UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIAS DA COMPUTAÇÃO

PAULO HENRIQUE GONÇALVES ROCHA

UMA SOLUÇÃO PARA *OFFLOADING* COMPUTACIONAL EM VANETS BASEADA NA PREDIÇÃO DO TEMPO DE VIDA DO ENLACE

FORTALEZA

2023

PAULO HENRIQUE GONÇALVES ROCHA

UMA SOLUÇÃO PARA *OFFLOADING* COMPUTACIONAL EM VANETS BASEADA NA PREDIÇÃO DO TEMPO DE VIDA DO ENLACE

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciências da Computação do Programa de Pós-Graduação em Ciências da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Sistemas de Informação

Orientador: Prof. Dr. Paulo Antônio Leal Rêgo

Coorientador: Prof. Dr. Francisco Airton Pereira da Silva

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

R575s Rocha, Paulo Henrique Gonçalves.
Uma Solução para Offloading Computacional em VANETs Baseada na Predição do Tempo de Vida do Enlace / Paulo Henrique Gonçalves Rocha. – 2023.
112 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2023.
Orientação: Prof. Dr. Paulo Antônio Leal Rêgo.
Coorientação: Prof. Dr. Francisco Airton Pereira da Silva.

1. redes veiculares. 2. tempo de vida do enlace. 3. offloading computacional. 4. machine learning. I. Título.

CDD 005

PAULO HENRIQUE GONÇALVES ROCHA

UMA SOLUÇÃO PARA *OFFLOADING* COMPUTACIONAL EM VANETS BASEADA NA PREDIÇÃO DO TEMPO DE VIDA DO ENLACE

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciências da Computação do Programa de Pós-Graduação em Ciências da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciências da Computação. Área de Concentração: Sistemas de Informação

Aprovada em: 06 de Julho de 2023

BANCA EXAMINADORA

Prof. Dr. Paulo Antônio Leal Rêgo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Francisco Airton Pereira da Silva (Coorientador)
Universidade Federal do Piauí (UFPI)

Prof. Dr. Fernando Antônio Mota Trinta
Universidade Federal do Ceará (UFC)

Prof. Dr. José Gilvan Rodrigues Maia
Universidade Federal do Ceará (UFC)

Prof. Dr. Alisson Barbosa de Souza
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo Romero Martins Maciel
Universidade Federal de Pernambuco (UFPE)

À minha família, por me apoiaram nos meus sonhos e objetivos e que sempre acreditaram na educação e de seu poder de transformação na vida das pessoas.

AGRADECIMENTOS

À Deus, Supremo Bem, que me destes força durante todas as tribulações desse período, sustentando-me e mostrando os caminhos que eu devia seguir.

Ao Prof. Dr. Paulo Antônio Leal Rêgo por me orientar em minha dissertação de mestrado. Por toda sua paciência e compreensão durante esse trajeto acadêmico, sempre me guiando da melhor maneira possível e tentando extrair o melhor de mim. Seus valorosos conselhos e instruções foram essenciais na realização deste trabalho e de todo o mestrado acadêmico. Deixo aqui minha estima e admiração por ele.

Ao Prof. Dr. Francisco Airton por me coorientar neste trabalho e por todo o cuidado que demonstrou comigo durante o mestrado, sempre em contato para saber como eu estava, me auxiliando nos artigos e na dissertação, tanto na parte escrita, como nas pesquisas.

Ao meu colega de pesquisa, o Prof. Dr Alisson Souza, que foi fundamental nessa pesquisa, me oferecendo todo o suporte técnico sobre a área de redes veiculares, além de me auxiliar em todas as pesquisas realizadas.

Aos professores César Lincoln e Gilvan Maia por todo o suporte em alguns trabalhos realizados e em partes dessa pesquisa, principalmente em relação às técnicas de *Machine learning*.

Aos colegas do laboratório GREat e Dell Lead pelo acolhimento durante o mestrado.

Aos meus pais, Antônio de Souza Rocha e Maria Hilda Gonçalves Rocha e a minha irmã, Patrícia, que sempre foram compreensíveis sobre os desafios que essa missão possuía e que sempre foram meu apoio nas situações difíceis enfrentadas durante esses anos.

Aos meus amigos que sempre acreditaram em mim e que sempre demonstraram apoio e curiosidade em relação à pesquisa e ao mestrado.

Agradeço à banca pela paciência na avaliação do trabalho e pelas sugestões de correção do texto, assim como conselhos sobre possíveis melhorias e novos direcionamentos.

Agradeço a todos os professores do Departamento de Computação da UFC, por todo a dedicação em ensinar e em transformar seus alunos em excelentes profissionais e pessoas.

E à Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (Funcap), pelo financiamento da pesquisa de mestrado via bolsa de estudos (#6945087/2019).

"Se a meta principal de um capitão fosse preservar seu barco, ele o conservaria no porto para sempre."

(São Tomás de Aquino)

RESUMO

As redes veiculares (VANETs) possibilitam aplicações inteligentes em cenários de mobilidade urbana, como, por exemplo, dados em tempo real de tráfego ou compartilhamento de recursos de componentes da VANET. No entanto, o tempo de comunicação (tempo de vida do enlace - TVE) entre os nós é geralmente curto devido ao dinamismo dos cenários móveis veiculares. Tal duração da comunicação afeta aplicações e processos em VANETs, como o processo de tomada de decisão de quando fazer o *offloading* de uma subparte da aplicação, também conhecida como tarefa. Além disso, há pouca integração de entidades, como pedestres, em trabalhos relacionados a VANETs, principalmente no campo de aplicações de *offloading* em VFC(*Vehicular Fog Computing*). Uma VFC é uma arquitetura que utiliza usuário final ou dispositivos de borda próximos ao requisitante de *offloading* para reduzir o tempo de processamento e transmissão de tarefas. Com isso, o objetivo do trabalho é propor um modelo de arquitetura, para o *offloading* computacional em VANETs, através da predição inteligente do TVE no suporte ao processo de decisão utilizando modelos de Aprendizado de Máquina (do inglês, *Machine Learning* - ML) escolhidos com base em contexto de tráfego do requisitante de *offloading*. Primeiramente foi realizada uma modelagem para a predição do TVE utilizando ML entre nós em VANETs, a fim de realizar predições mais eficientes e consequentemente melhorar o processo de tomada de decisão no *offloading* computacional. Posteriormente foi desenvolvido um algoritmo de decisão de *offloading* para realizar o balanceamento de tarefas entre os servidores de uma VFC a partir do dispositivo do pedestre. Finalmente, foi implementada uma arquitetura que utilize modelos de ML adaptativos por região, para realizar a predição do TVE, a fim de auxiliar o processo de decisão e balanceamento de tarefas de *offloading* em uma rede VFC. Os resultados demonstraram que a modelagem de predição do TVE através de algoritmos de ML foram efetivas, chegando a reduzir a taxa de perda de tarefas em 50% em comparação a abordagem tradicional de predição, utilizada em outros trabalhos. No processo de escolha adaptativa de modelos de ML executada pela arquitetura, os resultados indicaram que a abordagem reduziu em 50% o número de falhas no número de tarefas enviadas aos servidores VFC, resultando em um aumento de mais de 8% no número de tarefas executadas com sucesso em comparação com a escolha de um único modelo para predição.

Palavras-chave: redes veiculares; tempo de vida do enlace; *offloading* computacional; *machine learning*.

ABSTRACT

Vehicular networks (VANETs) facilitate the deployment of intelligent applications in urban mobility scenarios, such as real-time traffic data dissemination or resource sharing among VANET components. However, the communication time (referred to as link lifetime or TVE) between nodes is typically short due to the dynamic nature of vehicular mobile scenarios. This limited communication duration has an impact on applications and processes within VANETs, including the decision-making process for task offloading. Furthermore, there is limited integration of entities such as pedestrians in existing research on VANETs, particularly in the context of offloading applications. An architectural model is proposed for computational offloading in VANETs, which involves intelligent prediction of the link lifetime to support the decision-making process. Firstly, a machine learning (ML) model is proposed to predict TVE between nodes in VANETs. The aim is to enhance the efficiency of predictions and improve the decision-making process for computational offloading. Several ML models were trained to evaluate the feasibility of predicting TVE in both Highway and Urban scenarios. Subsequently, an offloading decision algorithm is developed to distribute tasks among resource servers from pedestrian devices. The algorithm's efficiency is evaluated in comparison to the random decision algorithm (FIFO, First in First out), as well as from different perspectives based on the available servers. The results demonstrate that TVE prediction modeling using SVR (Support Vector Regression) is effective, leading to a 50% reduction in task loss rate compared to the traditional approach. The decision algorithm exhibits lower recovery and false negative rates during the offloading process compared to the random approach, with the number of false negatives or local runs being approximately 40% lower.

Keywords: VANET; link lifetime; computational offloading; machine learning.

LISTA DE FIGURAS

Figura 1 – Visão geral das entidades envolvidas no *offloading* em VANET. 20
Figura 2 – Visão geral dos componentes de uma VANET 25
Figura 3 – Diagrama com as principais variações de comunicação em VANET 25
Figura 4 – Pilha de protocolos do WAVE. 27
Figura 5 – Exemplo de uma rede com acesso *mmWave* e servidores de borda e nuvem. . 28
Figura 6 – Exemplo de *Offloading* em VANET 30
Figura 7 – Estrutura do VFC. 32
Figura 8 – Fluxo de Trabalho do Aprendizado Supervisionado. 34
Figura 9 – Um Hiperplano de Separação para pontos de dados bidimensionais. 35
Figura 10 – Pontos de dados bidimensionais que não podem ser separados linearmente. . 36
Figura 11 – Aumento de dimensionalidade provocada pela função *kernel*. 36
Figura 12 – Visão geral da estrutura de uma árvore de decisão. 37
Figura 13 – Visão Geral de uma Random Forest. 38
Figura 14 – kNN Simples. 40
Figura 15 – Um *Multilayer Perceptron* com várias camadas ocultas. 41
Figura 16 – Modelo *Perceptron*, da esquerda para a direita: **a** Etapas do modelo. **b** Modelo simplificado 42
Figura 17 – Fluxo de Trabalho do Esquema de Predição Proposto. 50
Figura 18 – Representação Gráfica dos Cenários. 52
Figura 19 – Tempo de decisão para o *offloading* nos cenários avaliados. 59
Figura 20 – Resultados do *offloading* para o cenário de Rodovia. 62
Figura 21 – Resultados do *offloading* para o cenário Urbano. 62
Figura 22 – Processo de descoberta de veículos. 67
Figura 23 – Topologia de comunicação P2X. 68
Figura 24 – Cenário utilizado neste trabalho — região de Manhattan, Nova York, EUA. . 72
Figura 25 – Eficiência do *offloading* com cada algoritmo. 76
Figura 26 – Tempo médio de *offloading* com cada Algoritmo. 77
Figura 27 – Visão Geral da arquitetura proposta. 84
Figura 28 – Visão geral do processo de *Offloading* em VFC, utilizando o LECV com a predição por contexto. 85
Figura 29 – Taxa de *offloading* e execução local. 91

Figura 30 – Taxa de Recuperação das Tarefas. 92
Figura 31 – Tempo médio de atualização do modelo. 93

LISTA DE TABELAS

Tabela 1 – Exemplos de funções de ativação utilizadas em um *perceptron*. 42
Tabela 2 – Comparação com os Trabalhos Relacionados. 47
Tabela 3 – Hiperparâmetros dos modelos de ML. 56
Tabela 4 – Sumário dos resultados dos experimentos. 58
Tabela 5 – Fatores e níveis considerados nos experimentos. 60
Tabela 6 – Principais Configurações da Simulação. 72
Tabela 7 – Combinação dos Fatores de Densidade e Tamanho da Tarefa. 73
Tabela 8 – Valores de *p-value* para testes de *Wilcoxon* pareados de FIFO e LEV. 78
Tabela 9 – Valores de *p-value* para testes de *Wilcoxon* pareados de FIFO, LE e LEV. . . 78
Tabela 10 – Comparação com os Trabalhos Relacionados. 83
Tabela 11 – Resultados da Avaliação Experimental. 89
Tabela 12 – Valores de *p* para testes de *Wilcoxon* pareados. 93

LISTA DE ALGORITMOS

Algoritmo 1 – Processamento do Tempo de Comunicação. 55

Algoritmo 2 – Algoritmo de Decisão 69

Algoritmo 3 – balancerTask 70

LISTA DE ABREVIATURAS E SIGLAS

AODV	*Vetor de Distância sob Demanda Ad hoc*
BDMA	*Acesso Múltiplo por Divisão de Feixe*
BS	*Estação-Base*
BSS	*Conjunto de Serviço Básico*
CCH	*Canal de Controle*
DSRC	*Comunicação Dedicada de Curto-alcance*
FBMC	*Banco de Filtros Multiportadora*
FIFO	*First In, First Out*
GB	*Aumento de Gradiente*
GBDT	*Árvores de Decisão Impulsionadas por Gradiente*
GCF	*Guloso por CPU Livre*
GCFML	*Guloso por CPU Livre com Aprendizado de Máquina*
gNB	*gNodeB*
HVC	*Nuvem Veicular Híbrida*
IEEE	*Instituto de Engenheiros Eletricistas e Eletrônicos*
IoT	*Internet das Coisas*
ITS	*Sistema de Transporte Inteligente*
kNN	*k-Vizinhos mais próximos*
LE	*Local e Borda*
LEC	*Local, Borda e Nuvem*
LECV	*Local, Borda, Nuvem e Veículo*
LEV	*Local, Borda e Veículo*
LLT	*Link Lifetime*
MAE	*Erro Médio Absoluto*
MANET	*Rede Móvel Ad hoc*
MAPE	*Erro Percentual Médio Absoluto*
MCC	*Computação Móvel na Nuvem*
MEC	*Computação Móvel na Borda*
MLP	*Perceptron Multicamada*
mmWave	*Ondas Milimétricas para Acesso sem Fio em Ambiente Veicular*
MPBRP	*Predição de Mobilidade baseada em Protocolo de Roteamento*

MRE	*Erro Médio Relativo*
ns-3	*Simulador de Rede 3*
OBU	*Unidade On-boarding*
P2I	*Pedestre-para-infraestrutura*
P2V	*Pedestre-para-veículo*
P2X	*Pedestre-para-tudo*
QoE	*Qualidade de Experiência*
QoS	*Qualidade de Serviço*
RA	*Aceleração Relativa*
RF	*Floresta Aleatória*
RMSE	*Raiz do Erro Quadrado Médio*
RS	*Velocidade Relativa*
RSU	*Unidade de Borda de Estrada*
SDN	*Rede Definida por Software*
SLS	*Semelhança de Localidade Espacial*
SUMO	*Simulador de Mobilidade Urbano*
SVM	*Máquina de Vetor de Suporte*
SVR	*Vetor de Suporte a Regressão*
TFO	*Tempo Final de Offloading*
TVE	*Tempo de Vida do Enlace*
UE	*Equipamento de Usuário*
V2I	*Veículo-para-infraestrutura*
V2N	*Veículo-para-rede*
V2P	*Veículo-para-pedestre*
V2V	*Veículo-para-veículo*
V2X	*Veículo-para-tudo*
VANET	*Rede Veicular Ad hoc*
VCC	*Computação de Nuvem Veicular*
VEC	*Computação de Borda Veicular*
VRU	*Usuário Vulnerável de Estrada*
WAVE	*Acesso sem Fio em Ambiente Veicular*
WSA	*Anúncio de serviços do WAVE*

WSMP *Protocolo de mensagens curtas do WAVE*

XGBoost *Aumento de Gradiente Extremo*

LISTA DE SÍMBOLOS

Vx	Velocidade no Ponto x
Vy	Velocidade no Ponto y
Px	Posição no Ponto x
Py	Posição no Ponto y
D	Distância
Θ	Ângulo
pa	Pseudo-ângulo
Sv	Quadrado das velocidades
r	Alcance
Tc	Tempo de Comunicação
Yp	Tempo Predito
UPL	Upload
DWL	Download
TT	Tamanho de Tarefa
DR	Taxa de Transmissão de dados
VPM	Velocidade de Propagação do Meio
DLF	Atraso
$Proc$	Tempo de Processamento
NT	Número de Tarefas
C	CPU necessária para Processar uma Tarefa
RA	Percentual de CPU total disponível
p	Valor-p

SUMÁRIO

1 INTRODUÇÃO 19
1.1 Motivação 19
1.2 Objetivos 21
1.3 Fora de Escopo 22
1.4 Organização 22
2 FUNDAMENTAÇÃO TEÓRICA 24
2.1 Redes Veiculares Ad hoc (VANET) 24
2.1.1 WAVE 26
2.1.2 Comunicação 5G 27
2.2 *Offloading* Computacional 29
2.2.1 Offloading Computacional em VANET 30
2.2.2 Vehicular Fog Computing 31
2.3 Aprendizado de Máquina 33
2.3.1 Aprendizado Supervisionado baseado em Regressão 34
2.3.1.1 Vetor de Suporte a Regressão *35*
2.3.1.2 Floresta Aleatória *37*
2.3.1.3 K-Vizinhos mais Próximos *39*
2.3.1.4 MultiLayer Perceptron *40*
2.3.1.5 XGBoost *43*
2.3.1.6 Adaboost *44*
3 PREDIÇÃO DO TVE COM APRENDIZADO DE MÁQUINA 45
3.1 Contextualização 45
3.2 Trabalhos Relacionados 46
3.3 Metodologia 49
3.3.1 Problemática 49
3.3.2 Fluxo de Etapas no Processo de Predição do TVE 50
3.3.3 Geração de Cenários 51
3.3.4 Coleta de Dados e Definição de Atributos 52
3.3.5 Processo de Limpeza de Dados 55
3.3.6 Seleção dos Algoritmos de ML 56

3.3.7 *Avaliação das Abordagens de ML* 56
3.4 **Resultados dos Experimentos** 58
3.4.1 *Tempo de Decisão do Offloading Computacional* 58
3.4.2 *Desempenho Geral do Offloading Computacional* 60
3.5 **Considerações Finais** 63
4 **PROCESSO DE DECISÃO DE *OFFLOADING* EM *VEHICULAR FOG COMPUTING* COM PEDESTRE** 64
4.1 **Contextualização** 64
4.2 **Trabalhos Relacionados** 65
4.3 **Etapas para o *Offloading* de Tarefas do Pedestre em VFC** 66
4.3.1 *Processo de Descoberta de Veículo* 67
4.3.2 *Algoritmo de Decisão de Offloading* 68
4.3.3 *Arquitetura de Comunicação* 71
4.4 **Experimentos** 72
4.4.1 *Eficiência do offloading* 75
4.4.2 *Tempo de Offloading* 77
4.5 **Considerações Finais** 78
5 **ARQUITETURA** 80
5.1 **Contextualização** 80
5.2 **Trabalhos Relacionados** 81
5.3 **Arquitetura do Sistema** 82
5.3.1 *Descoberta de Veículos para VFC e Escolha de Modelo* 85
5.3.2 *Escolha de Veículo Fog para VFC e Offloading de Tarefas* 86
5.4 **Avaliação Experimental** 87
5.4.1 *Geração e Treinamento dos Modelos* 87
5.4.2 *Resultados da Avaliação dos Algoritmos de ML* 88
5.4.3 *Resultados do Offloading* 89
5.5 **Considerações Finais** 94
6 **CONCLUSÃO** 95
6.1 **Resultados Alcançados** 96
6.2 **Trabalhos Futuros** 97
REFERÊNCIAS 98

APÊNDICE A – UTILIZAÇÃO DO NS-3 E SUMO 108
APÊNDICE B – CÓDIGO DO PSEUDO-ÂNGULO 110
ANEXO A – TRACE SUMO 111

1 INTRODUÇÃO

Há uma demanda para a criação de sistemas que tornem o trânsito mais eficiente e seguro, fornecendo aos usuários diversas informações sobre as condições do trânsito e outros serviços. Essa área de pesquisa é chamada de *Sistema de Transporte Inteligente* (ITS)(ZHANG *et al.*, 2011)(VIZZARI *et al.*, 2023)(RANI; SHARMA, 2023). Uma *Rede Veicular Ad hoc* (VANET) desempenha um papel essencial no ITS, oferecendo diversos serviços e aplicativos que possibilitam sua implementação, aproveitando o crescente número de veículos capazes de se conectar e compartilhar dados com outras redes (SALEM *et al.*, 2022). As VANETs fornecem informações de tráfego em tempo real, que podem ser usadas, por exemplo, para gerenciamento de fluxo de tráfego e controle de congestionamento de veículos (RAUT; DEVANE, 2017).

Graças ao constante desenvolvimento do campo das VANETs, houve o surgimento de novas aplicações complexas, como direção autônoma, realidade aumentada, *streaming* de vídeo em veículos, navegação automática, monitoramento em tempo real, entre outras (ALABBASI; AGGARWAL, 2020). Tais aplicações representam um desafio à limitada quantidade de recursos computacionais de um veículo, uma vez que várias delas demandam poder computacional e são sensíveis à latência, o que tornam o cenário desafiador (HE *et al.*, 2021). Uma solução para melhorar o processamento das aplicações em VANETs é utilizar técnicas de *offloading* computacional para aproveitar os recursos ociosos de veículos próximos e transferir subpartes de aplicações, denominadas tarefas, ou aplicativos inteiros para serem executados nesses dispositivos (LI *et al.*, 2018).

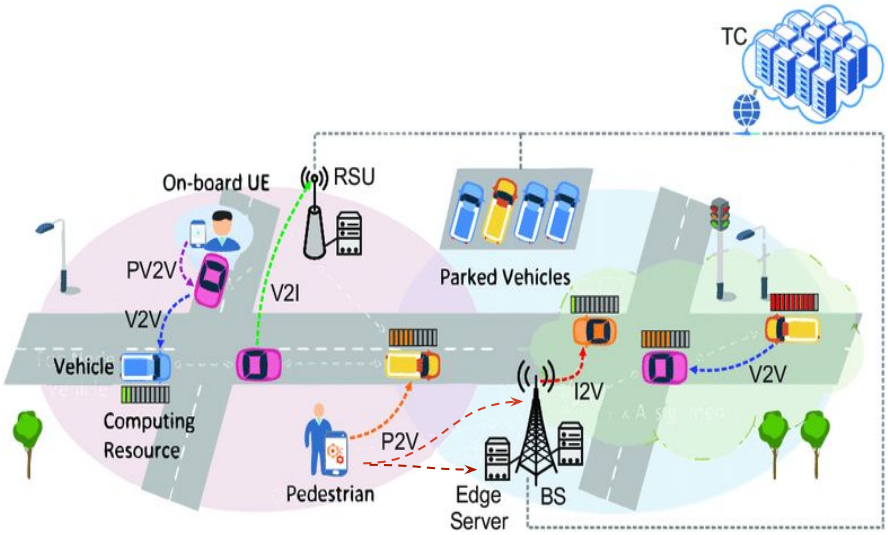
As técnicas de *offloading* computacional transferem tarefas ou aplicativos inteiros para outros dispositivos para melhor usar os recursos disponíveis, podendo oferecer um melhor desempenho para aplicativos executados em VANETs. Nesse contexto, dois fatores essenciais no *offloading* computacional, *atraso* e *custo*, dificilmente atingem valores desejáveis devido às frequentes falhas de comunicação causadas pela alta mobilidade dos veículos (LEE; ATKISON, 2021).

1.1 Motivação

Devido à alta mobilidade dos nós em uma rede veicular, é comum que a comunicação entre os nós seja curta. O baixo tempo de comunicação ou a quebra de comunicação reflete em problemas para as VANETs, como retransmissões, perdas de dados, o que consequentemente

ocasiona uma degradação nos serviços ofertados. As aplicações de *offloading* computacional são também afetadas por esses desafios de comunicação que as VANETs enfrentam. Uma das principais soluções para lidar com a questão do curto tempo de contato entre os nós de uma VANET é realizar predições mais realísticas, ou seja, com uma menor diferença entre um valor de tempo predito por uma aplicação de *offloading* e um tempo que seria o real. Outra solução comumente proposta é a inserção de mais elementos no processo de *offloading*, o que diminuiria a sobrecarga na rede e atrasos no processamento. Porém, a inserção de tais elementos em uma rede veicular, como pedestres, ainda não é alvo de muitos trabalhos da literatura, já que envolve uma complexidade maior por se tratar de um ator que apresenta peculiares características e desafios (NGUYEN; DRESSLER, 2020). A Figura 1 ilustra a visão geral de um cenário com o pedestre usando recursos provenientes de diferentes servidores, e também de diferentes tipos de comunicação em VANET (*Veículo-para-veículo* (V2V), *Veículo-para-infraestrutura* (V2I), *Pedestre-para-veículo* (P2V), *Pedestre-para-infraestrutura* (P2I)).

Figura 1 – Visão geral das entidades envolvidas no *offloading* em VANET.



Fonte: Souza *et al.* (2020)

A tecnologia *Pedestre-para-tudo* (P2X), envolve a comunicação de pedestre para qualquer entidade, incluindo outros pedestres (P2P), infraestrutura (servidor de borda e nuvem ou RSU)(P2I) e veículos (P2V) (DOMINGUEZ; SANGUINO, 2019). Logo, a comunicação P2X é o que permite o *offloading* de tarefas para essas entidades. A possibilidade do *offloading*

atender o pedestre, possibilitará uma melhoria nas suas aplicações, uma vez que os dispositivos móveis dos pedestres, possuem conhecidas limitações de recursos e baterias.

O *offloading* pode oferecer processamento extra para veículos e pedestres, o que possibilita que aplicações mais complexas e sensíveis à latência possam ser melhor utilizadas (ALDMOUR *et al.*, 2021). Para isso, a VFC(do inglês, *Vehicular Fog Computing*) surge como uma arquitetura que utiliza os recursos computacionais de veículos próximos ao cliente de *offloading*, chamados de nós *fog*, assim como dispositivos de borda próximos aos usuários (WAHEED *et al.*, 2022). Porém, o *offloading* em ambientes com veículos e pedestres como servidores *Fog* ainda apresenta desafios, principalmente em relação à alocação de tarefas devido aos servidores geograficamente dispersos de uma VFC (HAZARIKA *et al.*, 2022).

Enquanto algumas informações podem ser obtidas diretamente ou por meio de cálculos simples de dados de sensores de *smartphones*, outras (por exemplo, dados de movimento) exigem manipulação de dados mais extensa, como pré-processamento, extração de recursos e treinamento de modelos de aprendizado de máquina (RONAO; CHO, 2016; DUA *et al.*, 2021). Essa abundância de dados continua sendo gerada devido à dinamicidade da VANET. Técnicas de ML podem ser integradas com VANET para utilizar esses dados para várias aplicações de VANET de forma eficiente. O uso de modelos treinados por ML vem ganhando notoriedade em pesquisas mais recentes. Com o uso de tais modelos para predição e classificação de diversas informações, em diversos campos de atuação em VANET, como segurança, rotas, *offloading*, entre outros (BANGUI *et al.*, 2022)(KHATRI *et al.*, 2021).

1.2 Objetivos

Este trabalho busca avançar as pesquisas do nosso grupo sobre *offloading* computacional em VANETs, tendo como **objetivo principal melhorar o desempenho do processo de tomada de decisão para o *offloading* computacional em cenários VFC, através do uso de modelos de ML adaptativos para predição do tempo de vida do enlace**.

O trabalho propõe uma integração entre esses dois tópicos: a predição do tempo de vida e inserção de pedestres em uma rede veicular, a fim de utilizar os elementos da mesma como provedores de recursos para o *offloading* computacional das aplicações. Para atingir o objetivo principal, os seguintes objetivos específicos (OE) foram definidos:

OE1 - Desenvolver um esquema para predição do tempo de vida do enlace entre nós de VANETs;

OE2 - Desenvolver algoritmo de decisão ao nível de simulação para *offloading* computacional

em VFC envolvendo pedestres e veículos; e

OE3 - Projetar e implementar uma arquitetura conceitual para utilizar os algoritmos desenvolvidos no OE2 em um cenário simulado de VFC, utilizando modelos de ML adaptativos conforme a região.

1.3 Fora de Escopo

Alguns assuntos estão fora do escopo deste trabalho, a fim de deixar mais enxuto e objetivo o trabalho. Alguns deles, comuns na literatura, são mencionados a seguir:

- Predição de rota: é um tópico bastante pesquisado, quando se fala em uso de ML em redes veiculares. Porém, a predição de rotas ainda é pouco utilizada em conjunto com a predição do TVE, apesar de estudos produzidos recentemente, tentarem convergir para um uso simultâneo.
- Análise de Energia e Custo da Nuvem no VFC: A computação em Nuvem, assim como o uso de dispositivos móveis, requerem quase sempre uma avaliação do custo de processamento e consumo de energia respectivamente. Porém, o foco principal do trabalho é apresentar uma arquitetura, juntamente com uma solução de predição do TVE, a fim de diminuir erros de decisão no processo de *offloading* de tarefas em um cenário de VFC. Assim sendo, considerar custo e consumo de energia, tanto na arquitetura, quanto na avaliação, traria uma complexidade maior ao trabalho.
- Segurança: Tecnologias referentes a esse tópico, como criptografia, integridade, entre outros, traria sobrecarga no processo de *offloading* o que poderia impactar os resultados da avaliação de desempenho.

1.4 Organização

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 expõe o referencial teórico dos principais assuntos deste trabalho. O Capítulo 3 aborda com detalhes a parte referente ao OE1, que é a predição do tempo de vida do enlace, mostrando a primeira parte da formalização do problema e resultados alcançados, assim como os trabalhos relacionados. O Capítulo 4 apresenta a parte referente ao OE2, que introduz a implementação de um algoritmo de *offloading* em cenário de uma VFC com pedestre como cliente. Ainda são debatidos os resultados alcançados pela abordagem, assim como trabalhos relacionados a tal contexto. O

Capítulo 5 apresenta a arquitetura projetada (OE3) e os experimentos e resultados alcançados através do uso da mesma. Por fim, o Capítulo 6 apresenta as principais conclusões de todo o trabalho, assim como os desafios e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os principais conceitos e assuntos abordados no presente trabalho.

2.1 Redes Veiculares Ad hoc (VANET)

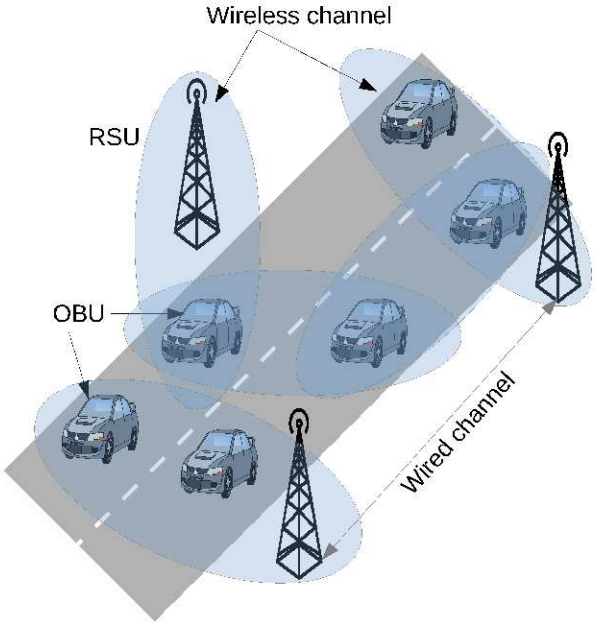
VANET é uma classe distinta da *Rede Móvel Ad hoc* (MANET), em que os nós ou veículos movem-se em rotas pré-definidas. Esses veículos estabelecem comunicação entre si, visando o compartilhamento de dados, especialmente informações de fluxo do trânsito, essencial no processo de habilitação do trânsito seguro e de demais aplicações para o ITS. As redes veiculares possuem dois componentes essenciais para sua construção: a *Unidade de Borda de Estrada* (RSU) e a *Unidade On-boarding* (OBU). Esses dois componentes exercem o papel de autoridades de cadastro, gerenciamento e coleta de dados em uma VANET (HASROUNY *et al.*, 2017; RAW *et al.*, 2013; ANWER; GUY, 2014; MISHRA *et al.*, 2016; GRASSI *et al.*, 2014).

As redes veiculares necessitam trocar mensagens e informações entre seus componentes para habilitar as aplicações conectadas aos veículos. Essas mensagens são trocadas por meio da OBU no veículo e podem ser desde alerta de colisão e acidentes até aplicações de lazer para os usuários do veículo. Através da RSU um veículo pode ter acesso a outras redes remotas e a internet, por exemplo. Por meio da comunicação *wireless*, um veículo pode tornar-se cliente ou servidor de aplicações VANET, em comunicação com outro veículo ou infraestrutura, tendo um raio de alcance de até 1000 metros (HAMDI *et al.*, 2021; SAINI *et al.*, 2015). A Figura 2 demonstra como os componentes RSU e OBU se comportam em uma rede veicular. A comunicação entre os veículos, por meio de sua OBU, com outros veículos ou RSU é realizado via *wireless* formando uma rede ad-hoc, com troca de pacotes.

A comunicação *wireless* entre as entidades de uma rede veicular é realizada por meio de uma tecnologia, chamada *Acesso sem Fio em Ambiente Veicular* (WAVE) (SHEIKH; LIANG, 2019). A WAVE aborda todos os protocolos e padrões utilizados nas camadas física e MAC (PHY/MAC) da comunicação veicular (SAINI *et al.*, 2015). Essa comunicação veicular acontece em um espectro dedicado para esse fim, chamado *Comunicação Dedicada de Curto-alcance* (DSRC) baseado na tecnologia IEEE 802.11p(KENNEY, 2011; AMADEO *et al.*, 2012).

Em relação aos cenários que ocorrem o processo de comunicação, as VANETs possuem algumas variações, entre elas as que se destacam são o V2V, V2I e um cenário

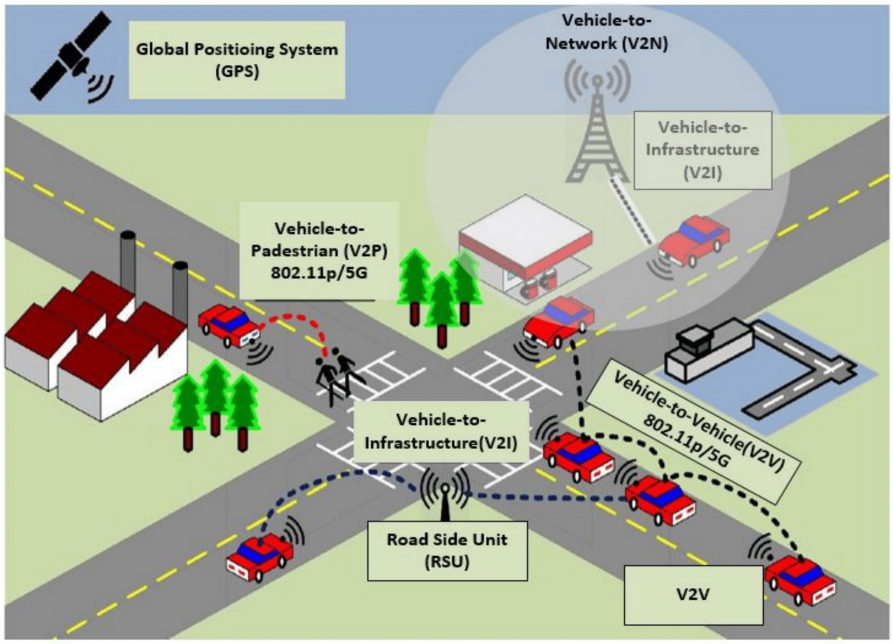
Figura 2 – Visão geral dos componentes de uma VANET



Fonte: Saini *et al.* (2015)

híbrido entre V2V e V2I. Porém, ultimamente, algumas variações vêm surgindo graças a maior interoperabilidade que as VANETs vêm alcançando, assim como o maior desenvolvimento da *Internet das Coisas* (IoT), entre elas destacam-se o *Veículo-para-tudo* (V2X) e *Veículo-para-pedestre* (V2P) (ZEKRI; JIA, 2018; ZHOU *et al.*, 2020). A Figura 3 demonstra uma arquitetura geral de VANET com todos as formas de comunicação.

Figura 3 – Diagrama com as principais variações de comunicação em VANET



Fonte: Eze *et al.* (2014)

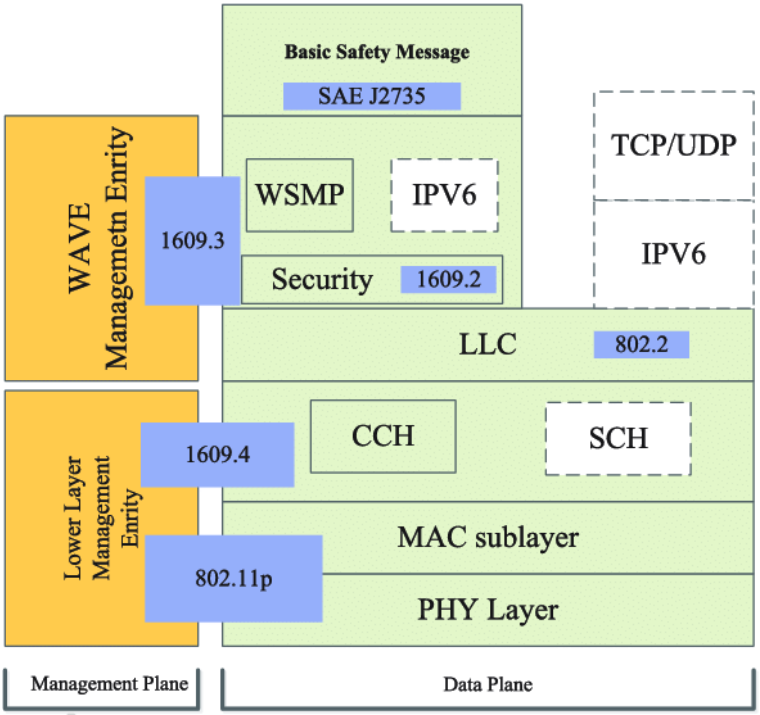
Na comunicação V2V, um veículo se comunica diretamente com outro veículo, onde normalmente essa comunicação visa ser mais rápida e limitada a veículos vizinhos, a fim de trocarem informações de tráfego ou compartilhar informações de sensores. Na comunicação V2I, um RSU ou qualquer servidor conecta-se com algum veículo, nesse cenário o RSU possui um alcance maior, o que permite que as comunicações durem um pouco mais de tempo. Como demonstrado na Figura 3, o RSU pode prover um acesso a mais redes, como a internet, por exemplo, a um veículo próximo (ARIF *et al.*, 2019). Essa comunicação entre veículos, com redes externas é realizada por meio da tecnologia *Veículo-para-rede* (V2N), que permite essa comunicação por meio de redes celulares. Na comunicação V2X, as informações são trocadas entre veículos vizinhos, a infraestrutura de beira de estrada ou pedestres, trazendo assim as vantagens e desafios de cada abordagem. A comunicação no ambiente veicular utiliza, em geral, duas tecnologias de acesso: o padrão IEEE 802.11p e as tecnologias de acesso celular (3G, LTE, 5G). O padrão IEEE 802.11p foi introduzido pelo *Instituto de Engenheiros Eletricistas e Eletrônicos* (IEEE) em 2010, trata-se de uma evolução do padrão IEEE 802.11a, otimizado para rede veicular (TAHIR; KATZ, 2022).

2.1.1 WAVE

IEEE 802.11p/WAVE é uma família emergente de padrões destinados a dar suporte ao acesso sem fio em VANETs (CAMPOLO *et al.*, 2010). A figura 4 demonstra todos os padrões que o WAVE engloba.

A pilha de protocolos WAVE inclui IEEE 1609.1, 1609.2, 1609.3, 1609.4 e 802.11p. O IEEE 802.11p define um mecanismo de comunicação especial, permitindo a operação fora do contexto de um *Conjunto de Serviço Básico* (BSS), que evita latência de gerenciamento adicional ao proibir serviços de associação, autenticação e confidencialidade de dados. O padrão IEEE 802.11p possui um conjunto de especificações úteis para possibilitar a comunicação no ambiente veicular; isto é, em um ambiente onde há mudanças rápidas. A frequência de operação do WAVE possui a banda de 5,85–5,925 GHz. Dentro desta faixa, o IEEE 1609.4 fornece operação em vários canais, o que permite que um único dispositivo WAVE suporte aplicações seguras e não seguras em diferentes canais. Há um canal reservado para o controle do sistema e mensagens relacionadas à segurança, e leva o nome de *Canal de Controle* (CCH); e até seis canais são usados para a troca de dados não seguros, canais de sinalização e aplicação de entretenimento (SCHs). (BU *et al.*, 2014; ARENA *et al.*, 2020).

Figura 4 – Pilha de protocolos do WAVE.



Fonte: Zheng *et al.* (2015)

O IEEE 1609.3 fornece serviços de dados e define um novo protocolo chamado *Protocolo de mensagens curtas do WAVE* (WSMP) e um pacote de gerenciamento especial chamado *Anúncio de serviços do WAVE* (WSA). O IEEE 1609.2 é projetado principalmente para serviços de segurança e o IEEE 1609.1 descreve uma camada de aplicação. (BU *et al.*, 2014).

O padrão SAE J2735 define conjuntos de mensagens, quadros de dados e elementos de dados usados por aplicativos para trocar dados pelo DSRC/WAVE e outros protocolos de comunicação. Além disso, o SAE J2735 inclui as seguintes categorias de mensagens: geral, segurança, geolocalização, informações do viajante e pagamento eletrônico (ARENA *et al.*, 2020).

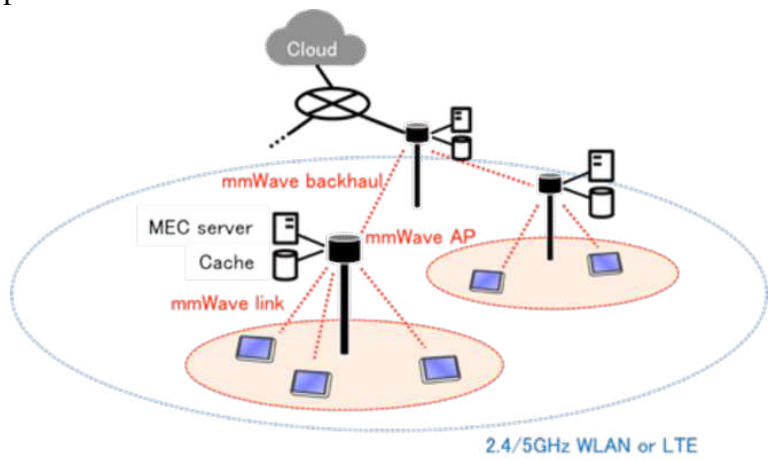
2.1.2 Comunicação 5G

5G é uma rede móvel ou rede de acesso celular de 5ª geração. É um novo padrão *wireless* global sucessora das redes 1G, 2G, 3G e 4G. Um dos pontos mais significativos do 5G, em relação às gerações anteriores, é a largura de banda. O 5G expande o uso de recursos do espectro, significando largura de banda mais ampla, de sub-3 GHz, empregado em 4G, para 100 GHz e muito mais. Além disso, pode funcionar tanto em *Ondas Milimétricas para Acesso sem Fio em Ambiente Veicular* (mmWave) (por exemplo, 24 GHz e superior) quanto em bandas mais baixas (por exemplo, sub-6 GHz), permitindo baixa latência e taxa de transferência multi-Gbps

(ZEBARI *et al.*, 2021).

A comunicação por *mmWave* é uma tecnologia de camada PHY, sendo, portanto, uma parte importante da comunicação 5G. O *mmWave* usa o espectro de radiofrequência aproximado entre 30 e 300 GHz, embora os desafios de pesquisa também se estendam a frequências mais baixas (ou seja, acima de 6 GHz) (MEZZAVILLA *et al.*, 2018). Outra característica do *mmWave* são os links inerentemente direcionais, demonstrados na Figura 5 (*mmWave link*). Com um pequeno comprimento de onda, arranjos de antenas orientáveis eletronicamente podem ser realizados. Então, controlando a fase do sinal transmitido por cada elemento de antena, o conjunto de antenas direciona seu feixe para qualquer direção eletronicamente. Com isso há um ganho no nível de sinal para a localização que a antena aponta, enquanto nas outras direções há uma perda maior de sinal (NIU *et al.*, 2015). A Figura 5 ilustra as antenas e como elas se relacionam com elementos de uma rede de comunicação. Neste caso elas podem ter uma comunicação estabelecida com servidores remotos, por exemplo, borda (*MEC server* e *Cache*) e nuvem (*Cloud*), e assim atuarem como nós intermediários (*mmWave backhaul*) entre esses servidores e os usuários finais, facilitando a orquestração e gerenciamento da rede.

Figura 5 – Exemplo de uma rede com acesso *mmWave* e servidores de borda e nuvem.



Fonte: Sakaguchi *et al.* (2017)

Com um aumento exponencial na demanda dos usuários, o 4G agora será facilmente substituído pelo 5G com uma tecnologia de acesso avançada chamada *Acesso Múltiplo por Divisão de Feixe* (BDMA) e acesso múltiplo não e quase ortogonal, também conhecido como *Banco de Filtros Multiportadora* (FBMC). O conceito por trás da técnica BDMA é explicado considerando o caso da estação base se comunicando com as estações móveis. Nesta comunicação, um feixe ortogonal é alocado para cada estação móvel e a técnica BDMA dividirá esse feixe de antena conforme as localizações das estações móveis para dar acessos múltiplos às estações

móveis, aumentando a capacidade do sistema de forma correspondente (WANG *et al.*, 2014).

2.2 *Offloading* Computacional

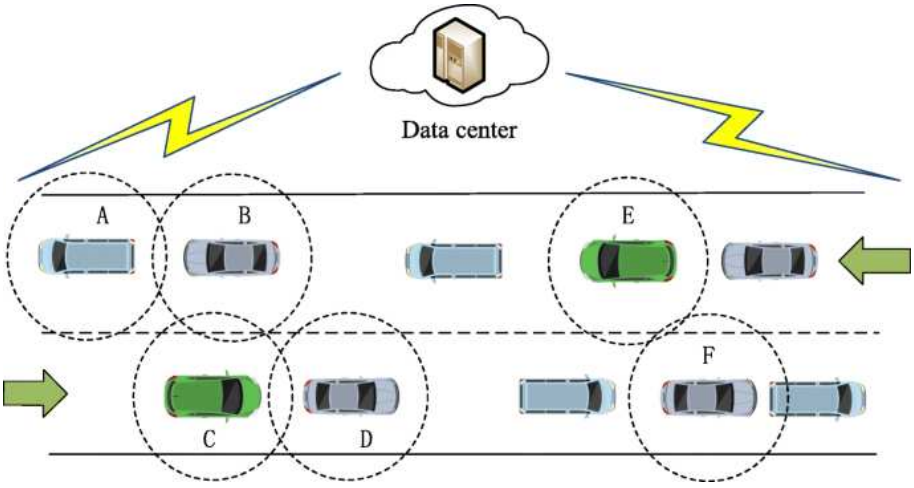
Com o crescente número de dispositivos móveis que surgiram nos últimos anos, como *smartphones* ou dispositivos *wearables*, a computação móvel foi impulsionada. Consequentemente, diversas aplicações surgiram e estão desenvolvendo-se nessa área, assim como a própria computação móvel está estendendo suas fronteiras de possibilidade para outras áreas, como VANET, por exemplo. Com o rápido crescimento das aplicações, houve uma maior demanda de recursos computacionais nos dispositivos móveis, já que as aplicações tornaram-se bastante complexas e computacionalmente exigentes. Uma vez que os dispositivos móveis enfrentam desafios em poder computacional, armazenamento e duração da bateria, foi necessária buscar soluções que lidassem com essas aplicações em dispositivos móveis, destacando-se nesse contexto o *offloading*. *Offloading* trata-se de uma solução que visa aumentar os recursos de sistemas móveis migrando a execução de aplicações ou processos para dispositivos com mais recursos computacionais (ou seja, servidores)(KUMAR *et al.*, 2013; LIN *et al.*, 2019a).

O *offloading* envia tarefas (partes de uma aplicação) ou uma aplicação inteira para ser executada em um servidor remoto. A execução de subpartes de aplicações em servidores remotos, com maior capacidade computacional, é benéfica para os dispositivos móveis, trazendo economia de energia e redução no tempo de execução da aplicação (SHIRAZ *et al.*, 2015). Algumas arquiteturas e esquemas de *offloading* surgiram recentemente, principalmente em relação aos tipos de servidores remotos utilizados. Um desses esquemas trata-se do *Computação Móvel na Nuvem* (MCC), descrita como um modelo de computação composta por serviços móveis e a computação em nuvem via Internet, onde as tarefas são enviadas para serem processadas em um servidor na nuvem. A integração e convergência dessas duas tecnologias em um único modelo contínuo é representada pelo MCC (QI; GANI, 2012; ALAHMAD *et al.*, 2021). O MCC apresenta desafios relacionados à latência e o tempo de resposta entre a nuvem e o dispositivo móvel, devido à distância entre esses últimos. Para lidar com esse desafio, os serviços em nuvem devem ser movidos para mais próximo dos dispositivos móveis, ou seja, para a borda da rede móvel, surgindo assim o paradigma *Computação Móvel na Borda* (MEC) (MACH; BECVAR, 2017).

2.2.1 Offloading Computacional em VANET

O processo de *offloading* em VANETs é baseado nos cenários de comunicação da VANET, sendo que o *offloading* pode ser realizado nas infraestruturas de comunicações de uma VANET, com destaque para as seguintes: V2V, V2I, V2X e V2P. No cenário V2I, as tarefas são enviadas do veículo para uma RSU no caso de uma comunicação 802.11p ou para uma *Estação-Base* (BS) e posteriormente para um servidor remoto, seja nuvem ou borda. A comunicação V2V é estabelecida para compartilhar dados diretamente entre um par de veículos próximos, onde infraestruturas fixas, como RSU ou BS, não são necessárias. Por fim, na comunicação V2X um veículo pode enviar tarefas para infraestruturas fixas e veículos próximos paralelamente (ZHOU *et al.*, 2018). No cenário de comunicação V2P, ocorre a conexão entre veículos em movimento e pedestres próximos à estrada. Com dispositivos móveis, como celulares, por exemplo, os passageiros podem facilmente ingressar em VANETs como nós de beira de estrada(LIU *et al.*, 2010). A Figura 6 ilustra como o *offloading* é realizado em uma rede veicular.

Figura 6 – Exemplo de *Offloading* em VANET



Fonte: Li *et al.* (2019)

O veículo que deseja realizar o *offloading*(verde) realiza uma operação de descoberta de vizinhos, que se tratam de veículos com recursos disponíveis e dispostos a executar tarefas. Uma vez que esses veículos são identificados, eles são selecionados pelo algoritmo de decisão do *offloading* e então o veículo cliente envia as tarefas para serem processadas nos nós vizinhos (veículos circulados na Figura). Caso algum servidor remoto (Nuvem ou servidor de borda, representados pelo nome de *Data Center* na Figura 6) esteja disponível, as tarefas podem ser enviadas para ele, por meio de alguma infraestrutura fixa.

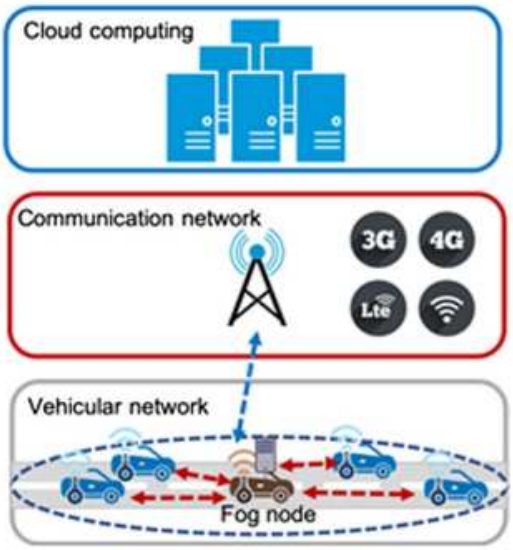
2.2.2 Vehicular Fog Computing

Para entender o conceito de *Vehicular Fog Computing*, é preciso observar o conceito do paradigma *Fog Computing*, uma vez que VFC trata-se de uma extensão desse paradigma. *Fog Computing* é uma arquitetura de computação distribuída geograficamente com uma gama de recursos computacionais que consiste em um ou mais dispositivos heterogêneos conectados na borda da rede(YI *et al.*, 2015). O objetivo é fornecer processamento, armazenamento e comunicação próximos ao cliente, a fim de mitigar a latência e tempo de processamento, que o cliente poderia enfrentar ao enviar sua aplicação para ser processada na nuvem. Especificamente, na arquitetura de *Fog Computing*, veículos e dispositivos móveis são usuários finais e, entre esses usuários e a nuvem, existe uma camada que funciona como servidor "*Fog*". Por outro lado, na arquitetura VFC, os próprios veículos e dispositivos móveis também fazem parte da "*fog*". Especificamente, a VFC é uma arquitetura que utiliza uma multiplicidade colaborativa de usuários finais (veículos, pedestres, etc.) para realizar uma quantidade substancial de comunicação e computação (HOU *et al.*, 2016).

VFC estende as capacidades de *Fog Computing*, assim como ela estende o paradigma tradicional da computação em Nuvem. Diferente da *Computação de Nuvem Veicular* (VCC), a estrutura VFC processa as tarefas dos veículos localmente o máximo possível, em vez de enviar solicitações de computação para servidores em nuvem. Como um componente significativo do ITS, a VFC pode oferecer suporte a serviços veiculares complexos por meio do agendamento colaborativo de veículos. A *Qualidade de Serviço* (QoS) e a *Qualidade de Experiência* (QoE) podem alcançar grandes melhorias com base na melhor utilização dos recursos computacionais e de comunicação de cada veículo (WU *et al.*, 2020).

Na Figura 7, a estrutura do paradigma VFC é detalhada. Há uma estação base que se encarrega da coordenação de recursos de comunicação intra-célula, alocação de recursos de computação e atribuição de tarefas. Durante o horário de pico, quando a estação base está sobrecarregada pelas demandas computacionais recebidas, um grupo de veículos é empregado para atuar como nós *fog* e aliviar o problema de sobrecarga por meio do *offloading* de tarefas. Qualquer veículo com recursos computacionais ociosos consegue atuar como um nó *fog* compartilhando seus recursos para processamento de tarefas. Na mesma célula também existe *Equipamento de Usuário* (UE), que pode ser uma OBU de outro veículo ou um *smartphone* de pedestre, por exemplo. Cada UE gera uma série de tarefas de computação, cada uma das quais pode ser processada pela estação base ou alocada para um nó *fog* veicular (ZHOU *et al.*, 2019).

Figura 7 – Estrutura do VFC.



Fonte: Zeadally *et al.* (2020)

Conforme demonstrado na Figura 7, a arquitetura VFC pode ser divida em três diferentes camadas: camada de serviço em nuvem; camada de borda, com algum equipamento que assuma uma função de nó *fog* (*cloudlet* ou RSU); e a Camada *Fog*.

Na camada de serviço em Nuvem, os servidores em nuvem fornecem monitoramento ao nível de cidade e controle centralizado a partir de um local remoto. Esses servidores obterão os dados carregados pelos nós *fog*, enquanto realizam análises computacionalmente intensivas para tomar decisões ideais de uma perspectiva holística (por exemplo, decisão no nível da cidade). Na Camada de borda, equipamentos como RSU, *Cloudlet* ou outros servidores de recursos, são implantadas em diferentes áreas de uma cidade. Esses equipamentos podem ser facilmente atualizados para atuar como nós *Fog*. Isso permite a coleta de dados enviados por veículos ou pedestres, o processamento dos dados coletados e o envio de relatório dos dados (processados) aos servidores em nuvem. Esses nós de borda também funcionam como *middleware* na função de um *link* de conexão entre os servidores em nuvem e os veículos em um sistema VFC. Como os conteúdos gerados pelos UEs de veículos ou pedestres são sempre de interesse local, as informações carregadas relacionadas às condições das estradas interessam apenas aos nós dentro ou ao redor de uma determinada região. Finalmente, a Camada *Fog* consiste em veículos/dispositivos na faixa de comunicação sem fio das RSUs ou estações-base de acesso celular. Esses veículos que podem estar em movimento ou estacionados atuam como infraestrutura de recursos computacionais, compartilhando seus recursos ociosos (HUANG *et al.*, 2017; NING *et al.*, 2019).

2.3 Aprendizado de Máquina

Aprendizado de Máquina trata-se de algoritmos computacionais que realizam algum processo de aprendizado em determinado contexto ou ambiente, emulando uma inteligência humana. Um algoritmo de aprendizado de máquina realiza a leitura de dados de entrada para realizar uma tarefa desejada sem ser literalmente programado (ou seja, "codificado") para produzir um resultado específico. Esses algoritmos, então, alteram ou adaptam automaticamente sua arquitetura por meio da repetição (ou seja, experiência), reconhecendo padrões nas informações processadas, para se tornarem cada vez melhores na realização da tarefa desejada (NAQA; MURPHY, 2015). De acordo com Alpaydin (2020) em seu livro, um dos conceitos de aprendizado de máquina trata-se:

> Programar computadores para otimizar um critério de desempenho usando dados de exemplo ou experiência.

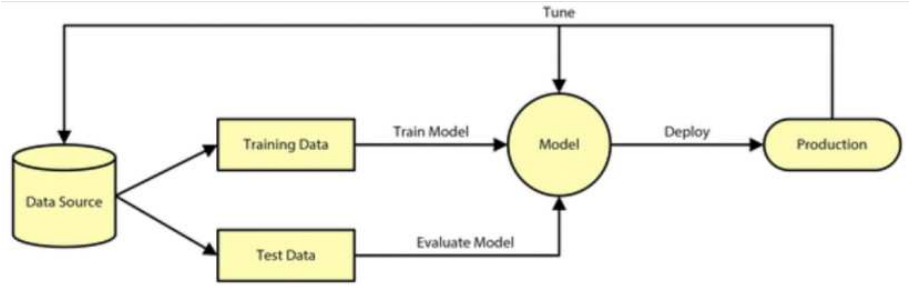
Para se ter uma ideia dos problemas que o aprendizado de máquina consegue resolver, três classes de problemas de aprendizado são definidas: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço (CARLEO *et al.*, 2019). No aprendizado supervisionado o objetivo é prever/classificar, com base em um conjunto de dados iniciais chamadas variáveis independentes, uma saída, chamada de variável dependente. O algoritmo de aprendizado supervisionado, então, gera uma função que mapeia as entradas para as saídas desejadas, partindo da premissa que os dados de entrada (utilizados para treinar o algoritmo) contém a resposta desejada. Com isso esses dados são chamados de dados rotulados com a resposta a ser prevista. Existem dois tipos de aprendizado supervisionado baseados nos tipos de respostas preditas, sendo a Regressão, baseada em respostas contínuas e a Classificação, que gera uma resposta categórica (JIANG *et al.*, 2020). Nem sempre os dados serão rotulados, ou seja, não é possível associar uma saída com uma lista de entradas, esse caso é conhecido como aprendizagem não supervisionada. A aprendizagem não supervisionada é usada para identificar novos padrões e achar uma representação mais realista dos dados. No aprendizado por reforço, o algoritmo guia o processo de aprendizado mediante uma interação com ambiente, onde o ambiente retorna *feedbacks* para o algoritmo, que adapta e reajusta seu processo de aprendizado com base em um sistema de recompensa até que o algoritmo escolha a melhor ação a se tomar (AYODELE, 2010). De todos os tipos de aprendizado apresentados anteriormente, o aprendizado supervisionado baseado em regressão é a solução adotada neste trabalho, uma vez que os dados

coletados em uma rede veicular são rotulados e a partir deles é possível extrair a informação pretendida, o TVE. Por isso, o aprendizado supervisionado baseado em regressão, será mais aprofundado no andamento deste Capítulo.

2.3.1 Aprendizado Supervisionado baseado em Regressão

Os algoritmos de aprendizado de máquina supervisionados precisam de dados externos, a fim de encontrar algum padrão nesses dados. A Figura 8 demonstra todo o fluxo de processos necessários para realizar o processo de treinamento pelos algoritmos de ML, a fim de encontrar padrões nos dados e a partir desses padrões, realizar a predição da informação pretendida. Inicialmente, o conjunto de dados de entrada (*Data Source*) é dividido em conjunto de dados de treinamento e teste. O conjunto de dados de treino tem uma variável de saída que precisa ser prevista ou classificada. Todos os algoritmos aprendem algum tipo de padrão do conjunto de dados de treinamento e os aplicam ao conjunto de dados de teste para previsão ou classificação (MAHESH, 2020). O fluxo de trabalho dos algoritmos de aprendizado de máquina supervisionado é fornecido na figura abaixo.

Figura 8 – Fluxo de Trabalho do Aprendizado Supervisionado.



Fonte: Mahesh (2020)

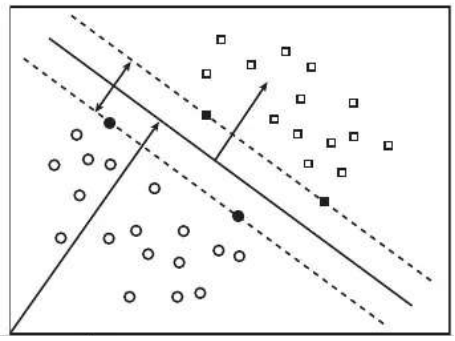
Utilizando o conjunto de dados de treino, o algoritmo realiza o treinamento dos dados e mapeia os atributos de entrada para o atributo de saída (valor que a ser predito) e gera um modelo (*Model*) de ML que captura esses padrões. Posteriormente esse modelo gerado é avaliado, utilizando os dados de entrada disponibilizados no conjunto de dados de teste, que gera a predição do valor de saída, onde esse mesmo valor é comparado com o valor de saída real associado àqueles dados de entrada. A avaliação do modelo é o processo que utiliza algumas métricas que ajudam a analisar o desempenho do modelo. O modelo é então enviado para a aplicação que fará uso dele (*Production*), sendo que o mesmo de tempos em tempos pode ser atualizado e retreinado, em um processo chamado *tune*, a medida que suas métricas de

desempenho mudem. Alguns dos mais utilizados algoritmos para regressão, serão explicadas nos próximos tópicos, onde esses serão utilizados no restante do trabalho e também na etapa final da dissertação.

2.3.1.1 Vetor de Suporte a Regressão

Uma *Máquina de Vetor de Suporte* (SVM) é um algoritmo de computador que aprende, por exemplo, a atribuir rótulos a objetos. Para entender a essência da classificação SVM, basta compreender quatro conceitos básicos: (i) o hiperplano, (ii) hiperplano ou linha de decisão, (iii) vetor de suporte e (iv) a função *kernel*(NOBLE, 2006). O hiperplano trata-se de uma linha de separação entre duas classes de dados, representada na Figura 9 pela linha contínua. Ainda na Figura 9 os círculos e os quadrados representam, pontos de dados em duas classes distintas. O hiperplano de decisão ou linha de decisão, também chamada de linha de contorno, são as duas linhas que contornam o hiperplano a uma determinada distância, sendo usadas para criar uma margem entre os pontos de dados, como demonstradas na 9, por meio das linhas tracejadas. O vetor de suporte são os pontos mais próximos do hiperplano, na Figura 9, eles são ilustrados nos pontos preenchidos em preto. Já o *kernel* é uma função que visa encontrar um hiperplano no espaço dimensional superior. Esse aumento de dimensão é necessário quando não se consegue encontrar um hiperplano de separação em uma determinada dimensão, sendo necessário mover-se em uma dimensão superior. O objetivo do SVM, é então encontrar um melhor hiperplano que possa separar um espaço n-dimensional os pontos de dados em classes distintas, conforme ilustrado na Figura 9 (MAMMONE *et al.*, 2009).

Figura 9 – Um Hiperplano de Separação para pontos de dados bidimensionais.

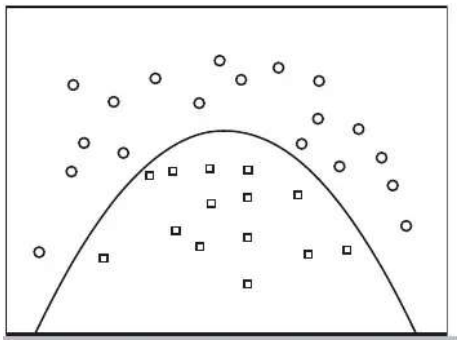


Fonte: Mammone *et al.* (2009)

Em dados reais, porém dificilmente ocorre uma separação linear entre as classes, conforme a Figura 10. Nesse caso os dados estão distribuídos de uma forma que não podem ser

separados por um hiperplano reto ou linear. Para lidar com casos como esses, o algoritmo SVM deve ser modificado adicionando uma 'margem suave' (MARÍN *et al.*, 2022). Essencialmente, isso permite que alguns pontos de dados passem pela outra margem do hiperplano de separação que ele estaria sem afetar o resultado. É como se na figura 10 alguns quadrados estivessem juntos no lado referente aos círculos, porém os mesmos não seriam considerados no treinamento.

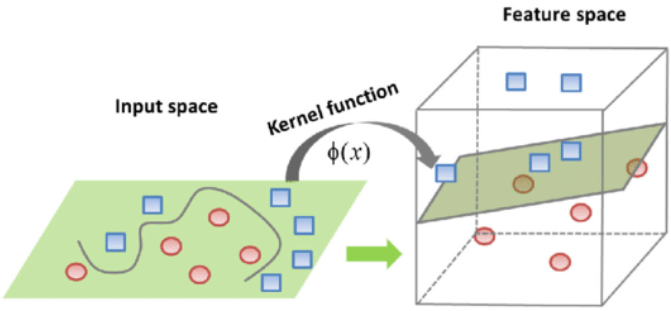
Figura 10 – Pontos de dados bidimensionais que não podem ser separados linearmente.



Fonte: Mammone *et al.* (2009)

Ainda assim, pode haver pontos no hiperplano que não é possível separá-lo entre as classes, mesmo com a adição de um valor de margem suave. Nesse caso a função kernel fornece uma solução para esse problema adicionando outra dimensão aos dados. Em geral, uma função de kernel projeta dados de um espaço de baixa dimensão para um espaço de maior dimensão, conforme a Figura 11.

Figura 11 – Aumento de dimensionalidade provocada pela função *kernel*.



Fonte: Dou *et al.* (2019)

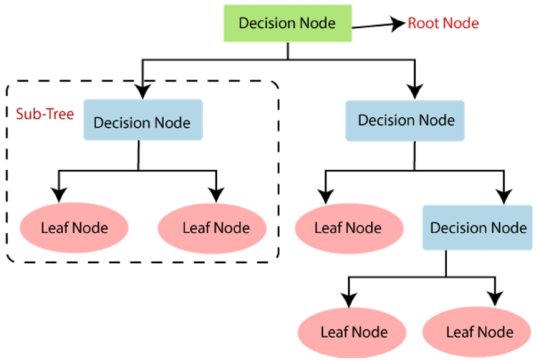
Como demonstrado na Figura 11, a função *kernel* mapeia dados de uma dimensão ou espaço de entrada(2d), para uma dimensão de recursos maior(3d). O espaço de entrada (*input space*) são os dados inseridos inicialmente para o processo de treino, já o espaço de recursos (*feature space*) são os dados mapeados e treinados, que permitem realizar predições. O SVM não permite realizar apenas classificações de classes de dados em diferentes agrupamentos, mas também permite predição de valores contínuos, através do *Vetor de Suporte a Regressão* (SVR).

O SVR foi desenvolvido por Vapnik e colaboradores estendendo seu algoritmo SVM para classificação, a fim de suportar problemas de regressão (DRUCKER *et al.*, 1996). O problema de regressão é uma generalização do problema de classificação, onde o modelo retorna uma saída de valor contínuo, em oposição a uma saída de um conjunto finito. Em outras palavras, um modelo de regressão estima uma função multivariada de valor contínuo. Os conceitos de SVM podem ser generalizados para se tornarem aplicáveis a problemas de regressão. Assim como na classificação, os vetores de suporte a regressão (SVR) são caracterizada pelo uso de *kernels*, solução margem suave entre outras características (ZHANG; O'DONNELL, 2020).

2.3.1.2 Floresta Aleatória

O algoritmo de Floresta Aleatória ou *Random Forest* é uma coleção de árvores de decisão correlacionadas. Uma árvore de decisão é um algoritmo de ML que usa uma estrutura semelhante a um fluxograma hierárquico no processo de predição. No aprendizado de árvore de decisão, a mesma é induzida a partir de um conjunto de dados de treinamento rotulados, representados por uma tupla de valor e um rótulo de classe(SU; ZHANG, 2006). Conforme demonstrado na Figura 12 a estrutura da árvore é composta por um nó raiz, ramificações, nós de decisão e nós folhas, formando uma estrutura hierárquica semelhante a uma árvore.

Figura 12 – Visão geral da estrutura de uma árvore de decisão.



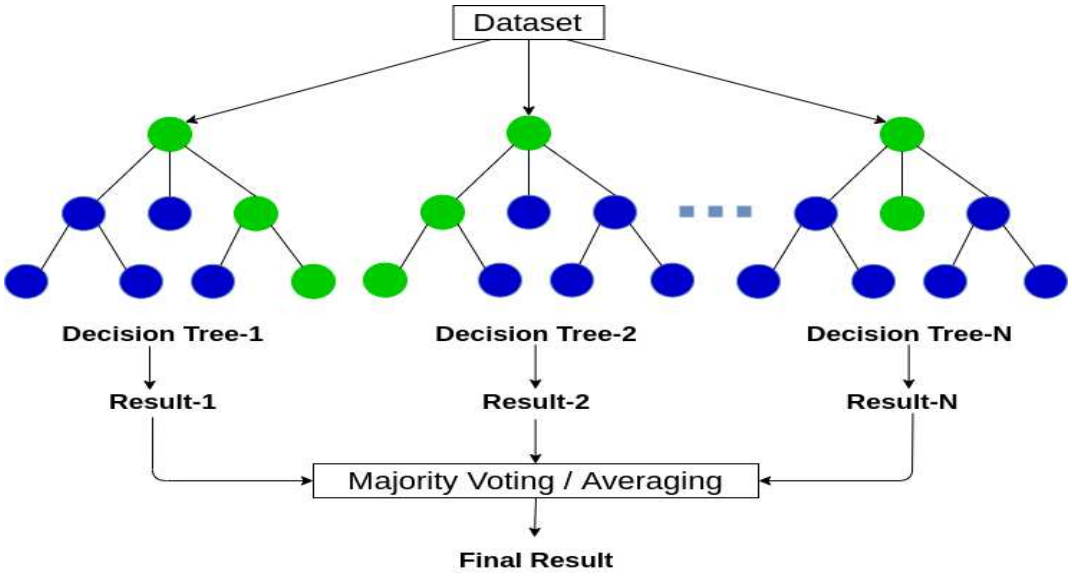
Fonte: Deshpande *et al.* (2021)

O Nó Raiz (verde) é o nó presente no início de uma árvore de decisão, a partir deste nó uma determinada população começa a se dividir de acordo com diversas características. Os Nós de Decisão (azul), são os nós que determinam o fluxo, na qual os dados serão prosseguirão durante a divisão ou classificação dos mesmos, de acordo com seus padrões ou características. O Nó Folha (vermelhos) é o nó onde a divisão de uma parcela da população não é possível. Ao usar uma árvore de decisão para prever um valor, começando no Nó Raiz, um teste no nó

atual (Nó de decisão) é executado e o fluxo correspondente ao resultado do teste é seguido, até que um Nó Folha seja alcançado, para o qual o valor predito correspondente é retornado. As árvores de decisão podem prever tanto valores rotulados (classificação), quanto valores contínuos (regressão) (JOHANSSON *et al.*, 2014).

Floresta Aleatória (RF) é um algoritmo de aprendizado supervisionado, que consiste em uma coleção de classificadores estruturados em árvore. O algoritmo cria de forma aleatória várias Árvores de Decisão e combina o resultado de todas elas para chegar no em um valor predito(GRÖMPING, 2009). No modelo RF de Breiman (BREIMAN, 2001), cada árvore é implementada utilizando um conjunto de dados de amostra para treinamento e uma variável aleatória correspondente à k-enésima árvore sendo denotada como Θ_k, resultando em um classificador $h(x, \Theta_k)$ onde x é o dado de entrada. Após k vezes executados, é obtida a sequência de classificadores $h1(x), h2(x), ...h_k(x)$, e esta sequência é utilizada para constituir um sistema de modelo de classificação, o resultado deste sistema é dado por maioria ordinária de votos (LIU *et al.*, 2012). Para uma dada variável de entrada, cada árvore tem direito a voto para selecionar o melhor resultado de classificação. Todo o processo de classificação de uma RF é resumida na Figura 13.

Figura 13 – Visão Geral de uma Random Forest.



Fonte: Abdulkareem *et al.* (2021)

A partir de um conjunto de dados de entrada (*dataset*) uma quantidade N de Árvores de decisão, são implementadas, onde cada Árvore recebe uma amostra dos dados selecionada aleatoriamente pelo algoritmo de RF. Para começar o processo de predição é preciso definir o

primeiro nó da árvore (Nó Raiz), que será a primeira condição verificada. Uma variável aleatória é escolhida para compor o Nó Raiz, essa variável aleatória, trata-se de um atributo de entrada. No RF, o algoritmo escolhe aleatoriamente duas ou mais variáveis, e então realiza cálculos com base nas amostras selecionadas, para definir qual dessas variáveis será utilizada no primeiro nó. Para escolha da variável do próximo nó, serão novamente escolhidas duas (ou mais) variáveis, excluindo as já selecionadas anteriormente, e o processo de escolha se repetirá. Desta forma a árvore será construída até o último nó e esse processo se repete na construção da próxima Árvore, porém diferente da anterior, pois tanto na seleção das amostras, quanto na seleção das variáveis, o processo acontece de maneira aleatória e isso segue até a Árvore N. Cada árvore criada irá apresentar o seu resultado, sendo que em problemas de regressão será realizada a média (*Averaging*) dos valores previstos e em problemas de classificação o resultado que mais vezes foi apresentado será o escolhido (*Majority Voting*).

2.3.1.3 K-Vizinhos mais Próximos

O *k-Vizinhos mais próximos* (kNN) é um algoritmo de aprendizado semi-supervisionado que assume que dados semelhantes estão próximos um do outro. Utilizando um conceito de similaridade (também chamada de distância ou proximidade), o algoritmo calcula a distância entre os pontos em um gráfico. Em resumo, o kNN tenta classificar cada amostra de um conjunto de dados com base na distância dessa amostra em relação aos vizinhos mais próximos. Se os vizinhos mais próximos são de uma classe (ou em sua maioria), a amostra em questão será classificada nesta categoria.

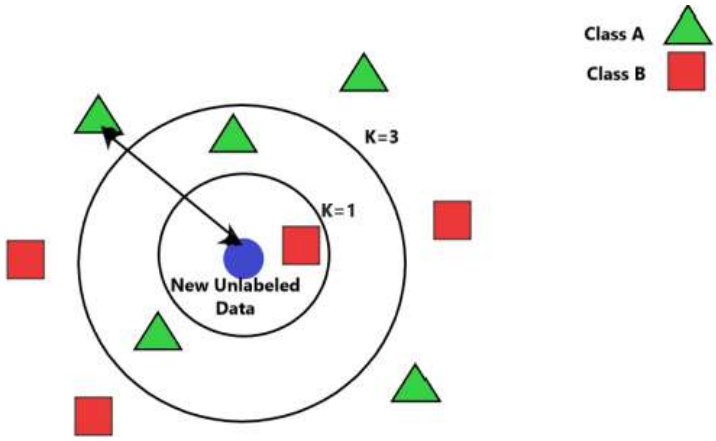
O kNN requer dados de treinamento e um valor k predefinido para encontrar os k dados mais próximos com base no cálculo da distância. Se k dados tiverem classes diferentes, o algoritmo prevê que a classe dos dados desconhecidos seja a mesma que a classe majoritária (CHOMBOON *et al.*, 2015).

A Figura 14, demonstra como o kNN funciona, onde o kNN classifica cada exemplo não rotulado, ou seja, sem classe conhecida pela classe majoritária entre seus k-vizinhos mais próximos no conjunto de treinamento. Seu desempenho, portanto, depende crucialmente da métrica de distância usada para identificar os vizinhos mais próximos, demonstrada na Figura 14 como uma seta. Na ausência de conhecimento prévio, a maioria dos classificadores kNN usa métrica euclidiana simples, com a seguinte equação 2.1 (WEINBERGER; SAUL, 2009).

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.1}$$

Onde “n”, trata-se da quantidade de atributos de entrada ou rótulos e as variáveis x e y são as amostras do conjunto de treino e teste respectivamente. Na qual a variável x representa o valor real do atributo de saída (atributo a ser predito) e a variável y representa o valor predito da variável de saída.

Figura 14 – kNN Simples.



Fonte: Taunk *et al.* (2019)

Quando um novo exemplo não rotulado (*New Unlabeled Data*), destacado em azul na Figura 14, é encontrado no conjunto de dados, o kNN executa duas operações. Primeiro, ele analisa os k pontos mais próximos do novo ponto de dados, ou seja, os k vizinhos mais próximos. Em segundo lugar, usando as classes dos vizinhos, kNN determina em qual classe os novos dados devem ser classificados (TAUNK *et al.*, 2019). Os novos dados não rotulados calculam a distância de cada um de seus vizinhos conforme o valor de K. Em seguida, determina a classe a que pertence, contendo o número máximo de vizinhos mais próximos.

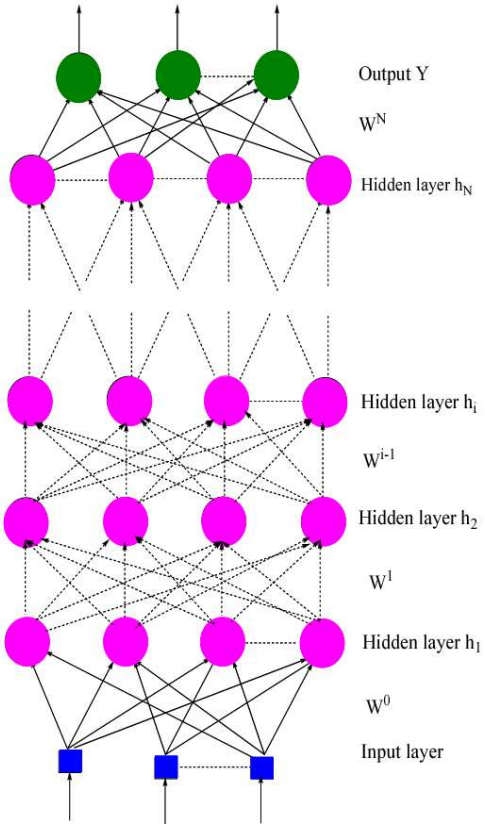
2.3.1.4 MultiLayer Perceptron

As redes neurais, ou mais precisamente as redes neurais artificiais, são um ramo da inteligência artificial (GARDNER; DORLING, 1998). São sistemas compostos por vários nós, ou neurônios, que se interconectam em diversas ramificações. Essas redes podem realizar funções de estimativas para modelos e lidar com funções lineares/não lineares aprendendo com as relações de dados e generalizando para situações desconhecidas (TAUD; MAS, 2018). Os neurônios, recebem e enviam informações para outros neurônios, sendo que cada um pode realizar

processamento de dados com os dados que recebem e posteriormente passam a informação processada por cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada (RAMCHOUN *et al.*, 2016). *MultiLayer Perceptron* forma um tipo de rede neural (GARDNER; DORLING, 1998).

Multilayer Perceptron consiste em camada de entrada (*Input layer*), camada de saída (*Output* Y) e camadas ocultas (*Hidden layer*) entre essas duas camadas, conforme demonstrado na Figura 15. Os nós são conectados por pesos e sinais de saída. Dado $i = 0, 1, ..., n$ onde n é o número de variáveis de entrada, as quantidades w_i são os pesos do neurônio. A saída y (em verde) representa o valor predito. O número dessas camadas ocultas é dependente do problema, sendo que cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação (EĞRIOĞLU *et al.*, 2008).

Figura 15 – Um *Multilayer Perceptron* com várias camadas ocultas.

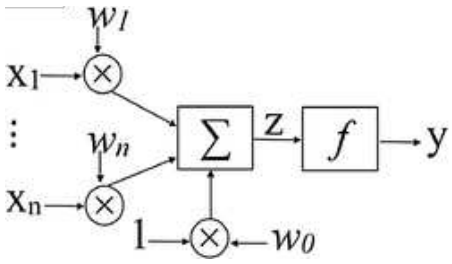


Fonte: Ramchoun *et al.* (2016)

Normalmente, o algoritmo utilizado nessas camadas é o Retro propagação (*o Back-propagation Algorithm*), que opera em duas fases: inicialmente na fase de treinamento, as amostras de dados de treino são fornecidas na camada de entrada para treinar a rede com um conjunto predefinido de classes de dados. Eventualmente, durante a fase de teste, a camada

de entrada recebe os dados de teste para predição dos padrões aplicados. A saída desejada é conhecida pela rede. Portanto, caso a saída calculada não corresponda à saída desejada, a diferença no resultado é retro propagada para a camada de entrada para que os pesos de conexão dos perceptrons sejam ajustados para reduzir o erro. O processo continua até que o erro seja reduzido a uma quantidade insignificante (KISHORE; KAUR, 2012). A Figura 16 mostra a representação simplificada de um *perceptron*.

Figura 16 – Modelo *Perceptron*, da esquerda para a direita: **a** Etapas do modelo. **b** Modelo simplificado



Fonte: Taud e Mas (2018)

A etapa de ponderação envolve a multiplicação de cada valor de recurso de entrada, denotado por x, por seu peso w, resultando em $z\{x_i w_i\}$ e na segunda etapa eles são somados $(x_0 w_0 + x_1 w_1 + ... + x_n w_n)$. A terceira é a etapa de transferência onde uma função de ativação f (também chamada de função de transferência) é aplicada à soma ($\sum$) que produz uma saída y. Por fim, conforme a Figura 16 há o *w0* que se trata do **bias**, o qual é uma constante adicionada ao produto de entradas e pesos (TAUD; MAS, 2018). Alguns exemplos de funções de ativação utilizadas por um *perceptron* são demonstradas na Tabela 1.

Tabela 1 – Exemplos de funções de ativação utilizadas em um *perceptron*.

Função de ativação	Fórmula
Linear	$f(z) = z$
Step (Heavyside)	$f(z) = \begin{cases} 0, & \text{se } z < 0 \\ 1, & \text{se } z \geq 0 \end{cases}$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$

Fonte: Apicella *et al.* (2021).

Com as respostas conhecidas dos dados de treinamento de entrada, a etapa de treinamento é concluída. Treinar um *Multilayer Perceptron* é o procedimento pelo qual os valores para os pesos individuais são determinados de modo que o relacionamento que a rede está modelando seja resolvido com precisão. O objetivo do aprendizado é otimizar os pesos minimizando as diferenças entre os valores preditos e reais. A validação do modelo é feita usando

novos dados para mostrar como a configuração pode ser generalizada para novas situações. O *Multilayer Perceptron* é aplicado a uma ampla variedade de tarefas, todas as quais podem ser categorizadas como previsão, aproximação de função ou classificação de padrões (TAUD; MAS, 2018; GARDNER; DORLING, 1998).

2.3.1.5 XGBoost

As florestas de decisão combinam várias árvores de decisão para fornecer uma única saída em tarefas de aprendizado de máquina supervisionadas. A capacidade de as florestas de decisão integrar diferentes hipóteses em um único modelo e sua robustez a qualquer tipo de conjunto de dados relacional impulsionaram sua popularidade na comunidade de ciência de dados (ROKACH, 2016). *Árvores de Decisão Impulsionadas por Gradiente* (GBDT) é um subgrupo de florestas de decisão que inclui modelos como *XGBoost*, *CatBoost* e *LightGBM*. O mais notável é o *Aumento de Gradiente Extremo* (XGBoost), um sistema de aprendizado de máquina escalável para incremento de árvores. (SAGI; ROKACH, 2021).

GBDT é um método de aumento entre os melhores desempenhos na classificação de dados. Para entender o GBDT, é preciso primeiro entender o *Aumento de Gradiente* (GB). GB é um *framework* para *boosting*. O *boosting* é um algoritmo genérico que melhora um modelo fraco (por exemplo, regressão, árvores de decisão, etc.). O algoritmo de *boosting* alimenta então um modelo fraco com um subconjunto diferente de dados de treino. Cada vez que é chamado, o algoritmo de aprendizado fraco gera uma nova regra de predição fraca e após muitas rodadas, o algoritmo *boosting* combina essas regras fracas em uma única regra de predição que será mais precisa do que qualquer uma das regras fracas anteriores (SCHAPIRE, 2003).

A ideia principal do GB é construir sequencialmente cada modelo de árvore de decisão, onde cada um visa minimizar o erro do modelo anterior, com base no resíduo (a diferença entre o valor previsto e o valor real de cada instância), por meio de uma função de perda. A função de perda descreve a precisão dos modelos, quanto maior o valor da função de perda, pior a capacidade de previsão dos modelos treinados. Se o valor da função de perda diminui com a adição de novos modelos, o poder de previsão desses modelos melhora. A maneira esperada é deixar o valor da função de perda declinar na direção de seu gradiente descendente (ZHANG *et al.*, 2019).

XGBoost é uma implementação de árvores de decisão GB, onde pode ser um método muito eficaz para problemas de regressão e classificação (QIU *et al.*, 2021). Neste algoritmo,

as árvores de decisão são criadas sequencialmente. O *XGBoost* implementa pesos, os quais são atribuídos as variáveis de entrada necessárias na predição dos resultados. O peso das variáveis que realizam uma predição incorretamente pela árvore é aumentado e essas variáveis são alimentadas para a segunda árvore de decisão. Esses classificadores/preditores individuais então se combinam para fornecer um modelo forte e mais preciso. O *XGBoost* utiliza uma combinação ideal de técnicas de optimização de *hardware* e *software* para alcançar grande poder de processamento computacional.

2.3.1.6 Adaboost

Adaboost é um algoritmo de *Boosting*, que também funciona com base no princípio do método de adição gradual, em que vários modelos fracos são usados para obter modelos fortes. Começando com a amostra de treinamento sem peso, o *AdaBoost* constrói um preditor ou classificador, por exemplo, uma árvore de classificação. Se um ponto de dados de treinamento for predito incorretamente, o peso desse ponto de dados de treinamento será aumentado (*boosted*). Um segundo preditor é construído com os novos pesos, que não são mais iguais. Novamente, dados de treinamento mal preditos têm seus pesos aumentados e o procedimento é repetido. Uma pontuação é atribuída a cada preditor e o preditor final é definido como a combinação linear dos preditores treinados (HASTIE *et al.*, 2009).

3 PREDIÇÃO DO TVE COM APRENDIZADO DE MÁQUINA

Este Capítulo é dedicado ao OE1, que visa aprimorar o desempenho da predição do valor do tempo de vida do enlace, através do uso de técnicas de ML. Assim sendo, a contribuição principal deste tópico é a implementação de um esquema para predição do tempo de vida do enlace entre nós de VANETs, com o uso de algoritmos de ML. Esse processo inclui a proposição de atributos de predição e o treinamento de modelos para prever o TVE. Posteriormente é realizada uma avaliação de desempenho desses algoritmos de ML em uma abordagem conhecida de *offloading*.

3.1 Contextualização

A transmissão dos pacotes de dados entre veículos de origem e destino é frequentemente interrompida, resultando em retransmissões (LEE; ATKISON, 2021). A literatura recente considera o *Tempo de Vida do Enlace* (TVE) entre dois veículos um fator habilitador para melhorar a qualidade da comunicação em VANETs (KHATRI *et al.*, 2021). As soluções existentes não preveem o tempo de vida do enlace eficientemente ou não são flexíveis o suficiente para lidar com possíveis alterações de atributos no processo de predição de acordo com novos cenários (DENG *et al.*, 2020; SOUZA; VILLAS, 2020).

Esta primeira etapa da metodologia proposta investiga o uso de técnicas de predição baseadas em regressão para estimar o TVE entre nós em VANETs. Simulações foram executadas para gerar *datasets* com diferentes parâmetros que podem impactar o TVE entre veículos em cenários Urbano e de Rodovia. Os *datasets* foram utilizados para treinar e avaliar diferentes algoritmos de ML da literatura, para estimar o TVE. O algoritmo de ML com os melhores resultados foi utilizado como função de predição no processo de decisão de um algoritmo de *offloading*, sendo realizada uma avaliação de desempenho dessa abordagem com outros algoritmos de *offloading* da literatura.

As principais contribuições dessa primeira parte do trabalho são: (i) um método baseado em simulação de cenários Urbanos e de Rodovias para gerar *datasets* voltado para a pesquisa da predição do TVE, propondo um conjunto de atributos (*features*) de interesse. Os *datasets* estão disponíveis para a comunidade[1]; (ii) a comparação de algoritmos de ML baseados em regressão para avaliar a solução mais eficiente na predição do TVE em VANETs; e (iii) a

[1] https://github.com/PauloHGR/datasetsSBRC-2022

extensão de um algoritmo de decisão de *offloading* proposto em um trabalho anterior (SOUZA *et al.*, 2020) para usufruir da capacidade de predição do TVE e aprimorar as decisões de *offloading* de tarefas.

3.2 Trabalhos Relacionados

Esta seção apresenta trabalhos que têm utilizado diferentes algoritmos para prever a localização de objetos móveis (e.g., veículos), bem como estimar o TVE entre dois nós. A Tabela 2 apresenta um resumo das contribuições dos trabalhos relacionados e os compara sob quatro aspectos: atributos utilizados na predição, cenário de experimentação, uso de ML na predição e se o trabalha lida com *offloading* computacional.

Conforme a Tabela 2, o primeiro aspecto refere-se aos atributos utilizados na predição. A alta mobilidade do cenário veicular exige que fatores específicos sejam observados, dentre eles, a distância, direção e velocidade. O segundo aspecto é o cenário utilizado na simulação, onde se destacam cenários Rodoviários e Urbanos. O terceiro aspecto diz respeito ao uso de técnicas de ML para a predição, onde se destaca o trabalho de (ZHANG *et al.*, 2017) e este trabalho. Por fim, o quarto aspecto identifica os trabalhos no contexto de *offloading* computacional.

Em (YE *et al.*, 2019), os autores propõem um novo protocolo chamado *Predição de Mobilidade baseada em Protocolo de Roteamento* (MPBRP) que visa prever a mobilidade dos nós, através das posições e ângulos a fim de encontrar a melhor rota.

Em (KULLA *et al.*, 2019), os autores abordam um método que prevê o TVE entre dois veículos em movimento com base em sua velocidade relativa. Posteriormente, esse algoritmo de predição de tempo de vida é implementado com o protocolo *Vetor de Distância sob Demanda Ad hoc* (AODV), criando um protocolo chamado AODV com Predição do Tempo de Vida (AODV-LP).

Em (NABIL *et al.*, 2019), os autores propõem dois protocolos, um para prever a rota mais estável e outro para prever o tempo de entrega dos pacotes antes de enviar os dados. Foi desenvolvido um esquema de predição do TVE, visando garantir a eficiência dos protocolos. Os autores consideram a aceleração e a desaceleração de dois veículos em comunicação. Além da aceleração, a velocidade e o posicionamento dos veículos são variáveis utilizadas no cálculo do TVE. Os autores consideram um cenário de Rodovia nas simulações, não realizando testes em outros cenários, como os Urbanos.

Em (BUTE *et al.*, 2021), um esquema de *offloading* de tarefas é proposto na comuni-

Tabela 2 – Comparação com os Trabalhos Relacionados.

Trabalho	**Features de Predição**	**Cenário**	**Usa ML na predição?**	**Realiza *Offloading*?**	**Contribuição**
(YE *et al.*, 2019)	Distância, Direção e Velocidade	Urbano	Não	Não	Um novo protocolo, chamado MPBRP (Mobility Prediction Based Routing Protocol) que busca predizer a mobilidade dos nós através de posição e ângulos para prever a melhor rota.
(KULLA *et al.*, 2019)	Velocidade e Posição	Rodovia	Não	Não	Um método de predição do TVE entre dois veículos em movimento baseado em sua velocidade.
(NABIL *et al.*, 2019)	Posição, Velocidade e Aceleração	Rodovia	Não	Não	Implementação de dois protocolos. Um para predição da rota mais estável, e outro para a predição do tempo de entrega de pacote antes de enviar o dado. Um esquema de predição do TVE foi desenvolvido.
(BUTE *et al.*, 2021), (BUTE *et al.*, 2022)	Posição, Velocidade e Ângulo	Rodovia	Não	Sim	Esquema de *offloading* de tarefas para comunicação celular de veículo-para-tudo (C-V2X). Avaliação de desempenho com outros esquemas de *offloading* através de um cenário de rodovia.
(PASHA *et al.*, 2021)	Velocidade, Distância e Alcançe Máximo de Transmissão	Rodovia	Não	Não	Implementação de um modelo para uso de TVE Residual e de um salto em VANET.
(HUANG *et al.*, 2018)	Velocidade e Distância	Rodovia	Não	Sim	Um método de *offloading* em comunicação V2V através da implementação de uma SDN (Software Defined Network) dentro de uma arquitetura MEC (Mobile Edge Computing), denominada SDNi-MEC.
(SOUZA *et al.*, 2020)	Posição, Velocidade, Aceleração e Alcance de Comunicação	Rodovia	Não	Sim	Um esquema de melhoria da performance de *offloading* em rede multi-interface, através de um algoritmo de escolha guloso chamado GCF.
(ZHANG *et al.*, 2017)	TVE, Função de Similaridade SLS, Mobilidade Relativa, Distância e Velocidade	Rodovia	Sim	Não	Os autores implementaram um método de predição do TVE em ambiente VANET usando o algoritmo de ML Adaboost. Eles executaram um *benchmark* entre Adaboost e outros algoritmos de ML.
Este trabalho	Distância, Ângulo do cliente, Ângulo do servidor, Pseudo-ângulo, Velocidade, Velocidade Relativa, Aceleração Relativa e SLS	Rodovia e Urbano	Sim	Sim	Um esquema de predição do Tempo de Vida do Enlace através do algoritmo de ML SVR, escolhido através de uma avaliação de desempenho entre outros métodos de ML. O Algoritmo GCF é estendido com uma função de predição baseada no SVR.

Fonte: elaborada pelo autor.

cação celular V2X. Os veículos são agrupados em um *cluster*, onde enviam tarefas para veículos no mesmo *cluster* via comunicação V2V ou com um servidor de borda. Para realizar a predição do TVE, eles utilizam uma função de predição baseada em variáveis de posição, velocidade e ângulo. Os autores consideram apenas o cenário Rodoviário no ambiente dos experimentos. Em (BUTE *et al.*, 2022), os autores estendem o esquema anterior para suportar o *offloading* em servidores através de *Computação de Borda Veicular* (VEC) e melhorar um algoritmo de correspondência utilizado, para otimizar as decisões de *offloading*.

Em (PASHA *et al.*, 2021), os autores usam a predição do tempo de vida do enlace para identificar veículos ou nós próximos para realizar um encaminhamento de pacotes para um veículo de destino próximo. Esse encaminhamento de pacotes para nós próximos é chamado de salto. A utilização de saltos é importante, pois os protocolos de rede baseados em localização em VANETs empregam a estratégia de salto para adquirir conhecimento de uma determinada rota. Os autores propõem um algoritmo para calcular o tempo de vida do enlace do veículo

próximo antes de efetuar o salto. Os autores realizam a modelagem da proposta em um ambiente rodoviário com tráfego intenso, considerando a velocidade de ambos os veículos, a distância e o alcance máximo de transmissão como características de predição.

Em (HUANG *et al.*, 2018), os autores propõem um método de *offloading* na comunicação V2V implementando uma *Rede Definida por Software* (SDN) em uma arquitetura MEC. Os veículos enviam seus dados de contexto para o MEC, e o controlador SDN calcula informações sobre a rota ideal entre os veículos cliente e servidor. Para calcular essa rota, os autores propõem um método para calcular o tempo de vida do enlace no controlador SDN. Este método é calculado com base em duas características: velocidade relativa e distância.

Em (SOUZA *et al.*, 2020), os autores propõem um esquema para melhorar o desempenho de *offloading* de aplicativos computacionalmente intensivos e assim lidar com os desafios de mobilidade de ambientes veiculares. Os resultados mostram que o esquema proposto reduz o tempo total de *offloading* em até 54,1% e aumenta a taxa de sucesso do *offloading* em 71,8% em comparação com outras abordagens. No entanto, o trabalho de Souza *et al.* (2020) utiliza um algoritmo de predição do TVE baseado em equação matemática, que poderia ser substituída por uma solução de predição por ML. O presente trabalho propõe o uso de algoritmos de Aprendizado de Máquina para melhorar a eficácia da predição do TVE e consequentemente aumentar as taxas de sucesso de *offloading*.

Em (ZHANG *et al.*, 2017), um método de predição do TVE é proposto em ambientes VANET usando o algoritmo de Aprendizado de Máquina *Adaboost*. Um *benchmark* entre o *Adaboost* e outros métodos de Aprendizado de Máquina foi realizado, porém, os autores não incluem soluções de predição do TVE baseadas em formulações matemáticas no *benchmark*. As simulações consideraram um ambiente de Rodovia, não sendo utilizados cenários Urbanos. Com base nas métricas de regressão avaliadas, os resultados do presente trabalho no cenário de Rodovia apresentaram menores taxas de erro do que o cenário de Rodovia implementado pelos autores.

Em suma, o presente trabalho apresenta oito variáveis para lidar com o problema de previsão do tempo de vida do enlace em VANETs. Trabalhos anteriores não consideraram algumas das variáveis propostas como pseudo-ângulo, ângulo e *Semelhança de Localidade Espacial* (SLS). Outra diferença relevante é que tanto um cenário urbano, quanto um cenário de rodovia, são considerados na avaliação do processo de predição. Esse esquema de predição em um ambiente variado proposto neste trabalho, não é considerado nos demais trabalhos, sobretudo

devido à complexidade de um ambiente urbano.

3.3 Metodologia

Esta seção apresenta a parte relacionada ao OE1 desta dissertação, onde é demonstrada a metodologia utilizada para aplicar técnicas de ML na predição do TVE.

3.3.1 Problemática

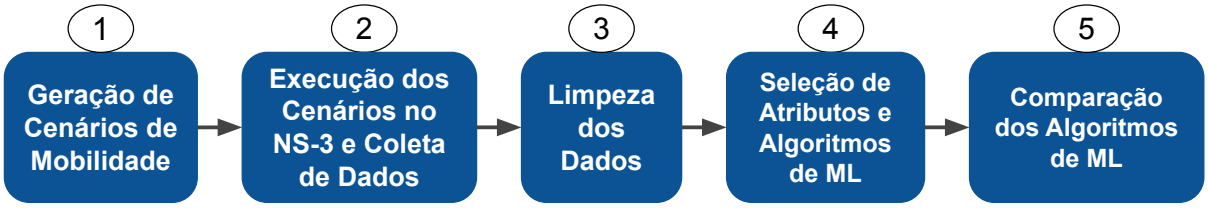
A inferência do TVE é importante em muitas aplicações na área de redes veiculares, visto que essas aplicações dependem cada vez mais do compartilhamento de informações, coleta e transmissão de dados, monitoramento de situações específicas, ou seja, dependem de uma boa comunicação com outras entidades para transmitir suas funcionalidades. A estimativa do TVE permite que as aplicações melhorem seus processos de decisão para saberem se estabelecem ou não uma comunicação. No processo de *offloading* computacional, a inferência do TVE é uma funcionalidade essencial, já que essa informação é um recurso que um algoritmo de decisão de *offloading* considera antes de decidir quantas e quais tarefas serão enviadas para serem processadas em um servidor remoto ou se não serão enviadas.

Atualmente o processo de predição do TVE é realizado majoritariamente por cálculos e equações diversas. Essa abordagem de predição traz três principais problemas ou limitações: (I) Baixa precisão na inferência do TVE, o que pode ocasionar altas taxas de erros, significando uma distância maior do valor de TVE predito em relação ao valor real; (II) Os teoremas de cálculo da predição utilizam um número limitado de variáveis, impedindo uma escalabilidade na inserção de variáveis na predição. Paralelamente, isso vai de lado oposto ao crescimento das redes veiculares, que devido a um número maior de sensores e informações compartilhadas, permite um surgimento de novos atributos; (III) Outro problema no processo de predição do tempo de vida do enlace é a limitação nos cenários utilizados. Os trabalhos se concentram em apenas uma categoria de cenário na etapa de coleta de atributos no processo de predição. Logo, as características e peculiaridades inerentes a cada ambiente de comunicação veicular acabam sendo deixadas de lado, o que pode ocasionar predições de valores não condizentes para cada contexto.

3.3.2 Fluxo de Etapas no Processo de Predição do TVE

A proposta desse trabalho consiste em desenvolver um esquema de predição do TVE para o processo de *offloading* computacional, que trate as limitações abordadas na Subseção 3.3.1. O esquema de predição proposto é uma abordagem baseada em ML, que visa detalhar as etapas necessárias para realizar um procedimento de predição do TVE. Os trabalhos que utilizam predição do TVE baseado em ML, não propõe as etapas abordadas neste trabalho, abstraindo grande parte do conhecimento e não entregando detalhes que podem ser importantes no aperfeiçoamento de todo o processo de predição do TVE e de outras variáveis importantes no *offloading* computacional. Dentre as principais contribuições que este esquema pode trazer, destaca-se a disponibilização de cenários e *datasets*, os quais são artefatos escassos na literatura e um conjunto de variáveis de entrada, que possibilitem a extração de padrões e a consequente predição do TVE. A descrição e proposição do fluxo de etapas permite um melhor ordenamento para a realização, aperfeiçoamento ou proposição de novas abordagens de predição do TVE, contribuindo para uma melhor disseminação do uso de ML para predição do TVE. Por fim, uma abordagem de predição do TVE baseada em Regressão é envelopada em um modelo de aprendizado treinado e utilizada como método de predição do TVE de um algoritmo de *offloading* computacional. Diversas etapas são necessárias para implementar o método de predição baseado em Regressão. A Figura 17 ilustra as etapas realizadas.

Figura 17 – Fluxo de Trabalho do Esquema de Predição Proposto.



Fonte: elaborada pelo autor.

O processo começa com a implementação de simulações com diferentes cenários de VANETs para coletar os atributos necessários para o processo de aprendizagem. Após esta etapa, é coletado e calculado um conjunto de atributos de comunicação entre os veículos. Cada cenário gera um *dataset* que passa por um processo de limpeza para torná-lo adequado ao treinamento. Cada *dataset* de um cenário é dividido em *dataset* de treino e teste, onde diferentes algoritmos de ML realizam o processo de treinamento de um modelo, utilizando o *dataset* de treino. Finalmente, o *dataset* de teste é utilizado para realizar a predição do valor do TVE e,

então, realiza a avaliação do algoritmo de ML, com base em diferentes métricas.

3.3.3 Geração de Cenários

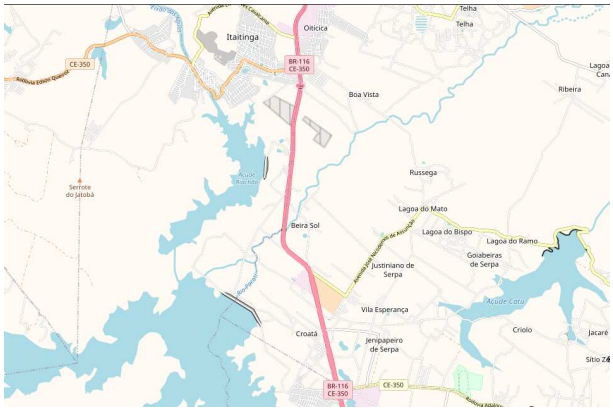
Uma parte importante do processo de aprendizagem é identificar os fatores que afetam uma característica que se deseja medir. Visando coletar esses fatores, também chamados atributos, foram construídos dois cenários de rede veicular: um de Rodovia e um Urbano. Ambos os cenários são amplamente utilizados em aplicações de rede veicular (HUSSAIN *et al.*, 2019; SUN *et al.*, 2020).

Alguns cenários com *traces* de tráfego simulados de cenários reais foram escolhidos nessa primeira etapa, de modo a montar dados mais realísticos possíveis. Para a realização da avaliação, foi escolhido um trecho de 5 quilômetros da BR-116 na região metropolitana de Fortaleza, Brasil (Figura 18(a) e 18(b)) para o cenário de Rodovia. O cenário Urbano consiste em um trecho de 2 quilômetros quadrados do centro de Manhattan, Nova York, EUA (Figura 18(c) e 18(d))(KIM *et al.*, 2020; ECKERMANN *et al.*, 2019; SOUZA *et al.*, 2020). O *Simulador de Mobilidade Urbano* (SUMO)[2] e o *Simulador de Rede 3* (ns-3)[3] foram utilizados para executar os cenários simulados e realizar a etapa de coleta de dados. O SUMO é uma ferramenta de geração de mapas e movimentações de veículos ou pedestres. O ns-3 é um simulador de rede consolidado que permite a utilização dos cenários gerados pelo SUMO. As Figuras 18(b) e 18(d) também ilustra as representações no SUMO para os cenários considerados.

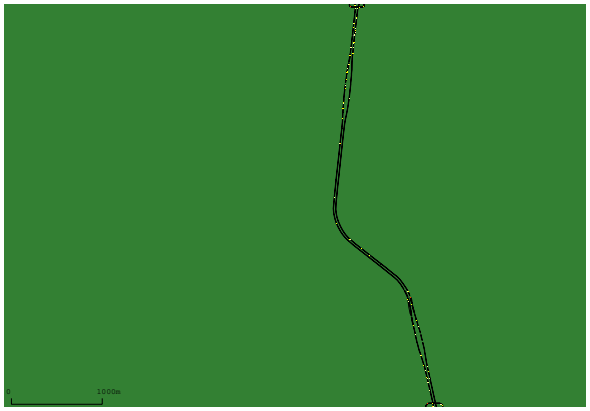
O tempo da simulação foi definido em 85 segundos, já que é um intervalo de tempo suficiente para gerar os conjuntos de dados, sem muitos dados redundantes. O alcance máximo de transmissão dos veículos foi configurado em 250 metros, através do modelo de rádio-propagação *Two-Ray Ground*, uma vez que após 250 m, há uma diminuição na força de sinal do modelo *Two-Ray Ground*, o que poderia ocasionar maior perda de dados(NOBRE *et al.*, 2010). Foi utilizada uma taxa de transferência de até 27 Mbps sobre uma largura de banda de canal de 10MHz permitida pelo módulo IEEE 802.11p do ns-3. O número de veículos em cada cenário varia devido às diferenças em suas topologias. Foram utilizados 100 veículos ao longo de 5 km no cenário de Rodovia e 200 veículos no cenário Urbano de 2 km². Tal quantidade foi utilizada a fim de evitar sobrecargas com mais veículos ou geração de pouca informação nos conjuntos de dados com menos veículos. As viagens de cada veículo dentro destes cenários são geradas

[2] https://www.eclipse.org/sumo/
[3] https://www.nsnam.org/

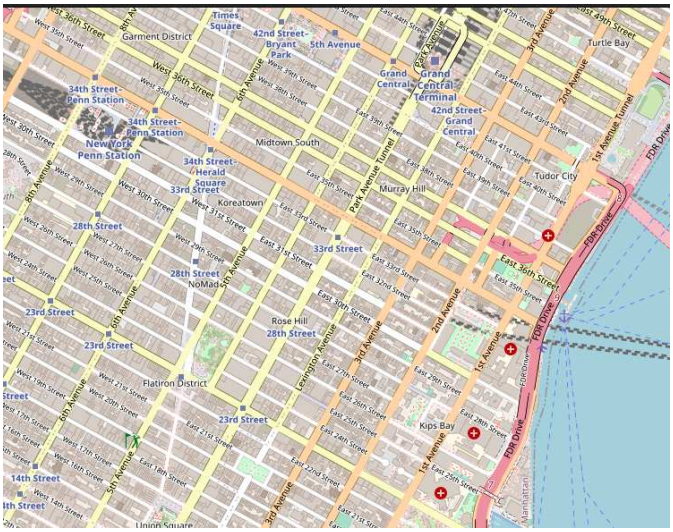
Figura 18 – Representação Gráfica dos Cenários.



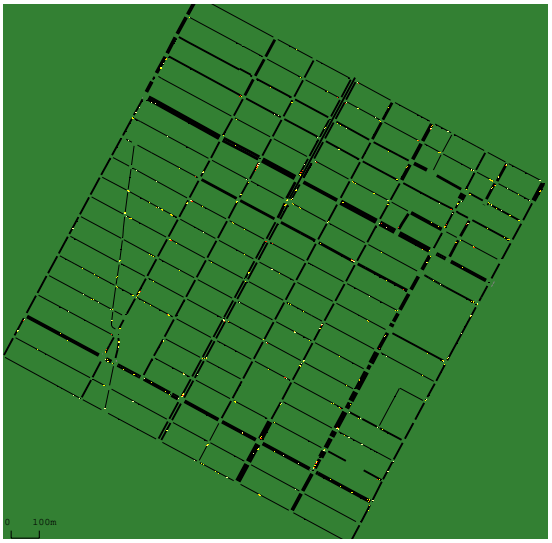
(a) Mapa da Rodovia.



(b) Cenário de Rodovia no SUMO.



(c) Mapa Urbano.



(d) Cenário Urbano no SUMO.

Fonte: OpenStreetMap e SUMO.

automaticamente pelo SUMO, respeitando as regras das vias e trajetos do mapa. Dessa forma, os parâmetros de configuração de cada viagem (como velocidades e posições dos veículos) são armazenados em um *trace* de mobilidade.

3.3.4 Coleta de Dados e Definição de Atributos

Com os cenários definidos e configurados no simulador, o próximo passo é a execução da simulação e coleta de dados. O simulador ns-3 permite coletar atributos relacionados à velocidade (V_x, V_y), à posição no plano (P_x, P_y) e à distância entre dois veículos, tais dados podem ser melhor visualizados no arquivo em anexo A. A partir das coordenadas do vetor velocidade pode-se obter a velocidade escalar como $Speed = \sqrt{V_x^2 + V_y^2}$. As coordenadas x e y dos vetores de posição e de velocidade são essenciais para obter atributos adicionais como, por exemplo,

ângulos, velocidade relativa e aceleração relativa. Exemplos, de extração dos dados de vetores do arquivo de mobilidade do SUMO pelo ns-3, pode ser visualizado no Apêndice A.

Os veículos fazem parte de uma rede de comunicação V2V (Veículo-para-Veículo), onde cada veículo solicita e inicia uma comunicação com veículos vizinhos em um esquema baseado em *broadcast*. Quando a comunicação é estabelecida, os veículos permanecem enviando/recebendo pacotes até que sua comunicação encerre. Durante o tempo de comunicação, o carro transmissor i (cliente) envia pacotes com suas informações de velocidade e posição em um determinado intervalo de tempo. Ao receber os pacotes, o carro receptor j (servidor) calcula as suas informações de velocidade e posição, além de outros atributos discutidos posteriormente. Por fim, esses dados são armazenados em um arquivo externo, que serve como um *dataset*.

Os trabalhos relacionados se concentram em um conjunto menor de atributos para prever o TVE, conforme resumido na Tabela 2. Neste trabalho, oito variáveis ou atributos de entrada são utilizadas no processo de predição. São atributos consolidados em diversos trabalhos na literatura e representam valores que podem ser coletados em um ambiente real de trânsito. A distância D_{ij} entre veículos é obtida através da seguinte fórmula: $D_{ij} = \sqrt{(P_{xi} - P_{xj})^2 + (P_{yi} - P_{yj})^2}$.

A partir do vetor velocidade, é possível calcular os ângulos Θ_i e Θ_j relativos aos veículos transmissores e receptores, respectivamente, através da equação : $\Theta = \arctan\left(\frac{V_y}{V_x}\right)\frac{180}{\pi}$.

Também é considerado o pseudo-ângulo entre os dois veículos, utilizado quando é preciso comparar ângulos diferentes, mas sem grandes custos de desempenho (NEŠOVIĆ, 2017). O pseudo-ângulo é utilizado para determinar a posição do veículo i em relação à posição do veículo j. Logo, o pseudo-ângulo é utilizado para saber se veículos estão se aproximando ou distanciando-se um do outro. O código de implementação desse atributo está listado no Apêndice B.

A soma do quadrado das velocidades de i e j é outro atributo considerado neste trabalho. O objetivo é dar mais peso às velocidades mais altas ou às maiores diferenças de velocidade entre os carros, já que, no ambiente de simulação, as velocidades podem ser semelhantes. O atributo é calculado por $S_v = Speed_i^2 + Speed_j^2$.

A relação entre velocidade e distância é feita pelo cálculo da SLS, seu uso neste trabalho deve-se ao fato dessa variável utilizar a distância e a diferença de velocidade entre os veículos para indicar o comprimento do enlace, com isso o tempo de vida do enlace é um dado possível de ser extraído a partir do SLS. O SLS foi considerado um atributo importante no trabalho de Zhang *et al.* (2017). Ao calcular o SLS, deve-se definir a velocidade máxima e

o alcance máximo de transmissão. Esses valores são definidos respectivamente por $S_{max} = 16$ m/s, o qual é a velocidade máxima nos cenários utilizados, e $r = 250$ metros. O atributo é então calculado por:

$$SLS = \left(1 + \frac{D_{ij}^2}{r^2} + \left(Speed_i - \frac{Speed_j}{2} S_{max}\right)^2\right)^{-1/2}. \tag{3.1}$$

Também são utilizados outros dois atributos mencionados em (NABIL *et al.*, 2019) e (HÄRRI *et al.*, 2008), a *Velocidade Relativa* (RS) e a *Aceleração Relativa* (RA) do carro cliente em relação ao carro vizinho. Essas variáveis são dadas por:

$$RS = (V_{xi} - V_{xj})^2 + (V_{yi} - V_{yj})^2, \tag{3.2}$$

$$RA = 2[(P_{xi} - P_{xj})(V_{xi} - V_{xj}) + (P_{yi} - P_{yj})(V_{yi} - V_{yj})]. \tag{3.3}$$

A variável de saída (valor a ser predito) será o tempo de comunicação Tc entre o veículo i e o veículo j. A variável alvo, ou variável dependente, é o valor que deverá ser predito usando técnicas de ML. O valor de Tc para dois veículos durante a simulação é determinado ao contar os registros armazenados pelo carro j de cada pacote entregue pelo carro i. Cada veículo possui um identificador $IdCar$, armazenado com os demais atributos. Outra informação capturada da simulação é o instante t em que ocorre a comunicação entre o veículo i e j. Com os valores de $IdCar$ dos carros i e j, é percorrido todo o *dataset* até chegar ao último registro de ambos os carros. Nesse ponto é identificado o instante do veículo i (t_i) e o instante do veículo (t_j). O Algoritmo 1 resume e formaliza esse procedimento.

O instante t_j é quando o carro vizinho (j) recebe o pacote do carro transmissor da simulação (i) e vice-versa para o instante t_i. O algoritmo busca linha por linha (l) em cada *dataset* os registros de comunicação de carros com $IdCar_i$ e $IdCar_j$. O primeiro registro é armazenado em r pelo algoritmo, e a variável c recebe o valor 1, que indica para as iterações subsequentes que o primeiro registro está em r. A variável $TransTime$ refere-se ao tempo de transmissão do pacote. É comparada a diferença entre o registro atual e o anterior com o tempo de transmissão do pacote. Se essa diferença de registro for menor que o tempo de transmissão, significa que o registro faz parte dos dados trocados naquele intervalo de tempo entre os veículos. Então, o t_f do primeiro registro é atualizado para receber o t_i do registro atual. Se a diferença de tempo nos registros for maior que o tempo de transmissão do pacote, significa que houve uma queda de enlace, sendo que os próximos registros são referente a outra comunicação e o valor de t_f é atualizado no *dataset*. Por fim, quando não forem encontrados mais carros com os identificadores especificados, as comparações terminam e $Tc = r[t_f]$ é executado.

Algoritmo 1: Processamento do Tempo de Comunicação.

Input: $IdCar_i$, $IdCar_j$, {dataset}, $TransTime$
Output: {dataset} + Tc

```
1  c ← 0;
2  r ← [];
3  foreach l in {dataset} do
4      if {IdCar_i,IdCar_j} ⊂ l then
5          if c = 0 then
6              r ← l;
7              c ← 1;
8          else
9              if (l[t_i] − r[t_i]) < TransTime then
10                 r[t_f] ← l[t_i];
11             else
12                 {dataset} ← r[t_f];
13                 r ← l;
14             end
15         end
16     end
17 end
18 if r ≠ ∅ then
19     {dataset} ← r[t_f];
20 end
```

3.3.5 Processo de Limpeza de Dados

Antes de treinar os modelos de ML, foi realizado um pré-processamento dos *datasets* para eliminar dados coletados por meio de eventos que podem gerar distorções na extração de padrões e posterior predição do TVE. Um desses eventos é quando perdas de comunicação ocorrem entre o cliente e os veículos vizinhos, havendo um restabelecimento da comunicação em seguida. Assim, é considerada uma janela mínima de 1 segundo durante a comunicação ociosa entre os veículos para armazenar o registro no *dataset*. Tal estratégia elimina quebras de conexão curtas, que podem comprometer o treinamento ao gerar *datasets* com muitas conexões breves.

Durante a simulação, é comum que veículos parados estabeleçam comunicação entre si. Nesse caso, amostras desses veículos foram eliminadas, já que veículos parados geram grandes tempos de comunicação, o que poderia ocasionar algum viés errôneo. Por consistência, também foram removidas amostras coletadas com distâncias superiores a 250 metros, o alcance máximo configurado.

Por fim, são identificados e removidos dados ruidosos referentes a amostras com tempos de comunicação distorcidos. A limpeza de dados distorcidos é realizada dividindo-se cada conjunto de dados em grupos. No cenário Urbano, o conjunto de dados é dividido com base na distância, ângulos e pseudo-ângulo. A partir dessas informações, são identificadas as

seguintes distorções a serem removidas: (1) carros se afastando, com grande distância e alto tempo de comunicação; (2) carros se aproximando com pouca distância um do outro e baixo tempo de comunicação.

3.3.6 Seleção dos Algoritmos de ML

Após construir e limpar os conjuntos de dados, os algoritmos de ML a serem avaliados na solução são selecionados. O objetivo é identificar a melhor abordagem de aprendizado para prever o TVE para cenários variados. São considerados métodos de regressão para prever o tempo de comunicação com base nas variáveis independentes previamente detalhadas. As seguintes técnicas são avaliadas: ε-SVR, KNN, *Random Forest*, *XGBoost* e Rede Neural (*Perceptron Multicamada* (MLP)). A Tabela 3 mostra os parâmetros usados na configuração de cada técnica de ML.

O método *Grid Search* foi utilizado para selecionar a combinação de hiperparâmetros com a menor taxa de erro de predição. Na rede neural, é utilizada a biblioteca *Keras*[4] e algumas combinações de hiperparâmetros fornecidos pelo *Keras* são testadas, com o melhor resultado mostrado na Tabela 3. Para realizar a medição das métricas, foi utilizada uma máquina virtual com uma *CPU Intel(R) Xeon(R) E5-2650 v3 @2.30GHz*, 1 núcleo de *CPU*, 1 GB de *RAM* e sistema operacional *Linux Ubuntu 20.04.2 LTS*, escolhida para simular um dispositivo com restrição de recursos, i.e., simulando um veículo.

Tabela 3 – Hiperparâmetros dos modelos de ML.

Algoritmo	**Hiperparâmetros**
Rede Neural	loss_function: mae; optimizer: adam, metrics: mse,mae; epochs: 1500
SVR	C:8, epsilon:0.05, gamma:1.0, kernel:rbf
KNN	n_neighbors:13; weights:distance; n_estimators:500
Random Forest	max_depth:100; max_features:3; min_samples_leaf:2; min_samples_split:8; n_estimators:500
XGBoost	colsample_bytree: 0.8; learning_rate: 0.1; max_depth: 8; min_child_weight: 3; n_estimators: 150, nthread: 4, subsample: 0.8

Fonte: elaborada pelo autor.

3.3.7 Avaliação das Abordagens de ML

Foi realizada uma comparação das abordagens de ML propostas com um algoritmo tradicional para prever o TVE entre nós em redes veiculares. O cálculo denominado *Link Lifetime*

[4] https://keras.io/

(LLT) é utilizado em diversos trabalhos na literatura em aplicações VANET e encontra-se bem descrito em Nabil *et al.* (2019), Härri *et al.* (2008), Souza *et al.* (2013).

As seguintes métricas foram utilizadas neste trabalho para medir a eficiência das abordagens de predição: *Erro Médio Absoluto* (MAE), *Erro Percentual Médio Absoluto* (MAPE), também conhecido como *Erro Médio Relativo* (MRE) e *Raiz do Erro Quadrado Médio* (RMSE). Essas métricas são calculadas por:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Tc - Yp|; \quad MAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|Tc - Yp|}{Tc}; \quad RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}Tc - Yp^2}, \qquad (3.4)$$

em que Tc é o valor real do tempo de duração da comunicação e Yp é o valor predito pelas soluções.

Para cada cenário foi utilizado um *dataset*, onde cada *dataset* dos cenários de Rodovia e Urbano foi dividido em outros dois conjuntos: um conjunto de treinamento (80% das amostras) e um conjunto de teste (20% restante). As previsões realizadas pelos modelos treinados nos *datasets* de teste foram avaliadas usando as métricas definidas anteriormente. Durante a simulação, foram calculados os valores a serem preditos, referentes ao TVE, pelo algoritmo LLT. Esses mesmos valores foram usados no conjunto de teste para calcular os erros do LLT.

A Tabela 4 mostra as métricas obtidas. O MAE demonstra a distância entre os valores reais de tempo de comunicação mensurados e os valores de tempo preditos. O MAPE é o percentual médio da diferença entre o valor predito e o valor real. O RMSE dá um peso maior para valores que sejam muito diferentes entre o previsto e o real, ideal para analisar o impacto de *outliers*. Os resultados indicam que a abordagem tradicional (LLT) apresenta previsões mais extremas com base nas altas taxas de erro relatadas pelas métricas. As soluções de ML generalizam bem e apresentam tempos de comunicação preditos mais próximos da realidade.

Por fim, foi medido o tempo que cada abordagem leva para prever o TVE, conforme também mostrado na Tabela 4. Inicialmente, supõe-se que cada modelo de ML esteja disponível e carregado em um servidor. Então, é realizada a medição do tempo para ler os atributos e inferir o valor predito. O algoritmo LLT foi implementado em um *script* fora do simulador. O processo de medição foi realizado fora do ambiente de simulação, pois o ns-3 utiliza tempo virtual, diferente do tempo real de execução do *hardware*.

Tais métricas são essenciais para decidir qual abordagem de predição escolher para *offloading* computacional em ambientes com elevado tráfego de veículos. Essa categoria de cenário seria o pior caso para tomada de decisão, pois o número de predições depende do número

Tabela 4 – Sumário dos resultados dos experimentos.

Algoritmo	Cenário	MAE (s)	MAPE (%)	RMSE (s)	Tempo de Predição (s)
LLT	Rodovia	24,14	208,86	39,59	**2,25e-05**
Rede Neural	Rodovia	1,89	**34,55**	3,13	1,03e-01
SVR	Rodovia	**1,82**	35,53	**3,00**	5,20e-04
KNN	Rodovia	1,99	37,29	3,12	3,66e-01
Random Forest	Rodovia	2,01	38,03	3,15	4,00e-02
XGBoost	Rodovia	2,07	37,04	3,24	2,70e-03
LLT	Urbano	25,68	190,44	40,08	**2,25e-05**
Rede Neural	Urbano	**1,65**	**19,31**	2,96	1,05e-01
SVR	Urbano	1,85	21,84	2,92	5,20e-04
KNN	Urbano	1,93	22,64	2,90	3,61e-01
Random Forest	Urbano	1,84	20,43	**2,79**	4,20e-02
XGBoost	Urbano	1,96	20,94	2,95	2,60e-03

Fonte: elaborada pelo autor.

de veículos que respondem ao processo de descoberta. Logo, muitos veículos gerariam uma sobrecarga de predições.

3.4 Resultados dos Experimentos

Esta Seção apresenta os experimentos e os resultados relacionados ao impacto causado pelo uso de técnicas de predição no *offloading* computacional. A Subseção 3.4.1 aborda como os diferentes algoritmos de ML afetam o tempo necessário para a tomada de decisão no processo de *offloading*. A Subseção 3.4.2 apresenta a avaliação de desempenho de uma solução de *offloading* computacional ao usar a solução de predição proposta.

3.4.1 Tempo de Decisão do Offloading Computacional

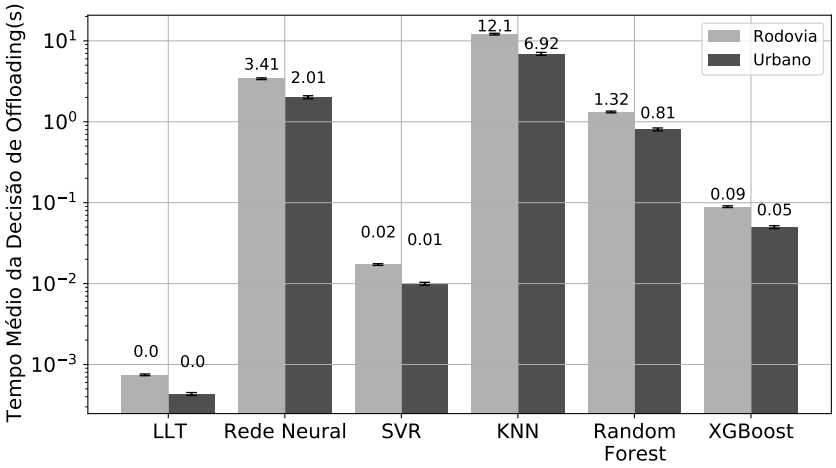
Foi realizado um experimento para inferir o tempo total médio que cada abordagem de ML requer para executar as predições de TVE. O objetivo é verificar quanto tempo um algoritmo de *offloading* aguarda antes de tomar a decisão de alocar tarefas. Esse ponto é bastante importante, pois antes de decidir, o algoritmo de *offloading* deve analisar o tempo previsto de cada veículo vizinho.

São realizadas simulações em Rodovias e ambientes Urbanos com alto tráfego, utilizando os maiores valores mostrados na Tabela 5. O tráfego mais denso é utilizado para analisar o desempenho dos modelos de ML e a abordagem LLT na pior categoria de cenário, onde há uma demanda maior nos modelos de predição. Primeiro, foi verificada a taxa média de vizinhos que um veículo encontra durante a simulação. Os valores médios que um determinado

veículo encontrou de veículos vizinhos foram 29 para o cenário Rodoviário e 23 para o cenário Urbano. Em seguida, foi realizado o produto desse resultado com o tempo que cada algoritmo leva para inferir o TVE, conforme a tabela 4, a fim de saber quanto tempo levaria para realizar o processo de escolha do veículo no *offloading*, já que deve haver a predição do TVE para todos os veículos.

Conforme mostrado na Figura 19, uma abordagem de *offloading* usando o modelo SVR resulta em um tempo de decisão menor do que as outras abordagens de ML. O LLT tem um tempo de predição próximo de zero por ser um algoritmo aritmético simples. As abordagens KNN, *Random Forest* e Rede Neural têm tempo total de decisão alto, o que significa que um algoritmo de *offloading* teria que esperar muito tempo até obter todos os tempos previstos antes de decidir.

Figura 19 – Tempo de decisão para o *offloading* nos cenários avaliados.



Fonte: elaborada pelo autor.

Esses resultados mostram que abordagens específicas de ML apresentam muita sobrecarga em cenários de alto tráfego. A sobrecarga é caracterizada pelo tempo que leva para tomar a decisão. Muita sobrecarga afeta o processo de decisão do algoritmo de *offloading*, pois ele deve esperar muito tempo antes de decidir. Em um ambiente de rede veicular dinâmico, longas esperas podem resultar em quebras de comunicação, perda de contato com veículos vizinhos e informações incompletas. Portanto, embora as abordagens de Rede Neural, KNN e *Random Forest* apresentem previsões com baixos erros, seu uso é inviável no *offloading* devido à sobrecarga que podem causar, pelo menos ao considerar um veículo com as capacidades computacionais descritas na Subseção 3.3.6.

3.4.2 Desempenho Geral do Offloading Computacional

A eficiência da abordagem proposta baseada em ML na predição de TVE é avaliada em uma aplicação VANET real. Os seguintes experimentos de *offloading* computacional consideraram apenas o modelo de aprendizado SVR, devido ao seu bom desempenho preditivo (baixas taxas de erro) e menor sobrecarga no processo de decisão de *offloading* em ambientes de alto tráfego, conforme a Figura 19.

A comparação foi feita com três algoritmos de *offloading* computacional: um algoritmo de escolha aleatória, o *Nuvem Veicular Híbrida* (HVC) e o *Guloso por CPU Livre* (GCF) de acordo com (SOUZA *et al.*, 2020). Tais algoritmos de *offloading* usam o algoritmo LLT como função para predizer o TVE entre os veículos. Conforme os resultados dos experimentos apresentados em (SOUZA *et al.*, 2020), o GCF apresentou as melhores taxas de sucesso no processo de decisão de *offloading*. Assim, o GCF foi escolhido e sua função de cálculo de TVE foi substituída pela abordagem baseada em ML. A solução resultante é chamada *Guloso por CPU Livre com Aprendizado de Máquina* (GCFML).

Foi executada uma simulação de *offloading* computacional nos cenários de Rodovia e Urbano usando o simulador ns-3. Os cenários considerados nos testes foram os mesmos da etapa anterior de coleta de dados. Um veículo é escolhido aleatoriamente para ser o transmissor da tarefa. Nos experimentos, foram utilizados dois fatores: número de veículos e tamanho da tarefa. O mesmo número de veículos de (SOUZA *et al.*, 2020; MONTEIRO *et al.*, 2012) foi utilizado nos ambientes de Rodovia e Urbano, conforme apresentado na Tabela 5.

Foram utilizados valores de tamanho da tarefa maiores que em (SOUZA *et al.*, 2020). O objetivo de colocar cargas de trabalho maiores era testar os algoritmos de *offloading* em situações mais críticas, que exigem maior eficiência na predição do TVE. Os fatores com seus níveis individuais para cada cenário também são mostrados na Tabela 5.

Tabela 5 – Fatores e níveis considerados nos experimentos.

Número de Veículos no Ambiente Urbano	**Número de Veículos na Rodovia**	**Tamanho da Tarefa (MB)**	**Nome**
50 (esparso)	55 (esparso)	1,5	E1
50 (esparso)	55 (esparso)	3,0	E2
275 (médio)	275 (médio)	1,5	M1
275 (médio)	275 (médio)	3,0	M2
509 (denso)	605 (denso)	1,5	D1
509 (denso)	605 (denso)	3,0	D2

Fonte: elaborada pelo autor.

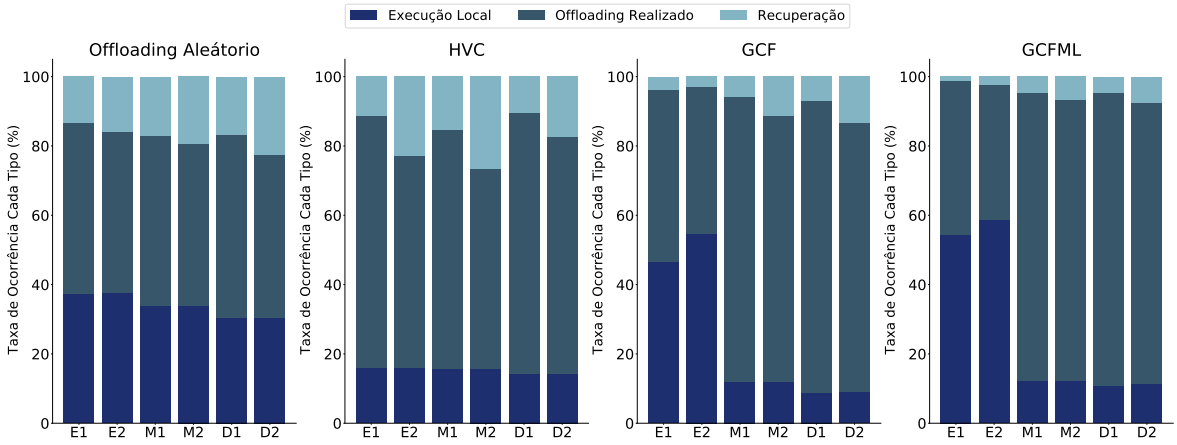
Os testes foram executados em uma máquina com as seguintes configurações: *Processador Intel(R) Xeon(R) E5-2650 v3 CPU @2.30GHz* com 8 núcleos de *CPU* e 16 GB de *RAM*, executando o sistema operacional *Linux Ubuntu 18.04.2 LTS*. Os modelos foram treinados usando os *datasets* de cada cenário (Rodovia e Urbano). Quando um veículo cliente encontra um veículo vizinho, ele realiza uma consulta para prever a duração do TVE entre eles. Os atributos são coletados e passados para o modelo que retorna o tempo predito.

No cenário de *offloading* configurado, um veículo ou cliente transmissor envia quatro tarefas para veículos vizinhos que podem processar essas tarefas, conforme (SOUZA *et al.*, 2020). As tarefas são processadas em veículos vizinhos, que retornam o resultado ao cliente. Em seguida, três métricas são avaliadas: a taxa de sucesso de *offloading*, a taxa de recuperação ou falha e a taxa de execução local. Se o processamento das quatro tarefas for bem-sucedido, significa que o *offloading* foi bem-sucedido. Caso algum veículo vizinho não retorne o resultado, houve uma falha de comunicação durante a etapa de processamento, sendo necessário realizar uma recuperação. Assim, o veículo cliente realiza o processamento de uma cópia dessa tarefa localmente. Quando não há veículos com capacidade suficiente para alocar todas as tarefas, elas são processadas localmente no veículo cliente.

Os resultados obtidos nos cenários de Rodovia (Figura 20) e Urbano (Figura 21) revelam o potencial do modelo SVR como substituto da abordagem tradicional de previsão. O GCFML demonstrou maior eficiência no cenário Rodoviário, apresentando taxas de recuperação de tarefas inferiores em comparação com suas contrapartes, principalmente em relação ao GCF. Em relação ao GCF, observou-se uma redução de até 66% no cenário L1 e aproximadamente 40% nos cenários M2 e H2. Nos cenários com tráfego médio e alto, a diminuição no número de recuperações resultou em taxas de *offloading* bem-sucedido aprimoradas, alcançando até 5% nos cenários M2 e H2.

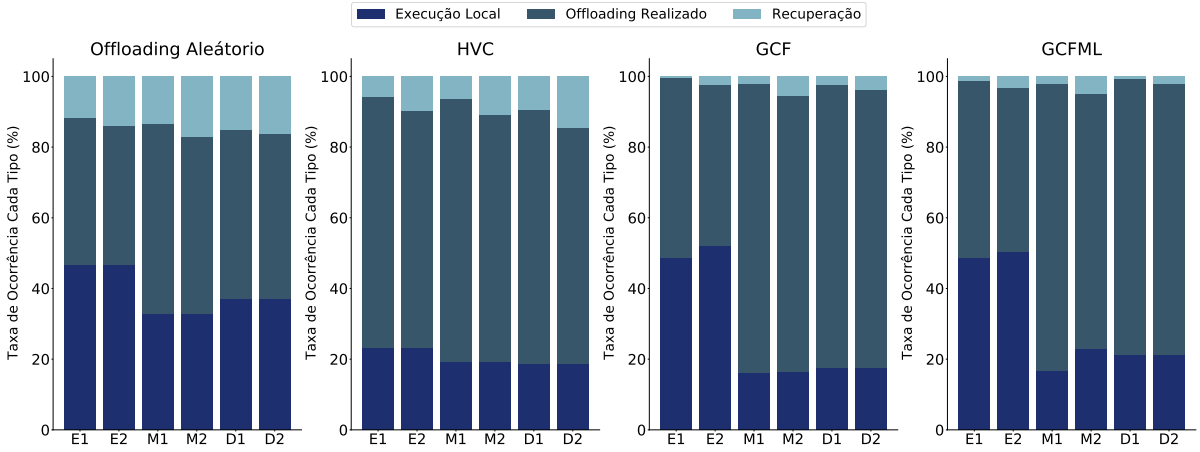
No ambiente Urbano, a abordagem proposta diminuiu as taxas de falha nas seguintes situações: cenários com menos tarefas e cenários de tráfego com muitos veículos. O GCFML toma melhores decisões de *offloading* ao optar por executar localmente em vez de enviar uma tarefa que será perdida e recuperada posteriormente. Portanto, o GCFML tem uma taxa de execução local mais alta. Outro motivo é o maior dinamismo e imprevisibilidade do cenário Urbano, levando a previsões de menores tempos de comunicação. O algoritmo de *offloading* tem menos opções para enviar tarefas com predições de TVE mais curtas, resultando em mais execuções locais.

Figura 20 – Resultados do *offloading* para o cenário de Rodovia.



Fonte: elaborada pelo autor.

Figura 21 – Resultados do *offloading* para o cenário Urbano.



Fonte: elaborada pelo autor.

Geralmente, os resultados demonstram que o modelo SVR reduz a taxa de recuperação no processo de *offloading* computacional em redes veiculares. Uma taxa de recuperação mais baixa fornece melhor qualidade de serviço para aplicações e usuários, ao haver menos retransmissões e menos processos de correção de erros. Como a Figura 20 indica, o cenário de Rodovia é o mais desafiador para as predições em comparação com o cenário Urbano, com mais recuperações em todos os cenários testados. Os melhores resultados no cenário de Rodovia foram obtidos com a abordagem GCFML, que resultou em menores taxas de recuperação e melhores taxas de sucesso. Apesar de o cenário Urbano possuir uma topologia mais complexa, os resultados de *offloading* usando o GCF e o GCFML se destacaram, gerando as menores taxas de recuperação e as maiores taxas de sucesso nos cenários de média e alta densidade veicular.

3.5 Considerações Finais

Os resultados experimentais alcançados em *datasets* simulados demonstram que as abordagens de ML baseadas em regressão, como SVR, são promissoras para estimar o TVE entre nós em VANETs. A estratégia melhorou a eficiência da predição. Além disso, identificar variáveis que influenciam o TVE provou ser uma importante contribuição para futuras pesquisas.

Como observado, no cenário de rodovia, a solução utilizando ML apresentou ganhos reais em relação à abordagem tradicional. Porém, no cenário urbano tais condições não foram repetidas. A fim de melhorar a eficiência no cenário urbano, já que é um ambiente dominante em cenários veiculares e com isso demanda melhor eficiência em operações de comunicação, uma nova hipótese será avaliada nos capítulos seguintes.

Este Capítulo considerou o uso de apenas um modelo de ML nos testes de *offloading* computacional. Os *datasets* utilizados no processo de treinamento dos modelos foram gerados com base nos cenários urbano e de rodovia. Logo, apenas um mesmo modelo foi utilizado em todas as densidades avaliadas por cenário. O Capítulo 5 aborda uma nova hipótese para geração dos dados e treinamento de modelos. Além disso, uma arquitetura foi implementada com recursos que permitem a atualização e troca de modelos de ML, de acordo com densidade atual de tráfego, em diferentes áreas e momentos da simulação.

4 PROCESSO DE DECISÃO DE *OFFLOADING* EM *VEHICULAR FOG COMPUTING* COM PEDESTRE

Este Capítulo aborda o OE2 desta dissertação, com foco na demonstração da implementação de um algoritmo de *offloading* em VFC ao nível de simulação utilizando um dispositivo móvel de pedestre como cliente. O algoritmo de decisão de *offloading* proposto é avaliado sob diferentes contextos a fim de validá-lo com o intuito de utilizar a mesma abordagem na proposta final.

4.1 Contextualização

Na comunicação P2X/V2X, os pedestres usam um *smartphone* para estabelecer comunicação por rede *Wi-Fi* ou via rede celular (por exemplo, LTE ou 5G) para se comunicarem com diversos componentes, como a nuvem, servidores de borda e veículos (KHAN *et al.*, 2022). No caso da comunicação celular, os usuários participam de uma rede maior mediante um sistema de comunicação móvel. Seguindo a evolução atual de LTE para 5G, a tecnologia traz muitas vantagens para aplicações de segurança para *Usuário Vulnerável de Estrada* (VRU) — e também é considerada para comunicação V2X (ZOGHLAMI *et al.*, 2023).

No entanto, um dos maiores problemas enfrentados pelos *smartphones* é a limitação de recursos. Quanto mais informações de contexto e algoritmos complexos são usados, maior é a carga de computação nos dispositivos (NGUYEN *et al.*, 2019). Para lidar com tal problema nos dispositivos móveis, a técnica de *offloading* computacional é utilizada para facilitar a execução de aplicações complexas computacionalmente, através da distribuição de partes da aplicação a outros dispositivos ou infraestruturas, sendo bastante empregada em comunicações veiculares (SOUZA *et al.*, 2020).

VFC emprega veículos como infraestrutura para fazer um melhor uso da comunicação veicular e seus recursos computacionais (KESHARI *et al.*, 2022). A tecnologia VFC é introduzida para apoiar o ITS, oferecendo recursos computacionais através de mais componentes disponibilizados como servidores e assim melhorar o processamento de aplicações, diminuindo a sobrecarga de processamento e as taxas de atraso na rede no *offloading* computacional (SHI *et al.*, 2020) (WU *et al.*, 2023). A integração de pedestres a uma arquitetura VFC tem potencial de melhorar suas capacidades de computação e diminuir a sobrecarga de rede no processo de *offloading* computacional, com poucos trabalhos na literatura desenvolvendo soluções que tratem do *offloading* de tarefas nesse contexto.

Este Capítulo tem em vista integrar VRUs, precisamente pedestres, à tecnologia VFC, criando cenários com diferentes componentes de comunicação, que incluem veículos, pedestres, servidores de borda e servidores de nuvem. O trabalho introduz um algoritmo de decisão para o processo de *offloading* computacional em VFC com o pedestre como origem, e avalia o desempenho do algoritmo proposto em cenários com suporte a três diferentes categorias de componentes e comparando também com o algoritmo clássico *First In, First Out* (FIFO).

4.2 Trabalhos Relacionados

Em (WANG *et al.*, 2018), é apresentado um *offloading* computacional em que usuários de dispositivos buscam veículos próximos para processar suas aplicações. O objetivo é utilizar efetivamente os recursos computacionais disponíveis nos veículos inteligentes ao redor dos pedestres, transformando tais veículos em *cloudlets*. Os autores ainda propõem um esquema de retransmissão entre os *cloudlets*, ao invés de executar todas as tarefas em um único *cloudlet*. Além disso, foi desenvolvido um algoritmo para apoiar o processo de decisão com base no melhor tempo de troca entre *cloudlets*. O tempo de troca entre *cloudlets* é o tempo para mudar para um novo *cloudlet* e selecionar o melhor como o próximo local de execução ou retornar a execução de volta ao veículo requisitante de *offloading*. Os autores utilizam apenas veículos como componentes do *offloading*, consequentemente seu algoritmo de decisão considera apenas essa entidade.

No trabalho de (LIN *et al.*, 2019b), os autores implementam uma arquitetura federada com o uso de servidores de borda e servidores *Fog* (veículos). Os autores desenvolveram um algoritmo guloso iterativo com o intuito de facilitar o processo de decisão de *offloading*, com base nos recursos de cada componente. O algoritmo verifica a disposição dos recursos e faz um balanceamento de quantas tarefas enviar para cada componente. Por fim, eles analisam a redução de custo proporcionada pela solução. Os autores, porém, não realizam uma medição do impacto do tempo de *offloading* no cenário e também não usam essa arquitetura como provedor de recursos para pedestres.

Um esquema de recuperação de dados para *offloading* computacional em VEC é proposto por Boukerche e Soto (2020). Os autores apresentam um modelo de sistema de *offloading* computacional que utiliza a posição e direção do movimento do usuário no momento do *offloading* computacional. Tal processo é implementado a fim de estimar a posição futura do veículo a assim realizar uma recuperação de dados mais confiável e eficiente. O trabalho, porém,

não envolve o compartilhamento de tarefas entre pedestres e veículos no processo de *offloading*, somente entre veículo/pedestre para RSU.

Alguns trabalhos implementam esquemas de comunicação, com foco no veículo como origem, como em (NGUYEN *et al.*, 2020), em que os autores propõem uma abordagem adaptativa para *smartphones*, que podem ser o destino para o *offloading* de tarefas vindas dos veículos. Os autores também permitem a execução local de tarefas, caso não existam opções mais vantajosas. Além disso, o consumo de energia e a latência de cada esquema são investigados. O trabalho utiliza também um servidor de borda, porém os autores não realizam medições da taxa de *offloading* bem-sucedido ou taxa de execução local no processamento das tarefas.

No trabalho de (GONÇALVES *et al.*, 2019), é implementado um esquema de avaliação da predição de mobilidade de usuários de dispositivos em *Fog Computing* no contexto de Cidade Inteligente. O objetivo do trabalho é analisar o comportamento do processo de migração de aplicações em diferentes cenários de um ambiente de *Fog Computing*. Contudo, não é considerado no trabalho uma aplicação de *offloading* computacional e uma posterior avaliação de desempenho deste processo nos cenários propostos.

Os autores Almeida *et al.* (2020) investigam a capacidade do *Wi-Fi Direct* oferecer conectividade no ambiente veicular. Com base nas trocas de informações entre um veículo e um pedestre em uma arquitetura V2P, são analisados os resultados de medições reais usando *smartphones* comerciais. Os autores também não consideram cenários de *offloading* computacional em suas avaliações.

O presente trabalho implementa um algoritmo de decisão de *offloading* em VFC, integrado a dispositivos de pedestres. Os trabalhos apresentados utilizam os pedestres em comunicação com servidores remotos como Borda ou veículos, mas nenhum trabalho utiliza VFC como servidor de recursos para pedestres. Os trabalhos que utilizam VFC têm como foco os veículos como origem do *offloading* e não pedestres. Além disso, as métricas analisadas no presente trabalho não foram abordadas em conjunto nos trabalhos citados.

4.3 Etapas para o *Offloading* de Tarefas do Pedestre em VFC

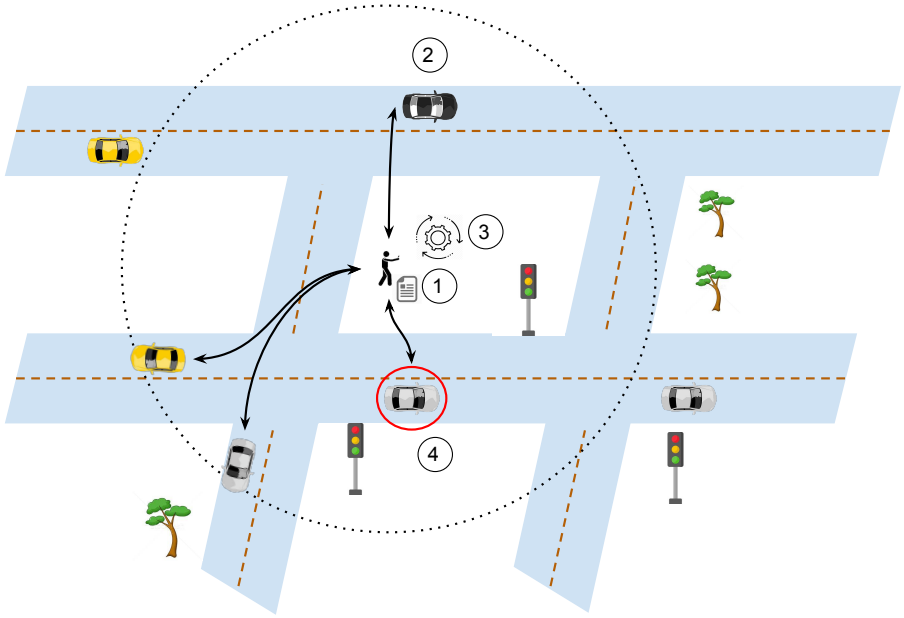
Esta seção apresenta as etapas do OE2 desta dissertação, dividida em três partes. A primeira parte engloba o processo de descoberta de veículo a ser integrado à comunicação VFC. Na segunda parte, é apresentado o algoritmo de tomada de decisão para o processo de *offloading* em P2X. A última parte trata da configuração do ambiente de rede veicular para suportar o P2X

e possibilitar o *offloading* computacional.

4.3.1 Processo de Descoberta de Veículo

Para que veículos e pedestres possam se comunicar e, consequentemente, realizar o processo de *offloading*, a primeira etapa é identificar o veículo mais apropriado para receber tarefas de um pedestre. Para isso, o processo de descoberta entre veículos próximos, ilustrado na Figura 22 é essencial.

Figura 22 – Processo de descoberta de veículos.



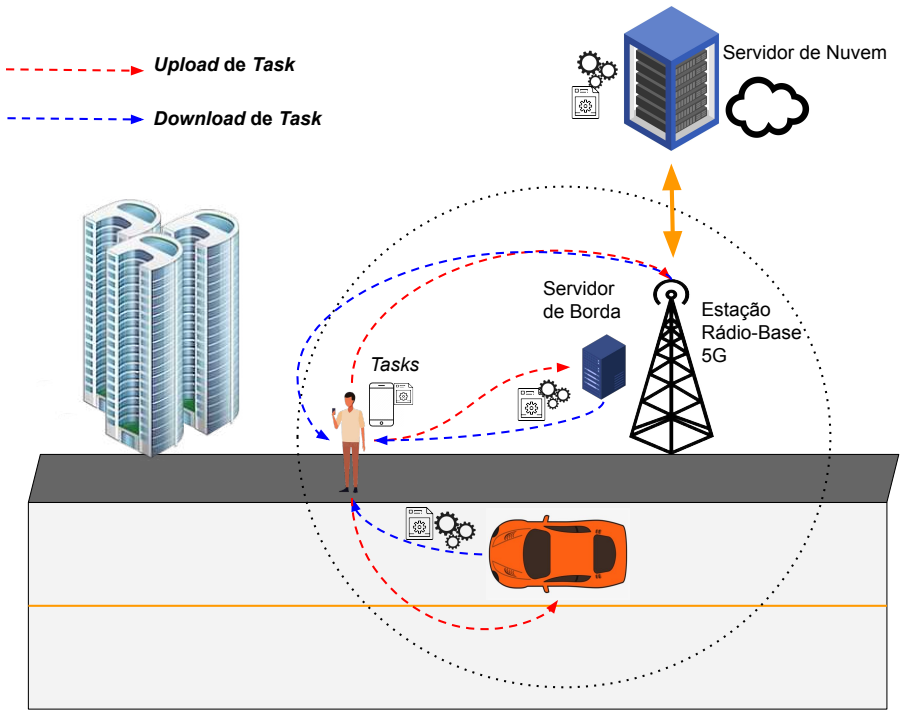
Fonte: elaborada pelo autor.

O pedestre envia requisições *broadcast* diretamente a todos os veículos no seu raio de alcance (Figura 22 — etapa 1). Os veículos próximos respondem à requisição com algumas informações importantes para o processo de escolha, tais como velocidade, posição e percentual de uso da CPU do veículo (Figura 22 — etapa 2). Inicia-se então o processo de decisão (Figura 22 — etapa 3) e o primeiro passo é calcular a estimativa do tempo de vida do enlace entre o pedestre(i) e um veículo(j), com base no cálculo LLT(*Link Lifetime*) (SOUZA *et al.*, 2013). Os veículos que tiveram uma predição de tempo $LLT_{ij} = 0$ foram descartados do processo de escolha.

Em seguida, foi coletado o percentual de uso da *CPU* de cada carro, descartando aqueles veículos com o uso total da *CPU*. Por último, o veículo que possui mais CPU livre e um maior tempo de vida do enlace estimado, nessa ordem, é escolhido. Uma vez escolhido o

melhor veículo do ambiente (Figura 22 - etapa 4), o mesmo é inserido em uma rede VFC, como servidor *fog*. Para finalizar a rede VFC, os outros componentes foram alocadas na mesma, como demonstrado na Figura 23, a estação base mais próxima do veículo foi identificada e instanciada como um servidor de borda.

Figura 23 – Topologia de comunicação P2X.



Fonte: elaborada pelo autor.

A comunicação de acesso entre veículo, pedestre e borda é realizada via rede celular, através do acesso 5G, como explicado na subseção 4.3.3. Com isso, diversas estações rádio-base foram espalhadas no cenário. Por fim, para realizar a comunicação com o servidor na nuvem, foi simulado um *link* entre a estação rádio-base, próxima dos componentes *fog* e borda, e a nuvem. Com todos os componentes necessários para o funcionamento da VFC, ocorre o *offloading* de *tasks* (tarefas) do cliente (pedestres) para os servidores e o posterior recebimento das tarefas processadas nos servidores para o cliente.

4.3.2 Algoritmo de Decisão de Offloading

Com todos os componentes identificados, é implementado então o processo de decisão do *offloading* computacional, que está definido no Algoritmo 2. Para iniciar o processo de decisão, o algoritmo recebe as informações de todos os componentes da comunicação, sendo *Edge*, *Vehicle* e *Cloud*. Essas informações são recebidas por meio de respostas à soli-

citação de *offloading* realizada pelo pedestre à rede VFC, sendo as respostas armazenadas em $\{DiscoveryPacket\}$. O $\{DiscoveryPacket\}$ contém as informações de cada pacote de resposta do componente, que será chamado servidor remoto.

Algoritmo 2: Algoritmo de Decisão

```
   Input: {DiscoveryPacket}, Edge, Vehicle, Cloud, numberOfTasks
   Output: {Tasks}
 1 providers ← {}; amountTasksSelected ← 0;
 2 Tasks ← {}; ST ← {}
 3 foreach packet in {DiscoveryPacket} do
 4 |   {providers} ← getComponent(packet);
 5 end
 6 {providers} ← Sort({providers});
 7 foreach provider in {providers} do
 8 |   if amountTasksSelected < numberOfTasks then
 9 |   |   cpu ← getCPU(provider);
10 |   |   taskToEachProvider ←
   |   |    balacerTask(provider, cpu, numberOfTasks, amountTasksSelected);
11 |   |   ST ← {ST} ∪ taskToEachProvider;
12 |   |   amountTasksSelected ← amountTasksSelected + taskToEachProvider;
13 |   end
14 end
15 if amountTasksSelected < numberOfTasks then
16 |   localTasks ← numberOfTasks − amountTasksSelected;
17 |   task ← localExecution(localTasks);
18 |   Tasks ← {Tasks} ∪ task;
19 end
20 Tasks ← sendTasks(providers, ST);
21 return {Tasks};
```

O conjunto $\{providers\}$ armazena os servidores remotos encontrados pelo pedestre para realizar o *offloading*. Já o conjunto $\{Tasks\}$ armazena as tarefas processadas pelos servidores remotos ou pelo próprio dispositivo do pedestre. O conjunto $\{ST\}$ *(Selected Tasks)* é responsável por armazenar as tarefas selecionadas para os servidores remotos pelo Algoritmo 3.

O processo então começa com o algoritmo realizando uma verificação de todas as respostas recebidas por meio de uma iteração (linha 3). É verificado cada pacote por meio da função *getComponent*, que identifica de qual componente veio a requisição, retornando o componente e suas informações para $\{providers\}$ (linhas 4). Os componentes *providers* são ordenados pela função *Sort* (linha 5), sendo os componentes mais prioritários aqueles que receberão mais tarefas. Os componentes são ordenados pelo seu tipo e percentual de CPU

livre, onde a prioridade dos tipos é definida na seguinte sequência: borda, veículo e Nuvem. Essa prioridade é devido aos custos estimados que levariam para os respectivos componentes realizarem o processamento das tarefas, sendo considerado a ordem do menos custoso ao mais custoso. Com a verificação terminada, o próximo passo é realizar o balanceamento de tarefas, o processo em que se decide quantas tarefas serão processadas em cada componente.

Iterativamente, é verificado se cada servidor remoto *provider* possui tarefas a serem processadas (linhas 6-7). A variável *amountTasksSelected* armazena a quantidade de tarefas já atribuídas a algum servidor, com o valor inicial começando em zero. Para cada componente pertencente ao *provider* a função *balancerTask* é chamada (linha 9), tendo como retorno o número de tarefas que o componente receberá. Essa função recebe o percentual de uso da CPU do componente, consultado pela função *getCPU*(linha 8), o próprio componente, o número total de tarefas a serem processadas (*numberOfTasks*) e a quantidade de tarefas já atribuídas a outros componentes (*amountTasksSelected*).

A função *balancerTask* é definida no Algoritmo 3. A função verifica se o uso de CPU pelo *provider* é menor que o percentual necessário para processar uma tarefa. Se for menor, então o *provider* consegue processar a tarefa, logo a variável (*tasksToProvider*) é incrementada. A variável $RA_{provider}$ (*Resource Available*) atualiza o seu novo estado com mais uma tarefa, tendo um maior percentual de CPU ocupada. Por fim, se não houver mais recursos, o valor de *tasksToProvider* é retornado.

Algoritmo 3: balancerTask

```
   Input: provider, cpu, numberOfTasks,amountTasksSelected
   Output: tasksToProvider
1  tasksToProvider ← 0
2  for i in range(amountTasksSelected,numberOfTasks) do
3      if cpu < RA_provider then
4          tasksToProvider ← tasksToProvider + 1;
           RA_provider ← RA_provider ∪ tasksToProvider;
5      end
6  end
7  return tasksToProvider;
```

De volta ao Algoritmo 2, o retorno da função *balancerTask()* é armazenado em $\{ST\}$ na mesma posição onde o componente foi inserido no conjunto $\{providers\}$ (linha 10). A quantidade de tarefas selecionadas é incrementada em *amountTasksSelected* (linha 11). O processo retorna para verificar se as tarefas selecionadas já atingiram o número total de tarefas

que o pedestre solicitou para serem processadas (*numberOfTasks*), encerrando o balanceamento assim que esse valor é alcançado (linha 7).

Caso o número de *amountTasksSelected* não tenha atingido o número de tarefas solicitadas pelo pedestre (*numberOfTasks*) após o laço de iteração encerrar, é realizada uma última verificação de modo a descobrir se sobraram tarefas a serem processadas (linha 12). Caso haja tarefas, é realizada uma operação para coletar a quantidade de tarefas remanescentes executá-las localmente no dispositivo do pedestre, através da função *localExecution()* (linha 14). O resultado é armazenado no conjunto $\{Tasks\}$, responsável por receber as tarefas processadas (linha 15).

Ademais, as tarefas são enviadas para serem executadas em outros componentes através da função *sendTasks* (linha 16). A função *sendTasks* recebe o conjunto de servidores e o conjunto $\{ST\}$ e envia para cada *provider* do conjunto de $\{providers\}$ as respectivas tarefas armazenadas no conjunto $\{ST\}$. Na função *sendTasks*, cada *provider* recebe as tarefas atribuídas a ele para processamento, com essa associação sendo realizada pela posição de *index* dos conjuntos $\{Providers\}$ e $\{ST\}$. Os resultados são inseridos no conjunto $\{Tasks\}$ e retornados ao requisitante (i.e., o pedestre).

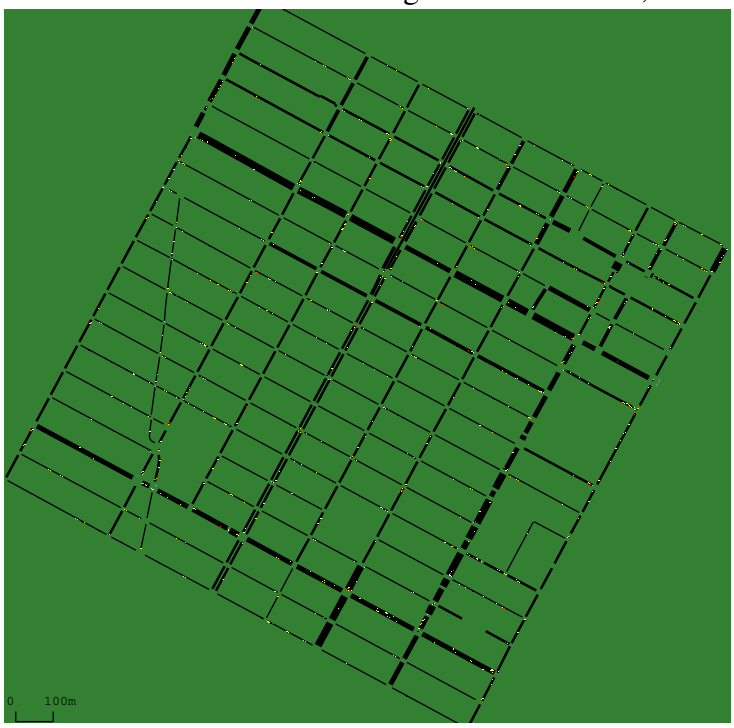
4.3.3 Arquitetura de Comunicação

Nesta parte da dissertação, a ferramenta SUMO e o simulador de redes ns-3 também foram utilizados. Um cenário urbano foi escolhido para ser o ambiente de tráfego da simulação, gerado com o SUMO, a partir de uma parte da região de Manhattan, na cidade de Nova York, EUA, equivalente a 2 km², como mostra a Figura 24 (SOUZA *et al.*, 2020).

Além do mapa da região, o SUMO gera um arquivo de mobilidade, com veículos, pedestres e rotas, que o ns-3 importa, e com isso os veículos e pedestres são transformados em nós de rede. Posteriormente, estações-base são instaladas no cenário para permitir a comunicação entre os nós, realizada pelo módulo *mmWave* do ns-3, responsável por implementar a comunicação de acesso 5G. A tecnologia *mmWave* permite a configuração das camadas físicas e de enlace do 5G nos dispositivos utilizados pelos componentes (i.e., pedestre, veículo e servidor de borda). As configurações utilizadas estão resumidas na Tabela 6.

Para possibilitar tal comunicação, uma estação-base nativa da comunicação 5G, denominada *gNodeB* (gNB), próxima a um pedestre é escolhida e um servidor de borda é instanciado próximo à gNB. Foram configuradas 16 estações-base no cenário, de modo a garantir

Figura 24 – Cenário utilizado neste trabalho — região de Manhattan, Nova York, EUA.



Fonte: SUMO.

Tabela 6 – Principais Configurações da Simulação.

Parâmetro	**Valor**
Acesso 5G	
Datarate	450 Mbps
Modelo de Rádio Propagação	5G *mmWave systems* (MEZZAVILLA *et al.*, 2018)
Demais Configurações	
Velocidade máxima dos veículos	16 m/s
Velocidade dos pedestres	1 m/s
Protocolo de Descoberta de Veículo	UDP
Protocolo de Transferência de Tarefas	TCP
Tempo de Simulação	160 s

Fonte: elaborada pelo autor.

cobertura total da rede 5G em todo o cenário.

4.4 Experimentos

De modo a validar o algoritmo proposto, é realizada uma série de experimentos para avaliar métricas de desempenho do processo de *offloading* computacional. As métricas abordadas são: taxa de sucesso de *offloading*, taxa de execuções locais, taxa de falhas de *offloading* e tempo total de *offloading* computacional.

A abordagem é avaliada em quatro diferentes contextos, com o Algoritmo 2 sendo utilizado e adaptado para cada contexto. No primeiro contexto, o algoritmo considera apenas

o servidor de borda interligado com o pedestre, cenário chamado *Local e Borda* (LE). Depois, uma abordagem chamada *Local, Borda e Nuvem* (LEC) é avaliada, onde o servidor de borda e o servidor de nuvem estão disponíveis para o pedestre. Uma abordagem chamada *Local, Borda e Veículo* (LEV) onde servidor de borda e servidor de névoa(veículo) são disponibilizados também é considerada. Por fim, num contexto chamado *Local, Borda, Nuvem e Veículo* (LECV), é analisado o algoritmo com todos os componentes da VFC, ou seja, com veículos e servidores de borda e nuvem interligados ao pedestre. Esses quatro contextos são avaliados e comparados ainda com um algoritmo de decisão aleatória de *offloading*, chamado FIFO. No FIFO, assim como no LECV, todos os componentes estão disponíveis, mas a alocação de tarefas é realizada de forma aleatória.

Além dos algoritmos, outros dois fatores foram utilizados nos experimentos, a densidade de veículos/pedestres e o tamanho das tarefas a serem executadas. Com base em (LI *et al.*, 2019) e (SOUZA *et al.*, 2020), foram definidos três diferentes níveis para a densidade: densidade baixa, com 50 veículos e 50 pedestres; densidade média, com 276 veículos e 276 pedestres e, por fim, densidade alta, com 609 veículos e 510 pedestres. Com relação ao tamanho da tarefa, foram utilizados dois níveis de tamanho de imagem: 558 KB e 1,2 MB. Esses valores foram escolhidos visando avaliar imagens com resoluções distintas em *megapixels* nos experimentos conduzidos por Souza *et al.* (2020). Os fatores e seus respectivos níveis estão resumidos na Tabela 7, que também demonstra as combinações de fatores entre densidade de tráfego e tamanho da tarefa.

Tabela 7 – Combinação dos Fatores de Densidade e Tamanho da Tarefa.

Número de Veículos	**Número de Pedestres**	**Tamanho da Tarefa**	**Identificador da Combinação de Fatores**
50	50	558 KB	B1
50	50	1,2 MB	B2
276	276	558 KB	M1
276	276	1,2 MB	M2
609	510	558 KB	D1
609	510	1,2 MB	D2

Fonte: elaborada pelo autor.

Com os parâmetros, fatores e níveis definidos, os experimentos foram executados em uma máquina com o ns-3 versão 3.29 instalado, com o módulo *mmWave* e com as seguintes configurações: processador *Xeon* E5645 @2.40GHz com 24 núcleos e 32 GB de *RAM*.

Na simulação, as seguintes variáveis são aleatórias: seleção do cliente, que será um

pedestre e o tempo de início da simulação. Em cada rodada de execução da simulação essas duas variáveis aleatórias mudam. Para cada combinação de fatores (algoritmo, densidade e tamanho da tarefa), foram realizadas 80 execuções da simulação com o parâmetro quantidade de tarefas solicitadas pelo pedestre fixado em 8. Foi utilizada a distribuição uniforme para gerar as variáveis aleatórias, na qual permite que os valores sejam uniformemente distribuídos ao longo de um intervalo semiaberto $[min, max)$. Onde dentro desse intervalo, o valor é escolhido aleatoriamente. A distribuição uniforme foi utilizada devido ao caráter determinístico do intervalo, já que as variáveis aleatórias utilizadas neste experimento, estavam em escopos limitados (quantidade de pedestres e tempo de simulação).

Por fim, foi definido o tempo de 160 segundos de simulação para a realização do *offloading* computacional. Porém, outro critério de parada foi configurado que foi o encerramento da simulação, assim que todas as *tasks* tenham sido executadas.

Para realizar a medição da métrica do tempo de *offloading* computacional, são coletados o tempo desde o envio da tarefa pelo pedestre até chegar ao componente servidor (*upload*), o tempo de processamento da tarefa e o tempo de envio da tarefa processada pelo servidor remoto ao pedestre (*download*). O compartilhamento de tarefas dentro do ns-3 ocorre por meio de funções disponibilizadas pelo simulador, sendo feito por geradores de *stream* de pacotes criados. O valor de *upload* é dado pela variável UPL e o valor de *download* é dado pela variável DWL, ambos os valores medidos em segundos, conforme as equações a seguir:

$$UPL_{i,j} = \frac{TT}{DR} + \frac{D_{ij}}{VPM} + DLF \qquad DWL_{i,j} = \frac{TP}{DR} + \frac{D_{ij}}{VPM} + DLF \tag{4.1}$$

onde TT refere-se ao tamanho das tarefas selecionadas para envio, DR é o *datarate*, $D_{i,j}$ é a distância entre o pedestre (i) e o componente servidor (j), VPM refere-se à velocidade de propagação do meio e DLF é o atraso na fila de transmissão ou recepção do envio/recebimento do pacote. No cálculo do *download*, TP é o tamanho do resultado do processamento das tarefas recebidas. A seguir é demonstrado o cálculo do tempo de processamento das tarefas em segundos:

$$Proc_j = \frac{NT * C_t}{RA} \tag{4.2}$$

onde NT refere-se ao número total de tarefas recebidas pelo servidor remoto, C_t é a quantidade de CPU necessária para processar uma tarefa e RA é a quantidade de recurso

disponível no servidor para realizar o processamento (XIAO *et al.*, 2019). Por fim, é calculado o *Tempo Final de Offloading* (TFO), somando os tempos anteriormente calculados, como demonstrado na Equação 4.3.

$$TFO = UPL_{i,j} + Proc_j + DWL_{i,j} \tag{4.3}$$

Dado que o processo de *offloading* foi finalizado, a quantidade de tarefas processadas em uma determinada execução é contabilizada e armazenada em um arquivo de *log*, com a identificação da *seed* configurada, para permitir uma comparação pareada.

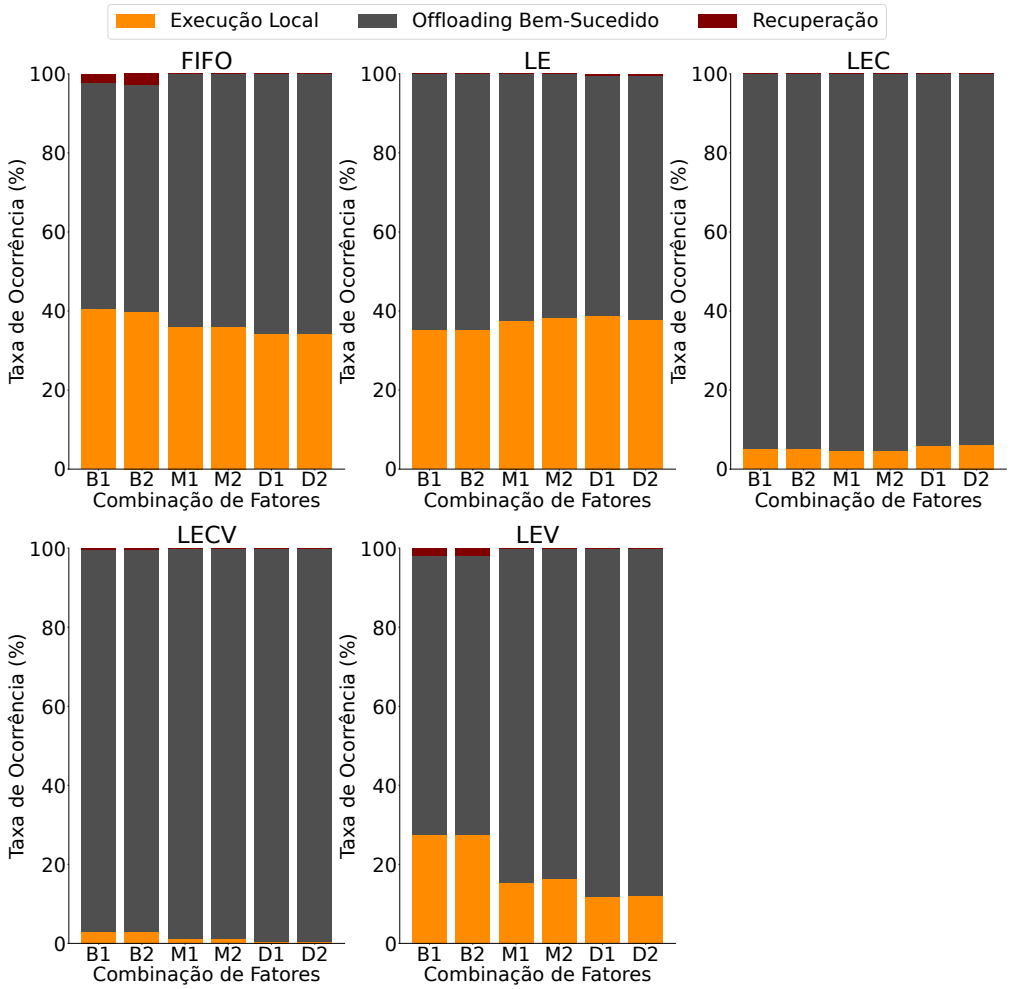
O resultado do processo de *offloading*, além da métrica TFO, permite computar outras três métricas relacionadas à eficiência do algoritmo, ao analisar o impacto da tomada de decisão no processo de *offloading*. A primeira é o percentual de tarefas processadas com sucesso e retornadas pelos servidores. A segunda refere-se ao percentual de tarefas executadas localmente no dispositivo do pedestre. E a última é o percentual de recuperações, que se refere à execução local de uma cópia da tarefa enviada a um servidor e que não foi retornada, devido a uma perda de conexão entre cliente e servidor.

4.4.1 Eficiência do offloading

A Figura 25 apresenta as métricas relacionadas à eficiência da tomada de decisão de *offloading*. Como pode-se observar, o algoritmo LECV é o que apresenta a maior taxa de *offloading* realizado, o que é esperado já que nessa abordagem há todos os componentes da rede veicular VFC no processo de decisão. Nota-se também que o algoritmo LEC e o LEV, apesar de terem menos componentes disponíveis, são melhores que o algoritmo FIFO. Um reflexo da eficiência na tomada de decisão é a baixa taxa de execuções locais, já que o pedestre envia mais tarefas para outros dispositivos e as escolhas parecem apropriadas, diminuindo as recuperações. Nesse sentido, o LECV apresentou uma diminuição de mais de 30% na taxa de execuções locais, garantindo assim que mais tarefas pudessem ser processadas remotamente. Apesar de não utilizar veículos no processo de decisão, o algoritmo LEC é também uma opção melhor do que o algoritmo FIFO.

O algoritmo FIFO apresenta altas taxas de recuperação, reflexos do processo de aleatoriedade de sua escolha, dado que componentes inaptos podem ser escolhidos para o processamento de tarefas. O algoritmo LE também apresenta altas taxas de recuperação, o que

Figura 25 – Eficiência do *offloading* com cada algoritmo.



Fonte: elaborada pelo autor.

pode ser justificado pela sobrecarga gerada no componente, já que o servidor de borda é o único componente disponível para o processamento de tarefas.

Nas execuções em cenários de baixa densidade de tráfego (B1 e B2) dos algoritmos FIFO e LEV, há uma maior taxa de execuções locais e recuperações devido à maior possibilidade de não haver veículos próximos ao pedestre. Em relação ao LEV, tal resultado não é percebido em cenários com maiores densidades de tráfego, visto que haviam mais veículos disponíveis no cenário. Com isso, nos cenários M1, M2, D1 e D2, houve uma queda no número de execuções locais do LEV em relação ao FIFO.

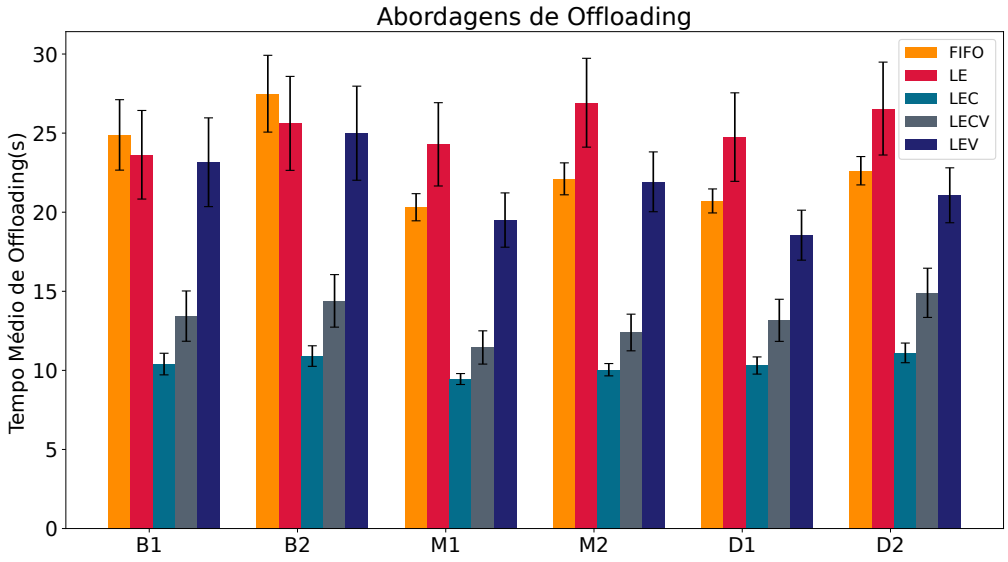
É importante ressaltar que o risco de utilizar veículos é maior, pois se aumenta a probabilidade de cliente e servidor perderem a comunicação entre si e, com isso, o pedestre não receber o resultado das tarefas compartilhadas devido às suas características de mobilidade.

Entretanto, os resultados mostram que a taxa de recuperação não é grande, o que indica que a adoção da técnica de *offloading* nessa categoria de cenário é segura.

4.4.2 Tempo de Offloading

A Figura 26 apresenta a média do tempo de *offloading* com intervalo de confiança de 95%. Os resultados demonstram que a presença de mais opções para o *offloading*, como nos algoritmos LEC, LEV e LECV, ajudam a diminuir o tempo de duração do *offloading*. Um tempo de duração menor indica um processamento de tarefas melhor e, consequentemente, menor latência e melhor experiência para o usuário das aplicações. A abordagem de decisão de *offloading* proposta neste trabalho é melhor em relação à abordagem aleatória, dado que os requisitos de recursos dos dispositivos (CPU) são considerados para ponderar as decisões de *offloading*, resultando em processamentos mais rápidos.

Figura 26 – Tempo médio de *offloading* com cada Algoritmo.



Fonte: elaborada pelo autor.

Os resultados do algoritmo LE mostram que a utilização de apenas um servidor de borda (com poder de processamento limitado) pode ser insuficiente para melhorar o tempo de execução da aplicação, principalmente por estar sujeito a sobrecargas de requisições, caso seja o único componente a tratar *offloading* computacional.

Foram realizados testes estatísticos adicionais para melhor avaliar os resultados com sobreposição de intervalos de confiança. Inicialmente, testes de *Shapiro-Wilk* foram realizados,

onde foi constatado que os dados não são normalmente distribuídos. Então, foram realizadas comparações pareadas entre os resultados usando os testes de *Wilcoxon*. Nos testes, a hipótese nula é que duas soluções comparadas possuem o mesmo desempenho. Se $p < 0,05$, a hipótese nula é rejeitada, sendo considerado que as soluções comparadas possuem desempenhos diferentes estatisticamente. As Tabelas 8 e 9 apresentam os valores de *p-value* para os testes executados.

Tabela 8 – Valores de *p-value* para testes de *Wilcoxon* pareados de FIFO e LEV.

	B1	**B2**	**M1**	**M2**	**D1**	**D2**
LEV x FIFO	0,52	0,335	0,344	0,16	0,82	0,983

Fonte: elaborada pelo autor.

Tabela 9 – Valores de *p-value* para testes de *Wilcoxon* pareados de FIFO, LE e LEV.

	B1	**B2**	**D2**
LE x FIFO	0,401	0,275	0,004
LE x LEV	0,919	0,909	0,016

Fonte: elaborada pelo autor.

Com base nos resultados estatísticos, apenas os cenários de alta densidade com o tamanho da tarefa de 1,2 MB (D2) apresentam diferenças estatísticas entre os resultados comparados (*p-value* < 0,05), apesar dos intervalos de confiança se sobreporem. Com o restante dos resultados de tempo médio sendo iguais entre FIFO, LE e LEV. O LECV demonstrou uma grande evolução de desempenho em relação ao FIFO, o que é uma consequência do processo de escolha mais eficaz, já que ambas as abordagens são as únicas que dispõe de todos os servidores da VFC.

Todas as soluções mostraram-se melhores opções em relação a executar todas as tarefas localmente no dispositivo do pedestre. O tempo de execução local observado foi: 56,5 segundos para tarefas de 558 KB e 59 segundos para tarefas de 1,2 MB. Mesmo nos piores casos, houve uma diminuição de até 30 segundos no tempo médio de *offloading*. O LECV demonstra uma redução de tempo de até 77% em relação à execução totalmente local, comprovando a importância do *offloading* e do uso de um VFC na execução de aplicações em redes veiculares.

4.5 Considerações Finais

Este Capítulo abordou o OE2 desta dissertação e investigou como melhorar os avanços de comunicação em cenário de redes veiculares, especificamente VFC. Foi implementada uma rede VFC, trazendo mais dispositivos para o ambiente com a inserção de pedestres, que

atuam como clientes do processo de *offloading* computacional. Um algoritmo de decisão de *offloading* foi apresentado e experimentos foram conduzidos para avaliar o desempenho do algoritmo aplicado em três contextos de densidade de tráfego e duas categorias de *workload*. Além disso, uma estratégia de decisão aleatória também foi utilizada nos experimentos.

Como os resultados demonstraram, a abordagem apresentada tem potencial para melhorar o tempo de execução e eficiência do *offloading* computacional em ambientes de VFC com pedestres atuando como clientes. Tais melhorias refletem em menos sobrecargas dos servidores de borda e menos erros na escolha dos destinos do *offloading*, diminuindo retransmissões e latência, e consequentemente melhorando a qualidade das aplicações de ITS ou aplicações pessoais dos pedestres.

5 ARQUITETURA

Este capítulo trata do OE3 do trabalho e apresenta a arquitetura planejada e desenvolvida para aplicar técnicas de ML na predição do TVE em uma rede VFC para *offloading* de tarefas. A arquitetura proposta, utiliza um processo de escolha de modelos de ML, treinados com diferentes conjuntos de dados e de acordo com determinada região na qual o cliente de *offloading* esteja situado, para prever o TVE. Posteriormente o TVE é utilizado no processo de decisão de *offloading* em uma VFC, utilizando os algoritmos implementados no OE2 como parte de um módulo de decisão na arquitetura. Além de descrever a arquitetura detalhadamente e discutir como a mesma se enquadra no contexto de *offloading* computacional em VFC, o presente Capítulo também apresenta uma avaliação de desempenho no processo de *offloading*. Durante essa avaliação, duas abordagens de escolhas dos modelos são analisadas, uma onde são utilizados modelos de ML escolhidos com base em um contexto e outra utilizando modelo de ML baseado em cenário único. O objetivo da avaliação é observar o impacto que as abordagens possuem na eficiência do *offloading* computacional em VFC.

5.1 Contextualização

Como avaliado anteriormente, o uso de modelos únicos para todas as densidades de tráfego avaliadas não trouxe melhorias relevantes no cenário Urbano. Com isso foi levantado uma hipótese em que a escolha do modelo de predição do TVE se daria com base na região na qual o cliente estivesse inserido. Assim, para estabelecer essa hipótese, foi necessário um sistema para adaptar-se e trocar o modelo de predição quando necessário. O uso de sistemas com base em contexto é algo que vem sendo implementado corriqueiramente em *offloading* computacional. Os sistemas de comunicação reais são em sua maioria multiusuários e estão localizados em diferentes locais e condições, assim, a decisão de *offloading* de tarefas deve ser feita com conhecimento do contexto e das condições existentes (FARAHBAKHSH *et al.*, 2021).

Trabalhos que buscam investigar o uso de dados de contexto para apoiar processos de tomada de decisão em *offloading* computacional com VFC são escassos. Para o melhor de nosso conhecimento, não há na literatura trabalhos que utilizam processo de predição do TVE a partir de modelos de ML para apoiar *offloading* computacional em VFC conforme a região que o cliente está inserido.

Portanto, neste Capítulo é demonstrado todo o processo de implementação de uma

arquitetura, a fim de avaliar a nova hipótese proposta. Paralelamente, a arquitetura visa convergir os processos realizados e descritos nos Capítulos 3 e 4 a fim de observar a evolução do algoritmo de decisão de *offloading* desenvolvido, quando o mesmo dispõe da tecnologia de predição do TVE com ML para auxiliá-lo.

5.2 Trabalhos Relacionados

Uma nova estrutura para *offloading* computacional multi-hop (MHCO) em VFC é proposta por Hussain e Beg (2021). A arquitetura proposta está centrada em três aspectos: decisão sobre local de processamento, seleção de nó *fog* servidor (veículo) e compartilhamento de tarefas do nó cliente para o nó *fog*. Essa arquitetura difere da arquitetura proposta neste trabalho por não considerar uma estimativa de TVE no processo decisório. Os autores utilizam a estimativa de latência entre o envio da *task* e o recebimento da mesma. Porém, essa estimativa é realizada por meio de cálculo fixo, sem uso de ML. De maneira semelhante, Hussain *et al.* (2023) propõem um modelo de otimização para investigar quais os melhores servidores em uma rede VFC para instalar nós *fog*. Esse modelo realiza uma escolha, com base no *trade-off* de dois atributos: latência e consumo de energia. Os autores desenvolvem um algoritmo para essa tarefa, porém, não avaliam o impacto de sua solução em cenários de *offloading*, não realizando medições de taxas de sucesso ou falha de *offloading* que o modelo ajudou a melhorar, por exemplo.

Em (CHOPRA *et al.*, 2021), é projetado um sistema colaborativo de *Fog Computing* para melhorar a utilização de recursos utilizada no *offloading* de tarefas na rede veicular. Para isso, é proposta uma arquitetura de computação veicular colaborativa visando atribuir tarefas em servidor local e servidores vizinhos (veículos). O *offloading* de tarefas é executado em um contexto adaptável com base na carga de trabalho dos servidores, a fim de selecionar o veículo mais adequado no ambiente. O trabalho de Rahman *et al.* (2020) também propõe uma arquitetura VFC adaptativa a contexto que permite o *offloading* de tarefas de computação para veículos vizinhos. O mecanismo de decisão incorpora parâmetros sensíveis ao contexto como, velocidade, direção, posição e tráfego do veículo para selecionar a melhor opção possível para *offloading* de tarefas e sua execução. Os principais módulos da arquitetura incluem um módulo de aplicativo, um mecanismo de decisão, módulo de execução local, seleção de dispositivo e módulo de descarregamento, gerenciador de tarefas externo e um módulo de comunicação. O presente trabalho também visa implementar uma arquitetura VFC com um contexto adaptativo, porém com algumas diferenças em relação aos trabalhos mencionados. A primeira diferença

é a agregação da predição do TVE no processo de decisão de *offloading*, resultando em mais módulos de funções na arquitetura, como o módulo de atualização de modelos e treinamento, por exemplo. Logo, a arquitetura torna-se flexível para a adição de mais funcionalidades de ML que não seja predição do TVE. A segunda diferença é o número maior de parâmetros sensíveis ao contexto que o presente trabalho utilizou, demonstrados na Tabela 10.

Diversos dados de contexto são considerados nos trabalhos de Osibo *et al.* (2022) e Shabir *et al.* (2022). No trabalho de Osibo *et al.* (2022) os autores implementam uma arquitetura de *Context-aware Computation Offloading*. Para isso, os autores classificam as tarefas em duas categorias: intensivas e não intensivas. A classificação é realizada por meio de duas informações de contexto, sendo o tempo de execução e o consumo de bateria nos dispositivos móveis. A partir da classificação é então proposto a alocação de tarefas no processo de *offloading*, através de um Algoritmo Genético. Já em (SHABIR *et al.*, 2022), os autores desenvolvem uma arquitetura para *offloading* em *Fog Computing*, onde veículos coletam algumas informações contextuais, tais como, tipo de recurso, custo do sistema, inferência de *offloading* e tempo de residência da tarefa, o qual é um nome dado pelos autores para o tempo que uma tarefa permanece em uma fila. Em (SOUZA *et al.*, 2021), as informações de contexto utilizadas no processo de decisão são tempo de vida do enlace, distâncias entre dispositivos, tempo de transmissão e processamento e se o cliente e o servidor estarão no alcance de comunicação um do outro em um momento específico. O trabalho de Lin *et al.* (2019b) adota uma arquitetura federada de dois níveis: *Edge* e *Vehicular-Fog* (EVF). Quando a borda não tem recursos disponíveis para atender à demanda dos usuários, ela realiza o *offloading* para o veículo, que posteriormente enviarão de volta os resultados para a borda após a conclusão do processamento das tarefas. O trabalho de Lin *et al.* (2019b) foi abordado com mais detalhes no Capítulo 4.

5.3 Arquitetura do Sistema

A Figura 27 demonstra a arquitetura geral do sistema proposto, onde há a integração do processo de predição do TVE na etapa de descoberta e escolha do melhor veículo a ser integrado à arquitetura VFC.

Cada componente executa um conjunto de módulos. Cada carro e cada pedestre, a depender do papel dentro do *offloading*(cliente ou provedor de recurso), possui uma execução diferente de módulos. Para o caso em que um veículo ou pedestre for provedor de recursos, apenas o módulo de execução de tarefas de *offloading* e o sub-módulo de descoberta serão

Tabela 10 – Comparação com os Trabalhos Relacionados.

Trabalho	**Utilizou VFC?**	**Utilizou TVE Como Informação de Contexto?**	**Utilizou Solução Adaptativa?**	**Utilizou ML?**
(HUSSAIN; BEG, 2021)	Sim	Não	Não	Não
(HUSSAIN *et al.*, 2023)	Sim	Não	Não	Não
(CHOPRA *et al.*, 2021)	Sim	Não	Sim (carga de trabalho)	Não
(RAHMAN *et al.*, 2020)	Sim	Não	Sim (velocidade relativa)	Não
(OSIBO *et al.*, 2022)	Não	Não	Sim (tempo de execução e consumo de energia)	Não
(SHABIR *et al.*, 2022)	Não	Não	Sim(carga de trabalho)	Não
(LIN *et al.*, 2019b)	Não	Não	Sim(latência estimada, custo e densidade de tráfego)	Não
(SOUZA *et al.*, 2021)	Não	Sim	Sim (TVE, recurso computacional, carga de trabalho e estimativa de rotas)	Não
Este trabalho	Sim	Sim	Sim (densidade de tráfego, TVE, recurso computacional e carga de trabalho)	Sim

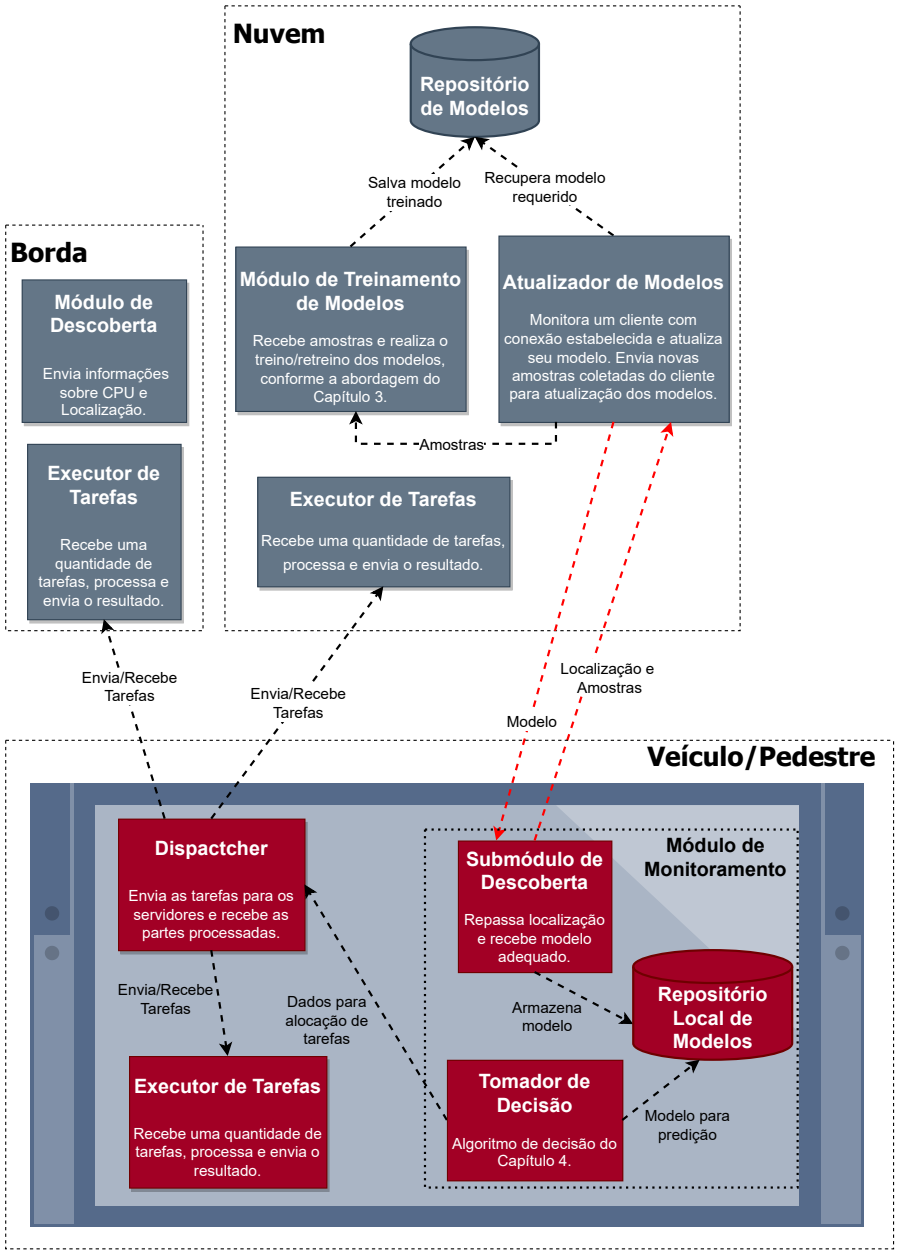
Fonte: elaborada pelo autor.

acionados.

Na Nuvem, há o módulo de treinamento de modelos, responsável por utilizar os dados de tráfego coletados, mencionados na subseção 3.3.4 para treinar os modelos de predição do TVE. É no módulo de treinamento em que estão implementados os procedimentos abordados no Capítulo 3. Após os treinamentos/retreinamentos, os modelos são armazenados no repositório de modelos que os agrupa por densidade e região/localização. Na Nuvem, existe um componente adicional chamado módulo Atualizador de Modelos. Esse módulo desempenha o papel de um *middleware*, estabelecendo uma comunicação entre a nuvem e os veículos/pedestres, coletando dados de densidade e/ou região. Sua função principal é garantir que o modelo apropriado seja enviado aos dispositivos, considerando a região ou a densidade onde a solicitação de *offloading* está ocorrendo. Por fim, a Nuvem também consegue executar as tarefas de *offloading* e retorná-las para o cliente.

Como veículo e pedestre podem ser requisitantes de *offloading* para outros componentes, os mesmos possuem a mesma arquitetura. Há dois módulos essenciais nesses componentes: monitoramento e *dispatcher*. O módulo *dispatcher* é responsável por realizar o *offloading* de tarefas. O módulo de monitoramento é o responsável por descobrir servidores disponíveis e decidir quais deles são os melhores para realizar o *offloading*. O submódulo Tomador de Decisão é o responsável por alocar as tarefas para os servidores encontrados pelo submódulo de descoberta. É no submódulo Tomador de Decisão que fica o algoritmo LECV proposto no Capítulo 4. Cada veículo ou pedestre leva uma cópia do modelo para uma tomada de decisão, e

Figura 27 – Visão Geral da arquitetura proposta.



Fonte: elaborada pelo autor.

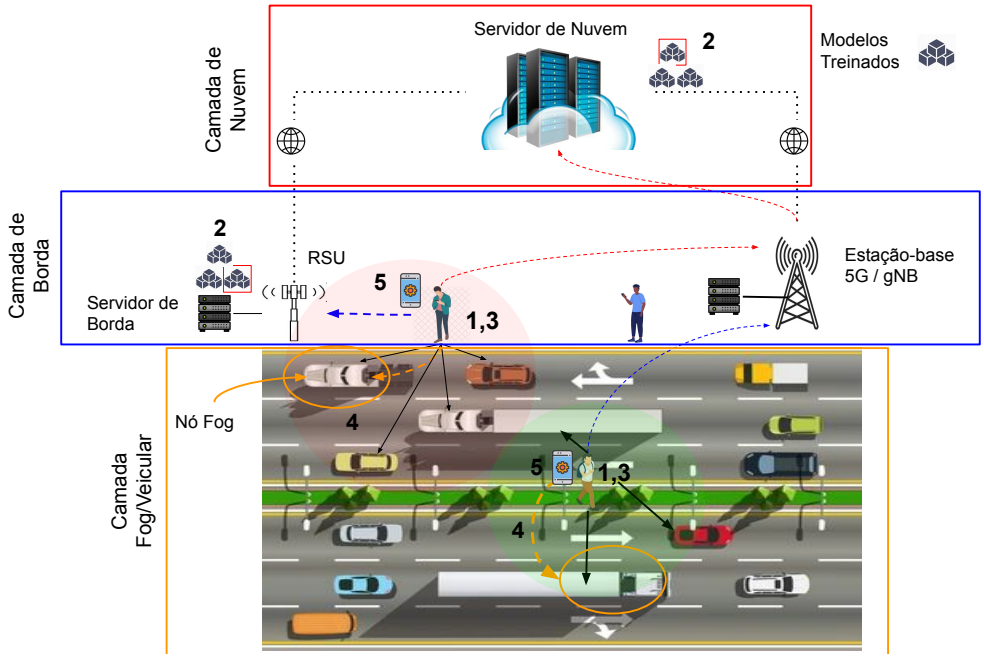
este pode ser atualizado periodicamente por uma comunicação com o módulo de atualização de modelos na Nuvem. Por fim, um veículo ou pedestre pode executar localmente tarefas para o processo de *offloading*.

A borda possui um módulo de execução de tarefas e um módulo de descoberta, responsável por responder às requisições de cliente. O módulo de descoberta envia também dados referentes ao servidor, como o uso de recursos e informações de posição para a realização do processo de predição do TVE.

A arquitetura segue algumas etapas, conforme demonstrado na Figura 28, na qual

mostra uma visão geral do fluxo de etapas dos componentes, sendo: 1) A descoberta de veículos vizinhos com recursos disponíveis; 2) escolha de um modelo treinado conforme o contexto ou tráfego que o pedestre está inserido; 3) cálculo e geração das *features* para predição do valor do tempo de vida do enlace, com informações coletadas a partir dos veículos; 4) o processo de escolha do veículo e integração na VFC; 5) *offloading* de tarefas para os componentes da VFC, a partir do algoritmo implementado anteriormente (LECV). Os detalhes de cada etapa ou módulo são discutidos a seguir.

Figura 28 – Visão geral do processo de *Offloading* em VFC, utilizando o LECV com a predição por contexto.



Fonte: elaborada pelo autor.

5.3.1 Descoberta de Veículos para VFC e Escolha de Modelo

Na Figura 28, o requisitante de *offloading*, representado pelos pedestres em destaque, irá analisar os veículos próximos a ele para identificar possíveis candidatos que possam atuar como componentes de Fog. Nessa etapa é também verificada a quantidade de veículos vizinhos do cliente. Essa informação é importante para a Etapa 2, já que alimenta o módulo Atualizador de Modelo da Nuvem, a fim de que a Nuvem comece a entender o contexto daquele veículo e eventualmente possa decidir trocar o modelo. Isso é importante, por possibilitar que o veículo sempre tenha o modelo específico para aquele contexto.

A etapa 2 só é realizada, caso o servidor não possua o modelo em seu repositório. Os modelos são disponibilizados na Nuvem ou localmente. É importante destacar que a etapa 2 também ocorre quando o cliente repentinamente entra em alguma área da região bastante diferente em termos de tráfego. O componente Nuvem então através do módulo Atualizador de Modelos percebe essa mudança e envia o modelo adequado para aquele contexto. A Nuvem consegue perceber tal mudança por receber informações dos veículos interessados em fazer *offloading*, o que a permite saber as densidades de diferentes regiões e o modelo mais apropriado. Esse mecanismo evita que haja sobrecargas desnecessárias na rede, com atualizações frequentes de modelo, o que impactaria o *offloading*.

5.3.2 Escolha de Veículo Fog para VFC e Offloading de Tarefas

Essa etapa envolve o processo de seleção do veículo para ser um nó *Fog* e posteriormente a escolha dos candidatos a receber *offloading* (demais componentes), através do TVE e dos recursos disponibilizados pelos componentes, que nesse trabalho é o percentual de CPU ocupada de cada nó.

Uma vez que o veículo/pedestre está com o modelo escolhido pela Nuvem como mais adequado ao seu contexto, a etapa 3 acontece, na qual os veículos enviam informações referentes à posição e sua velocidade. No ns-3, como mencionado no Capítulo 3, as informações coletadas a respeito da posição (Px, Py) e velocidade (Vx, Vy) são vetores. Como o ns-3 realiza apenas a coleta dessas informações, as demais *features* devem ser calculadas com base nessas informações. As *features* utilizadas são as mesmas descritas no Capítulo 3. Com as *features* calculadas, elas são submetidas ao modelo de ML treinado, que realiza a predição do valor do tempo de vida do enlace.

Com o tempo de vida do enlace e a taxa de ocupação da CPU, ocorre a Etapa 4. É realizado um *trade-off* entre os dois valores, a fim de identificar qual veículo possui a melhor combinação entre CPU livre e TVE predito. É escolhido o veículo com o percentual de CPU ocupada mais baixo e o valor predito mais alto do TVE. O veículo escolhido no processo anterior é integrado à arquitetura já configurada.

Finalmente, na etapa 5, o algoritmo de decisão do *offloading* implementado no Capítulo 4 realiza a distribuição das tarefas do pedestre para os servidores (veículo, borda e nuvem) a fim de serem processadas e retornadas ao cliente.

5.4 Avaliação Experimental

Na estratégia empregada neste novo procedimento de predição do TVE, foi adotado um método inovador para construir conjuntos de dados destinados ao treinamento de modelos de aprendizado de máquina. Os conjuntos de dados foram divididos com base na densidade de tráfego, em vez de considerar apenas um cenário mais geral.

5.4.1 Geração e Treinamento dos Modelos

Para treinar os modelos com base na densidade de tráfego em um ambiente urbano, foram criados três conjuntos de dados correspondentes às densidades baixa, média e alta, com os seguintes números de veículos: 50, 275 e 509, respectivamente. O processo de construção de cada *dataset* segue o fluxo comum de comunicação entre dois dispositivos, onde a cada 0,5 segundos é registrado uma amostra contendo as características necessárias para inferir o TVE: distância entre os nós, ângulos correspondentes, pseudo-ângulo, soma das velocidades, SLS, velocidade relativa e aceleração relativa.

A duração máxima de cada comunicação foi configurada para 160 segundos. Com base no *dataset* único utilizado no treinamento do modelo anteriormente utilizado nas predições, foram selecionados apenas os casos em que houve recuperações. Essa abordagem foi adotada visando fornecer conjuntos de dados mais realistas e, possivelmente, aprimorar o processo de predição do TVE em um ambiente urbano, considerando esses cenários mais complexos.

Após a preparação dos conjuntos de dados, eles foram submetidos a um processo de limpeza de dados para remover registros distorcidos. Os casos considerados distorcidos foram aqueles em que foi identificada perda de conexão seguida de restabelecimento da comunicação, resultando em perda de informação. Essas situações foram identificadas a partir dos registros, realizados a cada 0,5 segundos, nos quais foram observadas inconsistências. Além disso, foram eliminados registros com distância entre os nós superior a 250 metros, o qual é o alcance máximo configurado para as comunicações. Comunicações além desse limite indicam possíveis inconsistências no simulador. Após a etapa de limpeza, os conjuntos de dados passaram por um processo de normalização, uma vez que não seguiam uma distribuição normal e também porque os parâmetros de entrada possuem intervalos de valores muito diferentes entre si, sendo, portanto, necessária uma escala comum de valores. Foi empregada a técnica de normalização *MinMaxScaler* para esse propósito (RAJU *et al.*, 2020).

Com base nos resultados obtidos anteriormente, os algoritmos de ML SVR e XGBoost apresentaram o melhor desempenho. Por essa razão, esses algoritmos foram selecionados para o processo de avaliação de desempenho, juntamente com o algoritmo Adaboost e seus atributos de predição mencionados no trabalho de (ZHANG *et al.*, 2017). As seguintes métricas foram utilizadas para mensurar o desempenho dos modelos: MAE, RMSE, MAPE, tempo de predição e tamanho do modelo treinado. O tempo de predição refere-se ao tempo que o modelo leva para processar os dados de entrada e fornecer o TVE estimado.

Para realizar a avaliação dos modelos treinados, cada conjunto de dados foi dividido em dois conjuntos: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento corresponde a 80% do conjunto de dados original, enquanto os 20% restantes são destinados ao conjunto de teste. A divisão das amostras em cada conjunto é realizada de forma aleatória. Usando o conjunto de treinamento, os algoritmos de ML realizam o treinamento dos modelos para predição. Após o treinamento, os modelos são testados e avaliados utilizando as métricas mencionadas, utilizando as amostras do conjunto de teste como dados de entrada.

Cada algoritmo foi usado para treinar um modelo com base em cada conjunto de dados relacionado à densidade. Considerando que foram gerados três conjuntos de dados para as densidades baixa, média e alta, conforme os valores 50, 275 e 509 veículos respectivamente, houve um total de nove cenários avaliados. Portanto, para cada algoritmo, foram realizadas três avaliações diferentes, correspondendo aos diferentes conjuntos de dados de densidade.

5.4.2 Resultados da Avaliação dos Algoritmos de ML

Ao analisar as métricas avaliadas, pode-se observar que o algoritmo XGBoost apresentou taxas de erro menores nas métricas MAE, MAPE e RMSE em todos os modelos treinados, como demonstrado na Tabela 11. Esses resultados indicam que o XGBoost obteve uma diferença menor entre o TVE predito e o tempo de comunicação real das amostras. Isso sugere que as predições feitas pelo XGBoost estão mais próximas da realidade.

Outra conclusão importante é que as predições do XGBoost apresentaram um desvio médio menor em relação ao tempo de comunicação, conforme indicado pelos resultados do RMSE. Isso significa que as predições do XGBoost geraram menos distorções e valores discrepantes (*outliers*). Esses resultados destacam a eficácia do algoritmo XGBoost na predição do TVE, mostrando uma maior precisão e menor variabilidade em comparação com os outros algoritmos de ML avaliados.

Tabela 11 – Resultados da Avaliação Experimental.

Algoritmo	Densidade	MAE(s)	MAPE	RMSE(s)	Tempo de Predição (s)	Tamanho do Modelo
SVR	Baixa	3.23	0.47	4.13	**7,8e-04**	53.2 KB
XGBoost	Baixa	**2.34**	**0.28**	**3.20**	3,86e-03	787. 7 KB
Adaboost	Baixa	4.12	0.62	5.03	5,19e-03	**48.1 KB**
SVR	Média	5.13	1.10	6.69	**6,3e-04**	64 KB
XGBoost	Média	**4.11**	**1.00**	**5.55**	4,05e-03	1.6 MB
Adaboost	Média	4.58	1.27	5.61	6,49e-03	**59.5 KB**
SVR	Alta	4.10	0.43	5.37	**6,1e-04**	**22.8 KB**
XGBoost	Alta	**3.63**	**0.35**	**4.53**	4,015e-03	1 MB
Adaboost	Alta	4.34	0.48	5.33	8,06e-03	70 KB

Fonte: elaborada pelo autor.

Em relação ao tempo de predição, o algoritmo SVR obteve os melhores resultados nos modelos analisados, indicando ser mais rápido para inferir o TVE. No entanto, é importante observar que os algoritmos XGBoost e Adaboost apresentaram tempos de predição baixos, o que sugere haver menores chances de sobrecarga nos dispositivos que realizam as predições, independentemente do algoritmo escolhido entre esses três.

Quanto ao tamanho do modelo, os algoritmos Adaboost e SVR apresentaram modelos menores, enquanto os modelos treinados com o XGBoost mostraram-se significativamente maiores. Isso pode representar um problema se o dispositivo que carrega esses modelos tiver restrições de recursos de armazenamento. Nesse caso, é importante considerar a capacidade de armazenamento disponível no dispositivo ao escolher o algoritmo para a implementação.

Considerando que o tempo de inferência e as restrições de armazenamento não são um problema nos dispositivos simulados, a escolha do XGBoost para o sub-módulo Tomador de Decisão da arquitetura é justificada pelos melhores resultados obtidos nas métricas de erro. O XGBoost demonstrou maior precisão nas predições do TVE, conforme analisado anteriormente. Portanto, optar pelo XGBoost como algoritmo de escolha para esse experimento é uma decisão fundamentada na busca por resultados mais acurados e confiáveis.

5.4.3 Resultados do Offloading

Por último, para realizar a avaliação da solução, sobre a eficácia do *offloading* em ambiente urbano, dois cenários foram considerados. O primeiro cenário trata-se da abordagem adotada nos trabalhos da literatura e nos primeiros experimentos da dissertação, usando apenas um modelo de predição. O segundo cenário apresentado neste Capítulo, trata-se de um modelo adaptativo com base na densidade de tráfego, a partir do qual a arquitetura foi desenvolvida.

Com isso, foi realizada uma avaliação de dois algoritmos de decisão de VFC. Ambos foram implementados no capítulo 4, sendo o LECV e o LEV. A escolha dos dois se dá por serem os algoritmos que agregam veículos no processo de decisão. A função de predição do TVE dos dois algoritmos foi adaptada para utilizar os modelos de predição treinados pelo XGBoost.

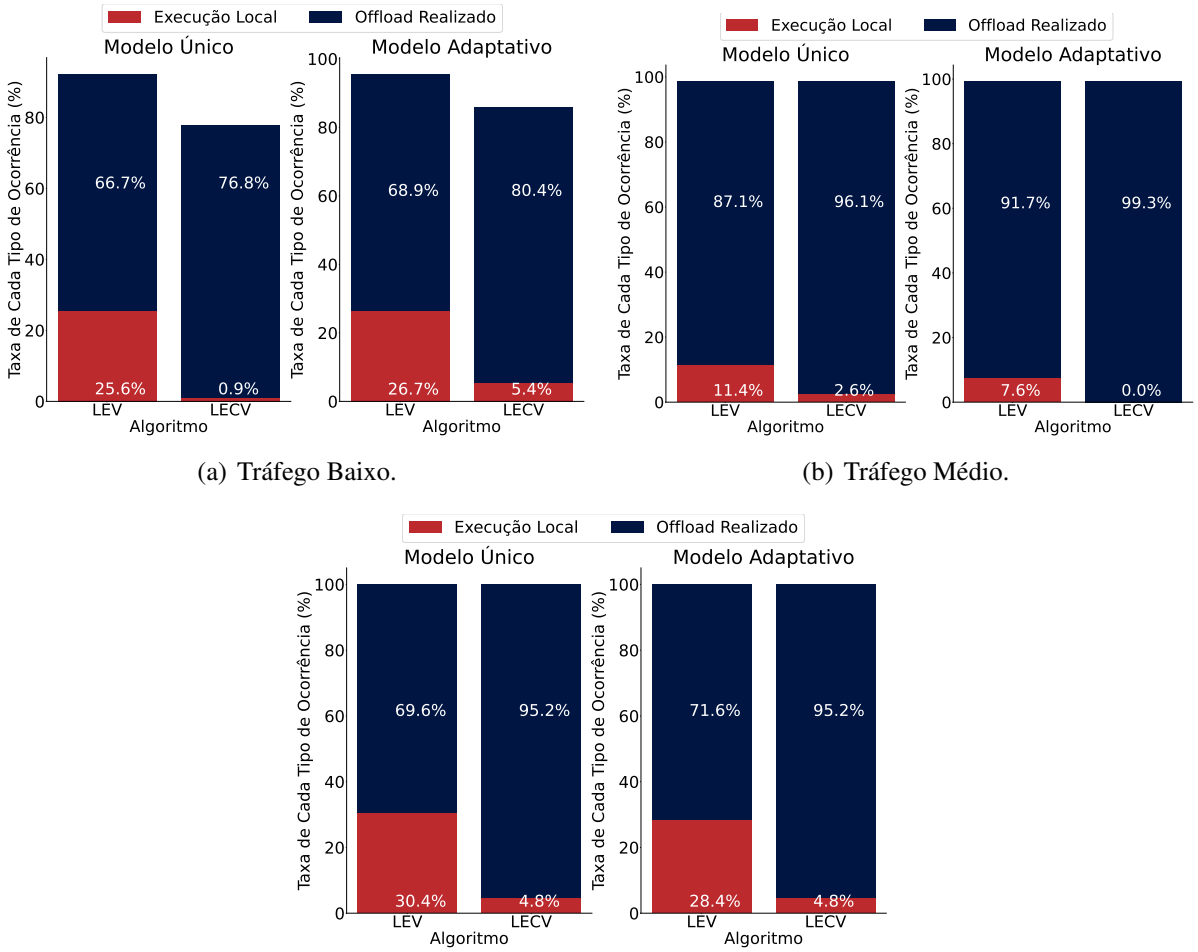
Aliado ao cenário e algoritmo de *offloading*, foi considerada a densidade de tráfego, com as densidades Baixa, Média e Alta, com seus valores definidos respectivamente em: 50, 276 e 509 veículos. A carga de trabalho aplicada na avaliação do *offloading* foi de imagem a ser processada no valor de 558 KB. A imagem representa uma placa de veículo a ser processada por uma aplicação de reconhecimento de placas, com o fluxo de execução simulado no ns-3. Com os fatores e níveis definidos foi realizada a execução dos experimentos. Para cada combinação foram realizadas 140 execuções, as quais são a simulação de um tráfego e a execução do *offloading* realizado pelo cliente para as entidades vizinhas. Foi definido que o fluxo de execução da aplicação de reconhecimento de placas geraria quatro tarefas ao todo de *offloading* para cada veículo. O número de veículos definidos como clientes foram 35. O processo de escolha do cliente foi realizado com base na distância percorrida, o objetivo era selecionar os nós clientes que mais se deslocassem no ambiente, a fim de gerar demanda e possibilitar a realização da avaliação do cenário com modelo adaptativo.

As primeiras métricas observadas são as taxas de *offloading* bem sucedidos e a taxa de execuções locais das tarefas, exibidas na Figura 29. A primeira análise é em torno do número total de *offloading* (Bem-sucedido e Local). Em todos os cenários analisados o modelo adaptativo obteve melhores resultados em relação ao modelo único, a exceção sendo no cenário com tráfego alto utilizando o LECV. Destacou-se o cenário de baixa densidade, onde o modelo adaptativo utilizando o algoritmo LECV obteve um percentual de *offloading* total de 8% a mais do que o modelo único utilizado o mesmo algoritmo.

Já, com o algoritmo de LEV no cenário de baixa densidade, o modelo adaptativo teve um percentual de *offloading* total de 3% a mais em relação ao modelo único. Demonstrando assim, que o modelo adaptativo consegue em diferentes condições com servidores disponíveis ser mais eficiente no processo de escolha, o que ocasiona mais *offloadings* computacionais, mais bem-sucedidos.

A taxa de execução local de tarefas, também foi considerada na avaliação. Para ter uma compreensão do que significa os números de execuções locais é importante observar a taxa de *offloading* bem-sucedido e a taxa de recuperação. Uma alta taxa de execução local,

Figura 29 – Taxa de *offloading* e execução local.



(a) Tráfego Baixo.

(b) Tráfego Médio.

(c) Tráfego Alto.

Fonte: elaborada pelo autor.

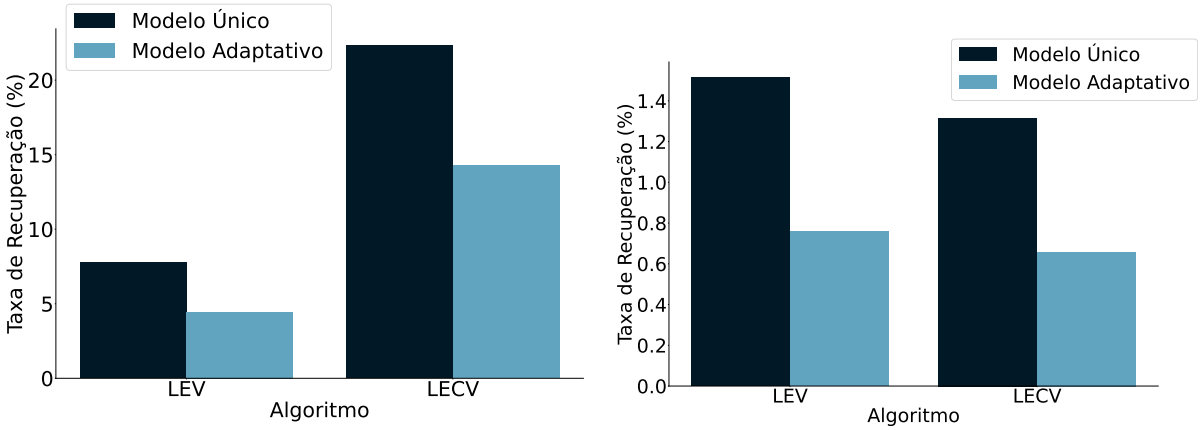
porém com baixo número de recuperações, indica que a predição foi eficiente, já que executar local foi melhor que enviar para um servidor que poderia ter uma alta probabilidade de perda de conexão. Porém, um alto número de execuções locais, sem grandes perdas de recuperações, indica que o modelo não está enviando tarefas para servidores com maior probabilidade realizar um processamento bem-sucedido.

Nos resultados obtidos, a taxa de execuções locais do modelo adaptativo foi menor em relação ao modelo único nos tráfegos de média e alta densidades nos algoritmos LECV e LEV. Com essa diminuição chegando a aproximadamente 25%, no cenário de alta densidade de tráfego. No cenário de densidade média teve destaque o algoritmo LECV que não obteve execuções locais, enquanto o LECV do cenário de modelo único obteve 2,6% de execuções locais.

Outra métrica analisada foi o número de recuperações de tarefas realizadas devido à perda de contato entre cliente e servidor durante o processo de *offloading*. Quando há recupera-

ções, são executadas cópias locais de tarefas, assim que uma queda de conexão é detectada. A Figura 30 apresenta os resultados dessa métrica nos cenários avaliados.

Figura 30 – Taxa de Recuperação das Tarefas.



(a) Tráfego Baixo. (b) Tráfego Médio.

Fonte: elaborada pelo autor.

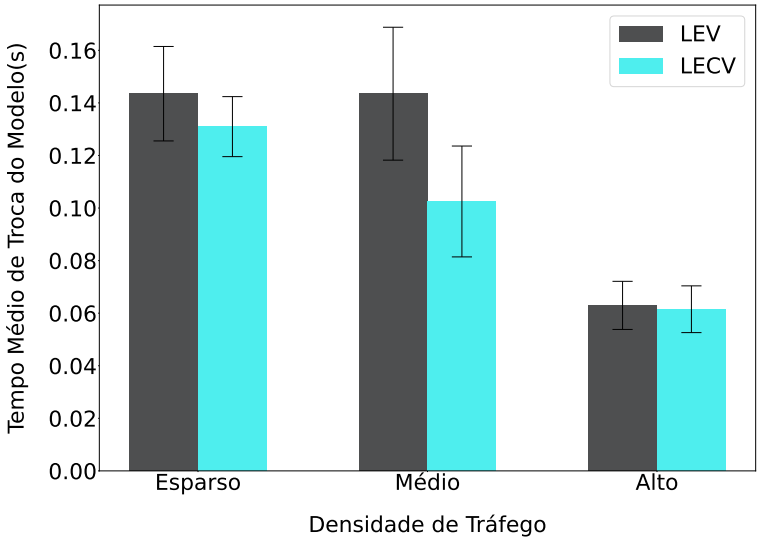
Nos cenários de tráfegos baixo e médio, o modelo adaptativo mostrou-se mais eficaz nas predições, realizando predições mais assertivas em relação ao modelo único. No cenário com tráfego Alto, os resultados foram iguais para os dois algoritmos de decisão nos dois cenários analisados, devido à quantidade de servidores disponíveis. Uma vez que há mais servidores, diminuem-se as chances de perda de conexão, por existirem muitos veículos próximos que podem ser escolhidos. A diminuição no número de recuperações que o modelo adaptativo obteve no cenário médio foi de 50%, no cenário baixo o percentual também aproximou-se desse número.

Com os resultados da taxa de recuperação, pode concluir-se que houve de fato uma melhor eficiência de predição nos cenários de baixa e média densidade. No modelo adaptativo do cenário de baixa densidade, o número maior de execuções locais, trata-se de decisões acertadas no processo de decisão, o que impactou positivamente na diminuição da taxa de recuperações.

O uso de um modelo adaptativo conforme a densidade de tráfego em um cenário urbano é bastante promissor para o processo de *offloading* computacional, como mostram os resultados. Porém, há uma preocupação acerca de uma possível sobrecarga que os modelos de ML podem gerar. No modelo adaptativo pode ser necessário mudanças de modelo e sincronização dos mesmos entre clientes e os repositórios remotos. Considerando essas preocupações, foi mensurado o tempo médio gasto por atualização do modelo por parte dos clientes. O cliente ao iniciar o processo de *offloading* verifica através do módulo de monitoramento o contexto de tráfego a qual está inserido. Uma vez identificado o tráfego da vizinhança, caso não possua

o modelo, solicita à nuvem o mesmo. Os resultados obtidos pela avaliação do tempo dessa atualização são demonstradas na Figura 31.

Figura 31 – Tempo médio de atualização do modelo.



Fonte: elaborada pelo autor.

Não houve uma sobrecarga muito alta, nas trocas de modelo. Haja vista que o algoritmo XGBoost gera modelos um pouco maiores, que o SVR ou Adaboost. Os resultados mostram um tempo médio das trocas realizadas por cada veículo em quatro momentos distintos durante a execução. No tráfego médio e baixo, por terem mais trocas, acabaram tendo um resultado maior em relação ao tráfego alto. No tráfego alto houve menos trocas, o que ajudou a ter uma média menor.

Por fim, foi avaliado também o tempo médio de *offloading*, os quais são o tempo total entre o envio das tarefas, processamento e recebimento de todas as tarefas. Os resultados demonstram que não houve diferenças entre os tempos, com uma análise estatística de Wilcoxon sendo realizada. Os resultados estão resumidos na Tabela 12.

Tabela 12 – Valores de *p* para testes de *Wilcoxon* pareados.

Tráfego	**LEV Modelo único x LEV Modelo adaptativo**	**LECV modelo único x LECV modelo adaptativo**
Baixo	0,861	0,206
Médio	0,181	0,237
Alto	0,555	0,846

Fonte: elaborada pelo autor.

Conforme observado, os valores "p"foram superiores a 0,05, que corresponde à taxa de confiança adotada, logo não há evidências para rejeitar a hipótese nula (tempos iguais

estatisticamente). Com isso, conclui-se que não há maiores prejuízos de tempo no processo de *offloading* computacional em relação aos algoritmos LECV e LEV nos cenários de baixa, média e alta densidade.

5.5 Considerações Finais

Este Capítulo abordou o OE3, demonstrando como agregar os avanços obtidos nos OE1 e OE2, através de uma arquitetura proposta e implementada. Cada módulo desenvolvido foi abordado, onde foi destacado em qual parte da aplicação os modelos de predição do TVE eram utilizados, assim como o algoritmo de decisão do *offloading*. Uma visão geral da comunicação VFC foi apresentada, demonstrando um uso prático da arquitetura em um cenário de VANET. Para realizar a avaliação da arquitetura, novos modelos de predição do TVE foram treinados, a fim de serem utilizados em contextos específicos pelos clientes de *offloading*.

Os resultados alcançados demonstraram que o uso de modelos de ML, escolhidos a partir de contexto para auxiliar no processo de decisão e balanceamento de tarefas de *offloading* mostrou-se eficiente em relação a abordagens comuns com uso de modelos generalistas, com base em todo o cenário. A abordagem adaptativa obteve êxito em diminuir o número de falhas no processo de decisão do *offloading* o que resultou em maiores taxas de *offloading* bem-sucedidos. Além disso, a abordagem adaptativa não demonstrou sobrecargas adicionais, devido ao processo de sincronização dos modelos, o que não afetou negativamente o processo de *offloading* computacional em VANET.

6 CONCLUSÃO

Este trabalho investigou como melhorar os avanços da comunicação em redes veiculares, especificamente VFC, através dos objetivos definidos e explorados nos capítulos desta dissertação. O primeiro objetivo específico foi propor uma metodologia de predição do tempo de vida do enlace por meio do uso de Aprendizado de Máquina. A eficácia de alguns algoritmos de Aprendizado de Máquina da literatura foram avaliados conforme um novo conjunto de atributos propostos neste trabalho. Os resultados experimentais alcançados em *datasets* simulados demonstram que as abordagens de ML baseadas em regressão, como SVR, são promissoras para estimar o TVE entre nós em VANETs. A estratégia melhorou a eficiência da predição em relação a abordagens comuns na literatura, baseadas em equações matemáticas. Além disso, identificar variáveis que influenciam o TVE provou ser uma importante contribuição para futuras pesquisas.

Prosseguindo, o segundo objetivo específico foi abordar uma solução para estender aplicações VFC em redes veiculares, trazendo mais dispositivos para o ambiente inserindo pedestres, que atuam como clientes do processo de *offloading* computacional. Um algoritmo de decisão de *offloading* foi implementado, sendo realizados experimentos para avaliar o desempenho do algoritmo. O algoritmo foi avaliado em três contextos de densidade de tráfego e duas categorias de carga de trabalho, juntamente com um algoritmo de decisão aleatória. Como os resultados demonstraram, essa abordagem tem o potencial de melhorar o tempo de execução e a eficiência do *offloading* computacional em ambientes VFC com pedestres atuando como clientes. Tais melhorias resultam em menor sobrecarga dos servidores de borda e menos erros na escolha dos destinos para *offloading*, diminuindo retransmissões e latência e, consequentemente, melhorando a qualidade dos aplicativos ITS ou aplicativos pessoais de pedestres.

Finalmente, o terceiro objetivo específico, foi implementar uma arquitetura, agregando o OE1 e OE2 desenvolvidos ao longo da dissertação, utilizando a metodologia de predição com Aprendizado de Máquina, juntamente em uma rede VFC. A partir dessa arquitetura foi proposta uma nova hipótese de melhoria nas taxas de *offloading* através do uso de modelos com base no contexto atual de tráfego na qual o cliente está inserido. Para realizar a avaliação foi considerando o algoritmo de decisão do *offloading* em VFC em dois aspectos: LEV e LECV. A fim de escolher o algoritmo de ML para ser inserido no módulo de monitoramento da arquitetura, foi realizada uma avaliação de desempenho com uma metodologia de predição da literatura, na qual o algoritmo XGBoost foi escolhido. Os resultados demonstram que a metodologia de predição abordada nesse trabalho, mostrou-se mais eficaz, com predições mais realistas no

módulo de monitoramento da arquitetura. Por sua vez, o modelo adaptativo obteve menos erros de decisão em relação ao modelo único, com menos recuperações de tarefas perdidas e consequentemente mais *offloadings* bem-sucedidos. Os resultados indicam que o uso de um modelo específico treinado para determinado tráfego, pode ser o caminho mais viável no uso de ML na predição do TVE em *offloading* computacional, em cenários de VFC.

Com isso, o presente trabalho conseguiu obter avanços no processo de tomada de decisão em *offloading* computacional em uma VFC. Os resultados alcançados com o uso de modelos adaptativos na predição do tempo de vida do enlace demonstraram-se melhores do que a utilização de modelos de predição individual. Tornando assim, o processo menos propenso a erros e sem gerar sobrecargas a mais para realizar os procedimentos. Além das contribuições já citadas, os artefatos desenvolvidos, tais como, *datasets* e *traces* de mobilidade estão disponibilizadas à comunidade, já que há dificuldade em encontrá-los em repositórios públicos.

6.1 Resultados Alcançados

Alguns dos resultados desta dissertação:

- Rocha, P. H., de Souza, A. B., Silva, F. A., & Rego, P. A. (2022, May). Algoritmo de Decisão para Offloading Computacional em Vehicular Fog Computing com Pedestres. In Anais do VI Workshop de Computação Urbana (pp. 224-237). SBC. (Qualis B1)(Menção honrosa).
- Rocha, P. H., de Souza, A. B., Maia, J. G., Mattos, C. L., Silva, F. A., & Rego, P. A. (2022, May). Avaliação da Predição do Tempo de Vida do Enlace no Processo de Offloading Computacional em VANETs. In Anais do VI Workshop de Computação Urbana (pp. 266-279). SBC. (Qualis B1)
- Rocha, P., Souza, A., Maia, G., Mattos, C., Silva, F. A., Rego, P., ... & Lee, J. W. (2022). Evaluating Link Lifetime Prediction to Support Computational Offloading Decision in VANETs. Sensors, 22(16), 6038. (Qualis A1)
- Rocha, P. H., de Souza, A. B., Silva, F. A., & Rego, P. A. (2022). Decision Algorithm for Computational Offloading in Vehicular Fog Computing with Pedestrians. In IEEE International Conference on Cloud Networking (CloudNet). (Qualis A2)

6.2 Trabalhos Futuros

Como trabalhos futuros e possíveis extensões e melhorias para este trabalho, poderiam ser considerado os seguintes tópicos:

- A inclusão de novas métricas de avaliação nos testes da arquitetura poderia ser considerada. Essas métricas seriam o consumo de energia dos dispositivos e o custo das execuções de tarefa na nuvem. Uma vez que a redução de energia nos dispositivos móveis, assim como os gastos no aluguel de servidores em Nuvem, são indicadores interessantes a serem observados, já que auxiliam na percepção da qualidade no processo de *offloading*.
- Outra área que pode ser abordada e estendida nesse trabalho é a parte de segurança, especialmente criptografia nos dados compartilhados no processo de *offloading*. A inclusão dessa funcionalidade pode ser analisada em uma eventual análise de desempenho, a fim de medir o seu impacto na qualidade de *offloading*.
- A inclusão de sistema de *caching* nos servidores, também pode ser pesquisada para este trabalho, uma vez que isso poderia melhorar a eficiência na disponibilização dos modelos armazenados.

REFERÊNCIAS

ABDULKAREEM, N. M.; ABDULAZEEZ, A. M. *et al.* Machine learning classification based on radom forest algorithm: A review. **International Journal of Science and Business**, IJSAB International, v. 5, n. 2, p. 128–142, 2021.

ALABBASI, A.; AGGARWAL, V. Joint information freshness and completion time optimization for vehicular networks. **IEEE Transactions on Services Computing**, IEEE, 2020.

ALAHMAD, A. S.; KAHTAN, H.; ALZOUBI, Y. I.; ALI, O.; JARADAT, A. Mobile cloud computing models security issues: A systematic review. **Journal of Network and Computer Applications**, Elsevier, v. 190, p. 103152, 2021.

ALDMOUR, R.; YOUSEF, S.; BAKER, T.; BENKHELIFA, E. An approach for offloading in mobile cloud computing to optimize power consumption and processing time. **Sustainable Computing: Informatics and Systems**, Elsevier, v. 31, p. 100562, 2021.

ALMEIDA, T. T. de; JÚNIOR, J. G. R.; CAMPISTA, M. E. M.; COSTA, L. H. M. K. Uma análise de desempenho do wi-fi direct para comunicações veículo-pedestre. In: SBC. **Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2020. p. 253–266.

ALPAYDIN, E. **Introduction to machine learning**. [S.l.: s.n.], 2020.

AMADEO, M.; CAMPOLO, C.; MOLINARO, A. Enhancing ieee 802.11 p/wave to provide infotainment applications in vanets. **Ad Hoc Networks**, Elsevier, v. 10, n. 2, p. 253–269, 2012.

ANWER, M. S.; GUY, C. A survey of vanet technologies. **Journal of Emerging Trends in Computing and Information Sciences**, Citeseer, v. 5, n. 9, p. 661–671, 2014.

APICELLA, A.; DONNARUMMA, F.; ISGRÒ, F.; PREVETE, R. A survey on modern trainable activation functions. **Neural Networks**, Elsevier, v. 138, p. 14–32, 2021.

ARENA, F.; PAU, G.; SEVERINO, A. A review on ieee 802.11 p for intelligent transportation systems. **Journal of Sensor and Actuator Networks**, MDPI, v. 9, n. 2, p. 22, 2020.

ARIF, M.; WANG, G.; BHUIYAN, M. Z. A.; WANG, T.; CHEN, J. A survey on security attacks in vanets: Communication, applications and challenges. **Vehicular Communications**, Elsevier, v. 19, p. 100179, 2019.

AYODELE, T. O. Types of machine learning algorithms. **New advances in machine learning**, InTech Rijeka, Croatia, v. 3, p. 19–48, 2010.

BANGUI, H.; GE, M.; BUHNOVA, B. A hybrid machine learning model for intrusion detection in vanet. **Computing**, Springer, v. 104, n. 3, p. 503–531, 2022.

BOUKERCHE, A.; SOTO, V. An efficient mobility-oriented retrieval protocol for computation offloading in vehicular edge multi-access network. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 21, n. 6, p. 2675–2688, 2020.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

BU, J.; TAN, G.; DING, N.; LIU, M.; SON, C. Implementation and evaluation of wave 1609.4/802.11 p in ns-3. In: **Proceedings of the 2014 Workshop on ns-3**. [S.l.: s.n.], 2014. p. 1–8.

BUTE, M. S.; FAN, P.; LIU, G.; ABBAS, F.; DING, Z. A collaborative task offloading scheme in vehicular edge computing. In: IEEE. **2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)**. [S.l.], 2021. p. 1–5.

BUTE, M. S.; FAN, P.; LIU, G.; ABBAS, F.; DING, Z. A cluster-based cooperative computation offloading scheme for c-v2x networks. **Ad Hoc Networks**, Elsevier, p. 102862, 2022.

CAMPOLO, C.; VINEL, A.; MOLINARO, A.; KOUCHERYAVY, Y. Modeling broadcasting in ieee 802.11 p/wave vehicular networks. **IEEE Communications letters**, IEEE, v. 15, n. 2, p. 199–201, 2010.

CARLEO, G.; CIRAC, I.; CRANMER, K.; DAUDET, L.; SCHULD, M.; TISHBY, N.; VOGT-MARANTO, L.; ZDEBOROVÁ, L. Machine learning and the physical sciences. **Reviews of Modern Physics**, APS, v. 91, n. 4, p. 045002, 2019.

CHOMBOON, K.; CHUJAI, P.; TEERARASSAMEE, P.; KERDPRASOP, K.; KERDPRASOP, N. An empirical study of distance metrics for k-nearest neighbor algorithm. In: **Proceedings of the 3rd international conference on industrial application engineering**. [S.l.: s.n.], 2015. p. 280–285.

CHOPRA, A.; RAHMAN, A. U.; MALIK, A. W.; RAVANA, S. D. Adaptive-learning-based vehicle-to-vehicle opportunistic resource-sharing framework. **IEEE Internet of Things Journal**, IEEE, v. 9, n. 14, p. 12497–12504, 2021.

DENG, Z.; CAI, Z.; LIANG, M. A multi-hop vanets-assisted offloading strategy in vehicular mobile edge computing. **IEEE Access**, IEEE, v. 8, p. 53062–53071, 2020.

DESHPANDE, N. M.; GITE, S.; ALUVALU, R. A review of microscopic analysis of blood cells for disease detection with ai perspective. **PeerJ Computer Science**, PeerJ Inc., v. 7, p. e460, 2021.

DOMINGUEZ, J. M. L.; SANGUINO, T. J. M. Review on v2x, i2x, and p2x communications and their applications: a comprehensive analysis over time. **Sensors**, MDPI, v. 19, n. 12, p. 2756, 2019.

DOU, J.; YUNUS, A. P.; BUI, D. T.; SAHANA, M.; CHEN, C.-W.; ZHU, Z.; WANG, W.; PHAM, B. T. Evaluating gis-based multiple statistical models and data mining for earthquake and rainfall-induced landslide susceptibility using the lidar dem. **Remote Sensing**, MDPI, v. 11, n. 6, p. 638, 2019.

DRUCKER, H.; BURGES, C. J.; KAUFMAN, L.; SMOLA, A.; VAPNIK, V. Support vector regression machines. **Advances in neural information processing systems**, v. 9, 1996.

DUA, N.; SINGH, S. N.; SEMWAL, V. B. Multi-input cnn-gru based human activity recognition using wearable sensors. **Computing**, Springer, v. 103, n. 7, p. 1461–1478, 2021.

ECKERMANN, F.; KAHLERT, M.; WIETFELD, C. Performance analysis of c-v2x mode 4 communication introducing an open-source c-v2x simulator. In: IEEE. **2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)**. [S.l.], 2019. p. 1–5.

EĞRIOĞLU, E.; ALADAĞ, Ç. H.; GÜNAY, S. A new model selection strategy in artificial neural networks. **Applied Mathematics and Computation**, Elsevier, v. 195, n. 2, p. 591–597, 2008.

EZE, E. C.; ZHANG, S.; LIU, E. Vehicular ad hoc networks (vanets): Current state, challenges, potentials and way forward. In: IEEE. **2014 20th international conference on automation and computing**. [S.l.], 2014. p. 176–181.

FARAHBAKHSH, F.; SHAHIDINEJAD, A.; GHOBAEI-ARANI, M. Context-aware computation offloading for mobile edge computing. **Journal of Ambient Intelligence and Humanized Computing**, Springer, p. 1–13, 2021.

GARDNER, M. W.; DORLING, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. **Atmospheric environment**, Elsevier, v. 32, n. 14-15, p. 2627–2636, 1998.

GONÇALVES, D. M.; BITTENCOURT, L. F.; MADEIRA, E. M. Análise da predição de mobilidade na migração de aplicações em computação em névoa. In: SBC. **Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2019. p. 580–593.

GRASSI, G.; PESAVENTO, D.; PAU, G.; VUYYURU, R.; WAKIKAWA, R.; ZHANG, L. Vanet via named data networking. In: IEEE. **2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)**. [S.l.], 2014. p. 410–415.

GRÖMPING, U. Variable importance assessment in regression: linear regression versus random forest. **The American Statistician**, Taylor & Francis, v. 63, n. 4, p. 308–319, 2009.

HAMDI, M. M.; AUDAH, L.; RASHID, S. A.; ALANI, S. Vanet-based traffic monitoring and incident detection system: A review. **International Journal of Electrical & Computer Engineering (2088-8708)**, v. 11, n. 4, 2021.

HÄRRI, J.; BONNET, C.; FILALI, F. Kinetic mobility management applied to vehicular ad hoc network protocols. **Computer Communications**, Elsevier, v. 31, n. 12, p. 2907–2924, 2008.

HASROUNY, H.; SAMHAT, A. E.; BASSIL, C.; LAOUITI, A. Vanet security challenges and solutions: A survey. **Vehicular Communications**, Elsevier, v. 7, p. 7–20, 2017.

HASTIE, T.; ROSSET, S.; ZHU, J.; ZOU, H. Multi-class adaboost. **Statistics and its Interface**, International Press of Boston, v. 2, n. 3, p. 349–360, 2009.

HAZARIKA, B.; SINGH, K.; BISWAS, S.; LI, C.-P. Drl-based resource allocation for computation offloading in iov networks. **IEEE Transactions on Industrial Informatics**, IEEE, v. 18, n. 11, p. 8027–8038, 2022.

HE, Y.; ZHAI, D.; HUANG, F.; WANG, D.; TANG, X.; ZHANG, R. Joint task offloading, resource allocation, and security assurance for mobile edge computing-enabled uav-assisted vanets. **Remote Sensing**, Multidisciplinary Digital Publishing Institute, v. 13, n. 8, p. 1547, 2021.

HOU, X.; LI, Y.; CHEN, M.; WU, D.; JIN, D.; CHEN, S. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. **IEEE Transactions on Vehicular Technology**, IEEE, v. 65, n. 6, p. 3860–3873, 2016.

HUANG, C.; LU, R.; CHOO, K.-K. R. Vehicular fog computing: architecture, use case, and security and forensic challenges. **IEEE Communications Magazine**, IEEE, v. 55, n. 11, p. 105–111, 2017.

HUANG, C.-M.; CHIANG, M.-S.; DAO, D.-T.; SU, W.-L.; XU, S.; ZHOU, H. V2v data offloading for cellular network based on the software defined network (sdn) inside mobile edge computing (mec) architecture. **IEEE Access**, IEEE, v. 6, p. 17741–17755, 2018.

HUSSAIN, M.; AZAR, A. T.; AHMED, R.; AMIN, S. U.; QURESHI, B.; REDDY, V. D.; ALAM, I.; KHAN, Z. I. *et al.* Song: A multi-objective evolutionary algorithm for delay and energy aware facility location in vehicular fog networks. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 23, n. 2, p. 667, 2023.

HUSSAIN, M. M.; BEG, M. S. Code-v: Multi-hop computation offloading in vehicular fog computing. **Future Generation Computer Systems**, Elsevier, v. 116, p. 86–102, 2021.

HUSSAIN, S.; WU, D.; MEMON, S.; BUX, N. K. Vehicular ad hoc network (vanet) connectivity analysis of a highway toll plaza. **Data**, Multidisciplinary Digital Publishing Institute, v. 4, n. 1, p. 28, 2019.

JIANG, T.; GRADUS, J. L.; ROSELLINI, A. J. Supervised machine learning: a brief primer. **Behavior Therapy**, Elsevier, v. 51, n. 5, p. 675–687, 2020.

JOHANSSON, U.; BOSTRÖM, H.; LÖFSTRÖM, T.; LINUSSON, H. Regression conformal prediction with random forests. **Machine learning**, Springer, v. 97, p. 155–176, 2014.

KENNEY, J. B. Dedicated short-range communications (dsrc) standards in the united states. **Proceedings of the IEEE**, IEEE, v. 99, n. 7, p. 1162–1182, 2011.

KESHARI, N.; SINGH, D.; MAURYA, A. K. A survey on vehicular fog computing: Current state-of-the-art and future directions. **Vehicular Communications**, Elsevier, v. 38, p. 100512, 2022.

KHAN, M. J.; KHAN, M. A.; BEG, A.; MALIK, S.; EL-SAYED, H. An overview of the 3gpp identified use cases for v2x services. **Procedia Computer Science**, Elsevier, v. 198, p. 750–756, 2022.

KHATRI, S.; VACHHANI, H.; SHAH, S.; BHATIA, J.; CHATURVEDI, M.; TANWAR, S.; KUMAR, N. Machine learning models and techniques for vanet based traffic management: Implementation issues and challenges. **Peer-to-Peer Networking and Applications**, Springer, v. 14, n. 3, p. 1778–1805, 2021.

KIM, K.; KOO, S.; CHOI, J.-W. Analysis on path rerouting algorithm based on v2x communication for traffic flow improvement. In: IEEE. **2020 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.], 2020. p. 251–254.

KISHORE, R.; KAUR, T. Backpropagation algorithm: an artificial neural network approach for pattern recognition. **International Journal of Scientific & Engineering Research**, v. 3, n. 6, p. 1–4, 2012.

KULLA, E.; MORITA, S.; KATAYAMA, K.; BAROLLI, L. Route lifetime prediction method in vanet by using aodv routing protocol (aodv-lp). In: BAROLLI, L.; JAVAID, N.; IKEDA, M.; TAKIZAWA, M. (Ed.). **Complex, Intelligent, and Software Intensive Systems**. Cham: Springer International Publishing, 2019. p. 3–11.

KUMAR, K.; LIU, J.; LU, Y.-H.; BHARGAVA, B. A survey of computation offloading for mobile systems. **Mobile networks and Applications**, Springer, v. 18, n. 1, p. 129–140, 2013.

LEE, M.; ATKISON, T. Vanet applications: Past, present, and future. **Vehicular Communications**, Elsevier, v. 28, p. 100310, 2021.

LI, B.; PENG, Z.; HOU, P.; HE, M.; ANISETTI, M.; JEON, G. Reliability and capability based computation offloading strategy for vehicular ad hoc clouds. **Journal of cloud computing**, Springer, v. 8, n. 1, p. 1–14, 2019.

LI, C.; WANG, S.; HUANG, X.; LI, X.; YU, R.; ZHAO, F. Parked vehicular computing for energy-efficient internet of vehicles: A contract theoretic approach. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 4, p. 6079–6088, 2018.

LI, F.; CHEN, W.; SHUI, Y.; WANG, J.; YANG, K.; XU, L.; YU, J.; LI, C. Connectivity probability analysis of vanets at different traffic densities using measured data at 5.9 ghz. **Physical Communication**, Elsevier, v. 35, p. 100709, 2019.

LIN, L.; LIAO, X.; JIN, H.; LI, P. Computation offloading toward edge computing. **Proceedings of the IEEE**, IEEE, v. 107, n. 8, p. 1584–1607, 2019.

LIN, Y.-D.; HU, J.-C.; KAR, B.; YEN, L.-H. Cost minimization with offloading to vehicles in two-tier federated edge and vehicular-fog systems. In: IEEE. **2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)**. [S.l.], 2019. p. 1–6.

LIU, N.; LIU, M.; CAO, J.; CHEN, G.; LOU, W. When transportation meets communication: V2p over vanets. In: IEEE. **2010 IEEE 30th International Conference on Distributed Computing Systems**. [S.l.], 2010. p. 567–576.

LIU, Y.; WANG, Y.; ZHANG, J. New machine learning algorithm: Random forest. In: SPRINGER. **International Conference on Information Computing and Applications**. [S.l.], 2012. p. 246–252.

MACH, P.; BECVAR, Z. Mobile edge computing: A survey on architecture and computation offloading. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 3, p. 1628–1656, 2017.

MAHESH, B. Machine learning algorithms-a review. **International Journal of Science and Research (IJSR).[Internet]**, v. 9, p. 381–386, 2020.

MAMMONE, A.; TURCHI, M.; CRISTIANINI, N. Support vector machines. **Wiley Interdisciplinary Reviews: Computational Statistics**, Wiley Online Library, v. 1, n. 3, p. 283–289, 2009.

MARÍN, A.; MARTÍNEZ-MERINO, L. I.; PUERTO, J.; RODRÍGUEZ-CHÍA, A. M. The soft-margin support vector machine with ordered weighted average. **Knowledge-Based Systems**, Elsevier, v. 237, p. 107705, 2022.

MEZZAVILLA, M.; ZHANG, M.; POLESE, M.; FORD, R.; DUTTA, S.; RANGAN, S.; ZORZI, M. End-to-end simulation of 5g mmwave networks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 20, n. 3, p. 2237–2263, 2018.

MISHRA, R.; SINGH, A.; KUMAR, R. Vanet security: Issues, challenges and solutions. In: IEEE. **2016 international conference on electrical, electronics, and optimization techniques (ICEEOT)**. [S.l.], 2016. p. 1050–1055.

MONTEIRO, R.; SARGENTO, S.; VIRIYASITAVAT, W.; TONGUZ, O. K. Improving vanet protocols via network science. In: IEEE. **2012 IEEE Vehicular Networking Conference (VNC)**. [S.l.], 2012. p. 17–24.

NABIL, M.; HAJAMI, A.; HAQIQ, A. Predicting the route of the longest lifetime and the data packet delivery time between two vehicles in vanet. **Mobile Information Systems**, Hindawi, v. 2019, 2019.

NAQA, I. E.; MURPHY, M. J. What is machine learning? In: **machine learning in radiation oncology**. [S.l.]: Springer, 2015. p. 3–11.

NEŠOVIĆ, E. On geometric interpretation of pseudo-angle in minkowski plane. **International Journal of Geometric Methods in Modern Physics**, World Scientific, v. 14, n. 05, p. 1750068, 2017.

NGUYEN, Q.-H.; DRESSLER, F. A smartphone perspective on computation offloading—a survey. **Computer Communications**, Elsevier, v. 159, p. 133–154, 2020.

NGUYEN, Q.-H.; MOROLD, M.; DAVID, K.; DRESSLER, F. Adaptive safety context information for vulnerable road users with mec support. In: IEEE. **2019 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)**. [S.l.], 2019. p. 28–35.

NGUYEN, Q.-H.; MOROLD, M.; DAVID, K.; DRESSLER, F. Car-to-pedestrian communication with mec-support for adaptive safety of vulnerable road users. **Computer Communications**, Elsevier, v. 150, p. 83–93, 2020.

NING, Z.; HUANG, J.; WANG, X. Vehicular fog computing: Enabling real-time traffic management for smart cities. **IEEE Wireless Communications**, IEEE, v. 26, n. 1, p. 87–93, 2019.

NIU, Y.; LI, Y.; JIN, D.; SU, L.; VASILAKOS, A. V. A survey of millimeter wave communications (mmwave) for 5g: opportunities and challenges. **Wireless networks**, Springer, v. 21, n. 8, p. 2657–2676, 2015.

NOBLE, W. S. What is a support vector machine? **Nature biotechnology**, Nature Publishing Group, v. 24, n. 12, p. 1565–1567, 2006.

NOBRE, M.; SILVA, I.; GUEDES, L. A.; PORTUGAL, P. Towards a wirelesshart module for the ns-3 simulator. In: IEEE. **2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)**. [S.l.], 2010. p. 1–4.

OSIBO, B. K.; JIN, Z.; MA, T.; MARAH, B. D.; ZHANG, C.; JIN, Y. An edge computational offloading architecture for ultra-low latency in smart mobile devices. **Wireless Networks**, Springer, v. 28, n. 5, p. 2061–2075, 2022.

PASHA, M.; FAROOQ, M. U.; RAMSHA, A. A. An efficient scheme for reliable file transfer using residual link lifetime estimation based on sparse vehicular routes. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2021. v. 1964, n. 4, p. 042007.

QI, H.; GANI, A. Research on mobile cloud computing: Review, trend and perspectives. In: IEEE. **2012 second international conference on digital information and communication technology and it's applications (DICTAP)**. [S.l.], 2012. p. 195–202.

QIU, Y.; ZHOU, J.; KHANDELWAL, M.; YANG, H.; YANG, P.; LI, C. Performance evaluation of hybrid woa-xgboost, gwo-xgboost and bo-xgboost models to predict blast-induced ground vibration. **Engineering with Computers**, Springer, p. 1–18, 2021.

RAHMAN, A. U.; MALIK, A. W.; SATI, V.; CHOPRA, A.; RAVANA, S. D. Context-aware opportunistic computing in vehicle-to-vehicle networks. **Vehicular Communications**, Elsevier, v. 24, p. 100236, 2020.

RAJU, V. G.; LAKSHMI, K. P.; JAIN, V. M.; KALIDINDI, A.; PADMA, V. Study the influence of normalization/transformation process on the accuracy of supervised classification. In: IEEE. **2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)**. [S.l.], 2020. p. 729–735.

RAMCHOUN, H.; GHANOU, Y.; ETTAOUIL, M.; IDRISSI, M. A. J. Multilayer perceptron: Architecture optimization and training. International Journal of Interactive Multimedia and Artificial Intelligence . . . , 2016.

RANI, P.; SHARMA, R. Intelligent transportation system for internet of vehicles based vehicular networks for smart cities. **Computers and Electrical Engineering**, Elsevier, v. 105, p. 108543, 2023.

RAUT, C. M.; DEVANE, S. R. Intelligent transportation system for smartcity using vanet. In: IEEE. **2017 International Conference on Communication and Signal Processing (ICCSP)**. [S.l.], 2017. p. 1602–1605.

RAW, R. S.; KUMAR, M.; SINGH, N. Security challenges, issues and their solutions for vanet. **International journal of network security & its applications**, Academy & Industry Research Collaboration Center (AIRCC), v. 5, n. 5, p. 95, 2013.

ROKACH, L. Decision forest: Twenty years of research. **Information Fusion**, Elsevier, v. 27, p. 111–125, 2016.

RONAO, C. A.; CHO, S.-B. Human activity recognition with smartphone sensors using deep learning neural networks. **Expert systems with applications**, Elsevier, v. 59, p. 235–244, 2016.

SAGI, O.; ROKACH, L. Approximating xgboost with an interpretable decision tree. **Information Sciences**, Elsevier, v. 572, p. 522–542, 2021.

SAINI, M.; ALELAIWI, A.; SADDIK, A. E. How close are we to realizing a pragmatic vanet solution? a meta-survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 48, n. 2, p. 1–40, 2015.

SAKAGUCHI, K.; HAUSTEIN, T.; BARBAROSSA, S.; STRINATI, E. C.; CLEMENTE, A.; DESTINO, G.; PÄRSSINEN, A.; KIM, I.; CHUNG, H.; KIM, J. *et al.* Where, when, and how mmwave is used in 5g and beyond. **IEICE Transactions on Electronics**, The Institute of Electronics, Information and Communication Engineers, v. 100, n. 10, p. 790–808, 2017.

SALEM, A. H.; DAMAJ, I. W.; MOUFTAH, H. T. Vehicle as a computational resource: Optimizing quality of experience for connected vehicles in a smart city. **Vehicular Communications**, Elsevier, v. 33, p. 100432, 2022.

SCHAPIRE, R. E. The boosting approach to machine learning: An overview. **Nonlinear estimation and classification**, Springer, p. 149–171, 2003.

SHABIR, B.; RAHMAN, A. U.; MALIK, A. W.; KHAN, M. A. On collective intellect for task offloading in vehicular fog paradigm. **IEEE Access**, IEEE, v. 10, p. 101445–101457, 2022.

SHEIKH, M. S.; LIANG, J. A comprehensive survey on vanet security services in traffic management system. **Wireless Communications and Mobile Computing**, Hindawi, v. 2019, 2019.

SHI, J.; DU, J.; WANG, J.; WANG, J.; YUAN, J. Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning. **IEEE Transactions on Vehicular Technology**, IEEE, v. 69, n. 12, p. 16067–16081, 2020.

SHIRAZ, M.; GANI, A.; SHAMIM, A.; KHAN, S.; AHMAD, R. W. Energy efficient computational offloading framework for mobile cloud computing. **Journal of Grid Computing**, Springer, v. 13, n. 1, p. 1–18, 2015.

SOUZA, A. B.; CELESTINO, J.; XAVIER, F. A.; OLIVEIRA, F. D.; PATEL, A.; LATIFI, M. Stable multicast trees based on ant colony optimization for vehicular ad hoc networks. In: IEEE. **The International Conference on Information Networking 2013 (ICOIN)**. [S.l.], 2013. p. 101–106.

SOUZA, A. B. D.; REGO, P. A.; CARNEIRO, T.; RODRIGUES, J. D. C.; FILHO, P. P. R.; SOUZA, J. N. D.; CHAMOLA, V.; ALBUQUERQUE, V. H. C. D.; SIKDAR, B. Computation offloading for vehicular environments: A survey. **IEEE Access**, IEEE, v. 8, p. 198214–198243, 2020.

SOUZA, A. B. de; REGO, P. A. L.; CARNEIRO, T.; ROCHA, P. H. G.; SOUZA, J. N. de. A context-oriented framework for computation offloading in vehicular edge computing using wave and 5g networks. **Vehicular Communications**, Elsevier, v. 32, p. 100389, 2021.

SOUZA, A. B. de; REGO, P. A. L.; ROCHA, P. H. G.; CARNEIRO, T.; SOUZA, J. N. de. A task offloading scheme for wave vehicular clouds and 5g mobile edge computing. In: IEEE. **GLOBECOM 2020-2020 IEEE Global Communications Conference**. [S.l.], 2020. p. 1–6.

SOUZA, A. M. de; VILLAS, L. A. Vem tranquilo: Rotas eficientes baseado na dinâmica urbana futura com deep learning e computação de borda. In: SBC. **Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2020. p. 351–364.

SU, J.; ZHANG, H. A fast decision tree learning algorithm. In: **Aaai**. [S.l.: s.n.], 2006. v. 6, p. 500–505.

SUN, P.; ALJERI, N.; BOUKERCHE, A. Dacon: A novel traffic prediction and data-highway-assisted content delivery protocol for intelligent vehicular networks. **IEEE Transactions on Sustainable Computing**, IEEE, v. 5, n. 4, p. 501–513, 2020.

TAHIR, M. N.; KATZ, M. Performance evaluation of ieee 802.11 p, lte and 5g in connected vehicles for cooperative awareness. **Engineering Reports**, Wiley Online Library, v. 4, n. 4, p. e12467, 2022.

TAUD, H.; MAS, J. Multilayer perceptron (mlp). In: **Geomatic approaches for modeling land change scenarios**. [S.l.]: Springer, 2018. p. 451–455.

TAUNK, K.; DE, S.; VERMA, S.; SWETAPADMA, A. A brief review of nearest neighbor algorithm for learning and classification. In: IEEE. **2019 International Conference on Intelligent Computing and Control Systems (ICCS)**. [S.l.], 2019. p. 1255–1260.

VIZZARI, D.; BAHRANI, N.; FULCO, G. Coexistence of energy harvesting roads and intelligent transportation systems (its). **Infrastructures**, v. 8, n. 1, 2023. ISSN 2412-3811. Disponível em: <https://www.mdpi.com/2412-3811/8/1/14>.

WAHEED, A.; SHAH, M. A.; MOHSIN, S. M.; KHAN, A.; MAPLE, C.; ASLAM, S.; SHAMSHIRBAND, S. A comprehensive review of computing paradigms, enabling computation offloading and task execution in vehicular networks. **IEEE Access**, IEEE, v. 10, p. 3580–3600, 2022.

WANG, C.-X.; HAIDER, F.; GAO, X.; YOU, X.-H.; YANG, Y.; YUAN, D.; AGGOUNE, H. M.; HAAS, H.; FLETCHER, S.; HEPSAYDIR, E. Cellular architecture and key technologies for 5g wireless communication networks. **IEEE communications magazine**, IEEE, v. 52, n. 2, p. 122–130, 2014.

WANG, Z.; ZHONG, Z.; ZHAO, D.; NI, M. Vehicle-based cloudlet relaying for mobile computation offloading. **IEEE Transactions on Vehicular Technology**, IEEE, v. 67, n. 11, p. 11181–11191, 2018.

WEINBERGER, K. Q.; SAUL, L. K. Distance metric learning for large margin nearest neighbor classification. **Journal of machine learning research**, v. 10, n. 2, 2009.

WU, X.; ZHAO, S.; DENG, H. Joint task assignment and resource allocation in vfc based on mobility prediction information. **Computer Communications**, Elsevier, v. 205, p. 24–34, 2023.

WU, X.; ZHAO, S.; ZHANG, R.; YANG, L. Mobility prediction-based joint task assignment and resource allocation in vehicular fog computing. In: IEEE. **2020 IEEE Wireless Communications and Networking Conference (WCNC)**. [S.l.], 2020. p. 1–6.

XIAO, L.; ZHUANG, W.; ZHOU, S.; CHEN, C. Learning while offloading: Task offloading in vehicular edge computing network. In: **Learning-based VANET Communication and Security Techniques**. [S.l.]: Springer, 2019. p. 49–77.

YE, M.; GUAN, L.; QUDDUS, M. Mpbrp-mobility prediction based routing protocol in vanets. In: IEEE. **2019 International Conference on Advanced Communication Technologies and Networking (CommNet)**. [S.l.], 2019. p. 1–7.

YI, S.; HAO, Z.; QIN, Z.; LI, Q. Fog computing: Platform and applications. In: IEEE. **2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)**. [S.l.], 2015. p. 73–78.

ZEADALLY, S.; GUERRERO, J.; CONTRERAS, J. A tutorial survey on vehicle-to-vehicle communications. **Telecommunication Systems**, Springer, v. 73, p. 469–489, 2020.

ZEBARI, G. M.; ZEBARI, D. A.; AL-ZEBARI, A. Fundamentals of 5g cellular networks: A review. **Journal of Information Technology and Informatics**, v. 1, n. 1, p. 1–5, 2021.

ZEKRI, A.; JIA, W. Heterogeneous vehicular communications: A comprehensive study. **Ad Hoc Networks**, Elsevier, v. 75, p. 52–79, 2018.

ZHANG, C.; ZHANG, Y.; SHI, X.; ALMPANIDIS, G.; FAN, G.; SHEN, X. On incremental learning for gradient boosting decision trees. **Neural Processing Letters**, Springer, v. 50, n. 1, p. 957–987, 2019.

ZHANG, F.; O'DONNELL, L. J. Support vector regression. In: **Machine Learning**. [S.l.]: Elsevier, 2020. p. 123–140.

ZHANG, J.; REN, M.; LABIOD, H.; KHOUKHI, L. Link duration prediction in vanets via adaboost. In: IEEE. **GLOBECOM 2017-2017 IEEE Global Communications Conference**. [S.l.], 2017. p. 1–6.

ZHANG, J.; WANG, F.-Y.; WANG, K.; LIN, W.-H.; XU, X.; CHEN, C. Data-driven intelligent transportation systems: A survey. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 12, n. 4, p. 1624–1639, 2011.

ZHENG, K.; ZHENG, Q.; CHATZIMISIOS, P.; XIANG, W.; ZHOU, Y. Heterogeneous vehicular networking: A survey on architecture, challenges, and solutions. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 4, p. 2377–2396, 2015.

ZHOU, H.; WANG, H.; CHEN, X.; LI, X.; XU, S. Data offloading techniques through vehicular ad hoc networks: A survey. **IEEE Access**, IEEE, v. 6, p. 65250–65259, 2018.

ZHOU, H.; XU, W.; CHEN, J.; WANG, W. Evolutionary v2x technologies toward the internet of vehicles: Challenges and opportunities. **Proceedings of the IEEE**, IEEE, v. 108, n. 2, p. 308–323, 2020.

ZHOU, Z.; LIU, P.; FENG, J.; ZHANG, Y.; MUMTAZ, S.; RODRIGUEZ, J. Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach. **IEEE Transactions on Vehicular Technology**, IEEE, v. 68, n. 4, p. 3113–3125, 2019.

ZOGHLAMI, C.; KACIMI, R.; DHAOU, R. 5g-enabled v2x communications for vulnerable road users safety applications: a review. **Wireless Networks**, Springer, v. 29, n. 3, p. 1237–1267, 2023.

APÊNDICE A – UTILIZAÇÃO DO NS-3 E SUMO

Código-fonte 1 – Importar trace SUMO.

```
1     NodeContainer c;
2     c.Create(numberOfNodes);
3
4     MobilityHelper mobility;
5     Ns2MobilityHelper ns2 = Ns2MobilityHelper (tracePath +
          "arquivo.tcl");
6     ns2.Install ();
7
8     Ptr<MobilityModel> model = c.Get(0)->GetObject<
          MobilityModel>();
9     Ptr<MobilityModel> model2 = c.Get(1)->GetObject<
          MobilityModel>();
10    double distance = GetDistance(model,model2);
11    double speed = GetSpeed(model);
12    double angle = GetAngle(model);
```

Código-fonte 2 – Utilizar dados do trace SUMO para realizar operações.

```
1
2  double GetSpeed(Ptr<const MobilityModel> model1){
3      return sqrt(pow(model1->GetVelocity().x, 2.0) +
4      pow(model1->GetVelocity().y, 2.0));
5  }
6
7  double GetDistance(Ptr<const MobilityModel> model1,
8  Ptr<const MobilityModel> model2){
9
10     double dist = sqrt(pow((model1->GetPosition().x -
11     model2->GetPosition().x), 2.0) + pow((model1->
```

```
        GetPosition().y - model2->GetPosition().y), 2.0));
    dist = round(dist / 10.0) * 10.0;
    return dist;
}

int GetAngle(Ptr<const MobilityModel> model1){
    double a;
    a = atan2(model1->GetVelocity().y,model1->GetVelocity()
        .x)*180/PI;
    return a;
}
```

APÊNDICE B – CÓDIGO DO PSEUDO-ÂNGULO

Código-fonte 3 – Pseudo-ângulo.

```
1   if (y >= 0){
2       if (x >= 0){
3           if (x >= y){
4               if(x > 0)
5                   return y / x;
6               else
7                   return 0;
8           }
9           return 2 - (x / y);
10      }
11
12      if ((x * -1) <= y)
13          return 2 + (x * -1) / y;
14      return 4 - y / (x * -1);
15  }
16
17  if (x < 0){
18      if ((x * -1) >= (y * -1))
19          return 4 + (y * -1) / (x * -1);
20      return 6 - (x * -1) / (y * -1);
21  }
22
23  if (x <= (y * -1))
24      return 6 + x / (y * -1);
25
26  return 8 - (y * -1) / x;
```

ANEXO A – TRACE SUMO

Código-fonte 4 – Parte de um trace utilizada nos cenários do SUMO

```
$node_(0) set X_ 2665.54
$node_(0) set Y_ 1556.04
$node_(0) set Z_ 0
$ns_ at 0.0 "$node_(0) setdest 2665.54 1556.04 0.00"
$node_(1) set X_ 1949.18
$node_(1) set Y_ 1764.72
$node_(1) set Z_ 0
$ns_ at 0.0 "$node_(1) setdest 1949.18 1764.72 0.00"
$node_(2) set X_ 2568.59
$node_(2) set Y_ 2429.49
$node_(2) set Z_ 0
$ns_ at 0.0 "$node_(2) setdest 2568.59 2429.49 0.00"
$ns_ at 1.0 "$node_(0) setdest 2666.77 1555.38 1.40"
$ns_ at 1.0 "$node_(1) setdest 1950.69 1763.91 1.71"
$node_(10) set X_ 3147.03
$node_(10) set Y_ 1704.48
$node_(10) set Z_ 0
$ns_ at 1.0 "$node_(10) setdest 3147.03 1704.48 0.00"
$node_(100) set X_ 3464.19
$node_(100) set Y_ 2110.77
$node_(100) set Z_ 0
$ns_ at 1.0 "$node_(100) setdest 3464.19 2110.77 0.00"
$node_(101) set X_ 3274.76
$node_(101) set Y_ 1790.18
$node_(101) set Z_ 0
$ns_ at 1.0 "$node_(101) setdest 3274.76 1790.18 0.00"
$node_(102) set X_ 2617.44
$node_(102) set Y_ 1581.91
$node_(102) set Z_ 0
```

```
$ns_ at 1.0 "$node_(102) setdest 2617.44 1581.91 0.00"
$node_(105) set X_ 2807.05
$node_(105) set Y_ 1073.4
$node_(105) set Z_ 0
$ns_ at 1.0 "$node_(105) setdest 2807.05 1073.4 0.00"
$node_(106) set X_ 2967.38
$node_(106) set Y_ 1838.71
$node_(106) set Z_ 0
$ns_ at 1.0 "$node_(106) setdest 2967.38 1838.71 0.00"
$node_(107) set X_ 2911.36
$node_(107) set Y_ 1272.38
$node_(107) set Z_ 0
$ns_ at 1.0 "$node_(107) setdest 2911.36 1272.38 0.00"
$node_(108) set X_ 1832.28
$node_(108) set Y_ 1649.07
$node_(108) set Z_ 0
$ns_ at 1.0 "$node_(108) setdest 1832.28 1649.07 0.00"
```