. Zhang, "A Hierarchical Blockchain-Enabled Federated Learning Algorithm for Knowledge Sharing in Internet of Vehicles," IEEE Transactions on Intelligent Transportation Systems, vol. 22, no. 7, pp. 3975–3986, Jul. 2021, doi: 10.1109/TITS.2020.3002712.
[76] Y. Fu, F. R. Yu, C. Li, T. H. Luan, and Y. Zhang, "Vehicular Blockchain-Based Collective Learning for Connected and Autonomous Vehicles," IEEE Wireless Communications, vol. 27, no. 2, pp. 197–203, Apr. 2020, doi: 10.1109/MNET.001.1900310.
[77] H. Liu et al., "Blockchain and Federated Learning for Collaborative Intrusion Detection in Vehicular Edge Computing," IEEE Transactions on Vehicular Technology, vol. 70, no. 6, pp. 6073–6084, Jun. 2021, doi: 10.1109/TVT.2021.3076780.
[78] M. Abdel-Basset, N. Moustafa, H. Hawash, I. Razzak, K. M. Sallam, and O. M. Elkomy, "Federated Intrusion Detection in Blockchain-Based Smart Transportation Systems," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 3, pp. 2523–2537, Mar. 2022, doi: 10.1109/TITS.2021.3119968.
[79] "Systematic Literature Review of Challenges in Blockchain Scalability," [Online]. Available: https://www.mdpi.com/2076-3417/11/20/9372. [Accessed 12 January 2025].
[80] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," Computer Science Review, vol. 34, p. 100199, Nov. 2019, doi: 10.1016/j.cosrev.2019.100199.
[81] C. Li and B. Palanisamy, "Privacy in Internet of Things: From Principles to Technologies," IEEE Internet of Things Journal, vol. 6, no. 1, pp. 488–505, Feb. 2019, doi: 10.1109/JIOT.2018.2864168.
[82] S. Lockey, N. Gillespie, D. Holm, and I. A. Someh, "A Review of Trust in Artificial Intelligence: Challenges, Vulnerabilities and Future Directions," presented at the Hawaii International Conference on System Sciences, 2021, doi: 10.24251/HICSS.2021.664.
[83] A. Chowdhury, G. Karmakar, J. Kamruzzaman, A. Jolfaei, and R. Das, "Attacks on Self-Driving Cars and Their Countermeasures: A Survey," IEEE Access, vol. 8, pp. 207308–207342, 2020, doi: 10.1109/ACCESS.2020.3037705.

[84] V. A. Koutsonikola, S. G. Petridou, A. I. Vakali, and G. I. Papadimitriou, “A new approach to web users clustering and validation: a divergence-based scheme,” International Journal of Web Information Systems, vol. 5, no. 3, pp. 348–371, Aug. 2009, doi: 10.1108/17440080910983583.