



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E
DE COMPUTAÇÃO



Treinamento Consciente de Quantização Vetorial Evolutiva: Um Método Adaptativo para Compactar Modelos de Aprendizado de Máquina

Thommas Kevin Sales Flores

Orientador: Prof. Dr. Ivanovitch Medeiros Dantas da Silva

Co-orientador: Prof. Dr. Daniel Gouveia Costa

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Processamento Inteligente da Informação) como parte dos requisitos para obtenção do título de Doutor em Ciências.

Número de ordem PPgEEC: D396
Natal, RN, fevereiro de 2026

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Flores, Thommas Kevin Sales.

Treinamento consciente de quantização vetorial evolutiva: um método adaptativo para compactar modelos de aprendizado de máquina / Thommas Kevin Sales Flores. - 2026.

152 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Natal, RN, 2026.

Orientação: Prof. Ivanovitch Medeiros Dantas da Silva.

Coorientação: Prof. Daniel Gouveia Costa.

1. Sistemas Embarcados - Tese. 2. TinyML - Tese. 3. Treinamento Consciente de Quantização - Tese. 4. Quantização Vetorial Evolutiva - Tese. I. Silva, Ivanovitch Medeiros Dantas da. II. Costa, Daniel Gouveia. III. Título.

RN/UF/BCZM

CDU 004.8(043.2)

*Dedico este trabalho a minha
família, em especial a meus pais,
minha esposa e meu filho.*

Agradecimentos

A Deus, por ter me concedido força, saúde e discernimento para cumprir minha missão e superar os desafios ao longo desta jornada.

À minha esposa, pela paciência, compreensão e apoio incondicional durante toda essa trajetória.

À minha mãe, por todo o amor, carinho e por jamais medir esforços para me proporcionar uma educação sólida. Seu exemplo é uma inspiração constante em minha vida.

Aos meus avós, pelo abrigo, cuidado e segurança ao longo dos anos, em especial à minha avó Walda Helena, que, embora não esteja mais entre nós, permanece viva em minhas lembranças e ensinamentos.

Ao meu pai e a todos os meus familiares, que, de diferentes formas, contribuíram para minha formação pessoal e acadêmica.

Ao meu orientador, Prof. Dr. Ivanovitch Medeiros Dantas da Silva, por ter me acolhido, acreditado em meu potencial e incentivado continuamente, além de proporcionar oportunidades valiosas de crescimento acadêmico e profissional.

Ao meu coorientador, Prof. Dr. Daniel Gouveia Costa, pela receptividade e pelas contribuições fundamentais durante o período de doutorado sanduíche no exterior.

Ao corpo docente do Programa de Pós-Graduação em Engenharia Elétrica e Computação (PPGEEC), pelas contribuições diretas e indiretas que foram essenciais para a construção desta tese.

À CAPES, ao CNPq e à FUNDEP, pelo apoio financeiro ao longo deste período, em especial durante o desenvolvimento da pesquisa no exterior.

Aos amigos que ganhei durante o doutorado: Morsinaldo, Breno, Mariane, Thiago Jesus, João Bittencourt e a todos os membros do grupo Conect2AI, pela parceria, apoio e troca de experiências ao longo dessa caminhada.

Resumo

A consolidação dos Grandes Modelos de Linguagem impõem desafios relevantes à sua implantação em dispositivos de borda, uma vez que a dependência de infraestruturas em nuvem compromete a privacidade dos dados, aumenta a latência e limita aplicações críticas em cenários com conectividade instável. As técnicas tradicionais de compressão, predominantemente aplicadas após o treinamento, mostram-se inadequadas nesse contexto, pois podem degradar o desempenho dos modelos ou dependem de cadeias de execução opacas, dificultando a análise e a verificação do comportamento do sistema. Diante desse cenário, esta tese propõe o Treinamento Consciente de Quantização Vetorial Evolutiva (EVQAT), cujo princípio central consiste em integrar a compressão ao próprio processo de aprendizado. Nessa abordagem, a quantização deixa de ser uma etapa posterior e passa a atuar como um mecanismo adaptativo durante o treinamento, permitindo que as transformações impostas aos parâmetros do modelo sejam explicitamente controladas e analisadas. A metodologia baseia-se na teoria de quantização vetorial evolutiva ao tratar os pesos do modelo de linguagem como fluxos contínuos de dados (*data streams*), possibilitando que algoritmos de agrupamento incremental, como AutoCloud, Mean-Shift e Affinity Propagation, ajustem dinamicamente os *codebooks* em sincronia com a otimização dos parâmetros de alta dimensão, viabilizando a compressão do modelo com controle explícito do erro de quantização. A integração da evolução dos *codebooks* ao treinamento assegura que o particionamento do espaço vetorial acompanhe a dinâmica do aprendizado, preservando propriedades estatísticas relevantes e reduzindo o impacto do ruído de quantização. Essa característica torna o EVQAT adequado para arquiteturas sensíveis à quantização, como os *Transformers*, ampliando sua aplicabilidade desde redes neurais multicamadas até modelos de linguagem generativos sob restrições de memória e computação. A proposta é avaliada em diferentes cenários experimentais, incluindo treinamento sem quantização, treinamento consciente de quantização nas modalidades vetorial evolutiva e inteiros de 8 bits (INT8), bem como quantização pós-treinamento nas mesmas variantes, com implantações em plataformas microcontroladas. Nesse contexto, a auditabilidade é devido a transparência no acesso aos parâmetros do modelo, como vocabulário, pesos de cada camada oculta e de atenção. Como resultado, a metodologia permite a geração automática de código C++ padrão e autossuficiente, com fluxo de inferência em tempo real e executável diretamente em diferentes arquiteturas de *hardware*, sem dependência de bibliotecas de terceiros ou sistemas operacionais, assegurando explicabilidade nos níveis do treinamento, do modelo e da execução, bem como interoperabilidade entre plataformas.

Palavras-chave: TinyML, Quantização Vetorial Evolutiva, Treinamento Consciente de Quantização, Sistemas Embarcados.

Abstract

The consolidation of Large Language Models poses significant challenges to their deployment on edge devices, as the reliance on cloud-based infrastructures compromises data privacy, increases latency, and limits critical applications in scenarios with unstable connectivity. Traditional compression techniques, predominantly applied after training, prove inadequate in this context, as they may degrade model performance or rely on opaque execution pipelines, hindering the analysis and verification of system behavior. In response to these challenges, this thesis proposes Evolving Vector Quantization-Aware Training (EVQAT), whose central principle consists of integrating compression directly into the learning process. In this approach, quantization is no longer treated as a post-training step but rather as an adaptive mechanism during training, allowing the transformations imposed on model parameters to be explicitly controlled and analyzed. The methodology is grounded in evolutionary vector quantization theory by treating the weights of the language model as continuous data streams, enabling incremental clustering algorithms such as AutoCloud, Mean-Shift, and Affinity Propagation to dynamically adjust the codebooks in synchrony with the optimization of high-dimensional parameters, thereby enabling model compression with explicit control over quantization error. The integration of codebook evolution into the training process ensures that the partitioning of the vector space follows the learning dynamics, preserving relevant statistical properties and reducing the impact of quantization noise. This characteristic makes EVQAT suitable for quantization-sensitive architectures such as Transformers, extending its applicability from multilayer neural networks to generative language models under strict memory and computational constraints. The proposed approach is evaluated across multiple experimental scenarios, including training without quantization, quantization-aware training using evolutionary vector quantization and 8-bit integer (INT8) representations, as well as post-training quantization under the same variants, with deployments on microcontroller-based platforms. In this context, auditability is achieved through transparent access to model parameters, including the vocabulary, the weights of each hidden layer, and the attention mechanisms. As a result, the methodology enables the automatic generation of standard, self-contained C++ code, with real-time inference capabilities and direct execution on diverse hardware architectures, without reliance on third-party libraries or operating systems, ensuring explainability at the levels of training, model, and execution, as well as interoperability across platforms.

Keywords: TinyML, Evolving Vector Quantization, Quantization-Aware Training, Embedded Systems.

Sumário

Sumário	i
Lista de Figuras	v
Lista de Tabelas	vii
Lista de Símbolos e Abreviaturas	ix
1 Introdução	1
1.1 Contextualização	1
1.2 Motivação	3
1.3 Problemática e Hipótese	4
1.4 Objetivos Gerais e Específicos	6
1.4.1 Objetivo Geral	6
1.4.2 Objetivos Específicos	6
1.5 Contribuições	7
1.6 Metodologia da Pesquisa	8
1.6.1 Natureza da Pesquisa	8
1.6.2 Abordagem Metodológica	9
1.6.3 Objetivos da Investigação	9
1.6.4 Procedimentos Técnicos	9
1.7 Estrutura da Tese	10
2 Trabalhos Relacionados	13
2.1 Metodologia de Pesquisa Bibliográfica	13
2.2 Modelos de Linguagem em Sistemas Embarcados	15
2.2.1 Avaliação Comparativa e Caracterização Empírica	15
2.2.2 Frameworks para Implantação em Microcontroladores	17
2.2.3 Otimização a Nível de Sistema e Hardware	18
2.2.4 Aplicações Inovadoras e Adaptação On-Device	19
2.2.5 Técnicas de Quantização e Compressão	20
2.2.6 Síntese e Lacunas de Pesquisa	22
3 Fundamentação Teórica	27
3.1 Taxonomia de Técnicas de Quantização	27
3.1.1 Nível de Precisão	28

3.1.2	Consciência da Arquitetura (<i>Hardware-Awareness</i>)	28
3.1.3	Transparência e Execução	29
3.1.4	Momento da Quantização	29
3.1.5	Representação Numérica	30
3.1.6	Granularidade e Dinâmica	30
3.1.7	Objeto Quantizado	30
3.2	Quantização Vetorial	31
3.2.1	Definição Formal	31
3.2.2	Processo de Treinamento e Codificação	32
3.3	Métodos de Agrupamento	33
3.3.1	AutoCloud	33
3.3.2	Mean-Shift	34
3.3.3	Affinity Propagation	35
3.3.4	DBStream	35
3.4	LLM e Transformers	36
3.4.1	GPT-2: Arquitetura e Componentes	40
4	Metodologia	45
4.1	Visão Geral	45
4.2	Aquisição de Dados	46
4.3	Extração de Conhecimento	47
4.3.1	Entrada	47
4.3.2	Segmentação Semântica	47
4.3.3	Geração Sintética de Perguntas e Respostas	48
4.3.4	Segmentação e Armazenamento dos dados	49
4.4	Pré-processamento e Tokenização	50
4.5	Arquitetura do Modelo	50
4.6	Treinamento do Modelo	52
4.6.1	Treinamento Padrão	52
4.6.2	Treinamento Consciente de Quantização Vetorial Evolutiva	55
4.6.3	Treinamento Consciente de Quantização INT8	56
4.7	Incremento e Condição de Término	57
4.8	Quantização Pós-Treinamento	57
4.8.1	Quantização Pós-Treinamento Vetorial Evolutiva	58
4.8.2	Quantização Pós-Treinamento INT8	58
4.9	Exportação dos Parâmetros do Modelo	59
4.10	Geração de Código	61
4.11	Implantação de Borda e Inferência	62
5	Planejamento dos Experimentos	65
5.1	Questões de Pesquisa	65
5.2	Cenários de Avaliação	66
5.3	Hiperparâmetros do Modelo	68
5.3.1	Cenário de Referência (FP32)	68

5.3.2	Cenário 1: Treinamento Consciente de Quantização Vetorial Evolutiva	68
5.3.3	Cenário 2: Treinamento Consciente de Quantização INT8	69
5.3.4	Cenário 3: Quantização Pós-Treinamento Evolutiva	70
5.3.5	Cenário 4: Quantização Pós-Treinamento INT8 (INT8-PTQ)	70
5.4	Protocolo e Métricas de Avaliação	70
5.4.1	Métricas de Otimização e Incerteza	70
5.4.2	Precisão Linguística e Qualidade Semântica	71
5.4.3	Eficiência Computacional e Sustentabilidade	71
5.4.4	Métricas de Inferência em Dispositivos Embarcados	72
5.5	Ambiente Experimental	72
5.5.1	<i>Hardware</i>	73
5.5.2	<i>Software</i>	73
6	Resultados	77
6.1	Cenário de Referência: Sem Quantização	77
6.1.1	Análise de Desempenho Preditivo	78
6.1.2	Análise de Compressão	78
6.1.3	Avaliação de Sustentabilidade e Consumo Energético	78
6.1.4	Implantação em Dispositivos Embarcados	80
6.2	Cenário 1: EVQAT	81
6.2.1	Análise de Desempenho Preditivo	82
6.2.2	Análise de Compressão	85
6.2.3	Avaliação de Sustentabilidade e Consumo Energético	86
6.2.4	Implantação em Dispositivos Embarcados	86
6.3	Cenário 2: QAT-INT8	88
6.3.1	Análise de Desempenho Preditivo	89
6.3.2	Análise de Compressão	89
6.3.3	Avaliação de Sustentabilidade e Consumo Energético	89
6.3.4	Implantação em Dispositivos Embarcados	90
6.4	Cenário 3: EVQ-PTQ	91
6.4.1	Análise de Desempenho Preditivo	91
6.4.2	Análise de Compressão	92
6.4.3	Avaliação de Sustentabilidade e Consumo Energético	94
6.4.4	Implantação em Dispositivos Embarcados	94
6.5	Cenário 4: INT8-PTQ	96
6.5.1	Análise de Desempenho Preditivo	96
6.5.2	Análise de Compressão	97
6.5.3	Avaliação de Sustentabilidade e Consumo Energético	99
6.5.4	Implantação em Dispositivos Embarcados	99
6.5.5	Implantação em Dispositivos Embarcados	100
6.6	Discussão Geral e Análise Comparativa	102
6.6.1	Análise Comparativa das Técnicas de Quantização	102

6.6.2	Impacto dos Métodos de Quantização no Desempenho de Sistemas Embarcados	104
6.6.3	Posicionamento Frente ao Estado da Arte	105
7	Conclusões	109
7.1	Trabalhos Futuros	112
7.2	Artigos Publicados Durante a Pesquisa	113
7.2.1	Artigos Relacionados Diretamente	113
7.2.2	Artigos Relacionados Transversalmente	114
	Referências bibliográficas	116

Lista de Figuras

3.1	Comparativo de Arquitetura (<i>Encoder-Decoder</i> vs. <i>Decoder-Only</i>). . . .	38
3.2	Detalhamento do Mecanismo de Atenção (<i>Scaled Dot-Product Attention</i>). . . .	39
4.1	Visão geral da metodologia proposta desde o treinamento até a implantação de um SLM em dispositivos de baixa potência.	46
4.2	Fluxo da ingestão do manual técnico (PDF), segmentação semântica controlada, geração de pares instrução–resposta via LLaMA 3.2 e estruturação final em JSON para treinamento do modelo.	47
4.3	Visão geral da arquitetura do modelo baseado em GPT proposto, mostrando o fluxo de processamento da consulta de entrada desde a tokenização até a geração de saída. O painel direito ilustra a estrutura interna de um Bloco <i>Transformer</i> com atenção multi-cabeça mascarada, normalização e sub-camadas <i>feed-forward</i>	51
4.4	Fluxograma do processo de treinamento da metodologia proposta, detalhando as etapas de inicialização, os três modos de quantização (Padrão, Vetorial Evolutiva e INT8) e o ciclo de otimização com atualização dos parâmetros mestres.	53
6.1	Desempenho do modelo base FP32 em diferentes configurações arquiteturais	79
6.2	Desempenho do EVQAT sob diferentes algoritmos de agrupamento	83
6.3	Evolução da perda e do número de grupos no EVQAT	84
6.4	Comparação da perda no treino e teste para EVQ-PTQ	93
6.5	Resultados consolidados do INT8-PTQ	98
6.6	Desempenho comparativo de modelos de linguagem pequenos após quantização, implantados em microcontroladores. Métricas incluem latência (associada ao uso de <i>Flash</i>), potência (associada ao uso de RAM) e configurações de quantização avaliadas.	105
6.7	Panorama comparativo dos modelos de EdgeAI do Estado da Arte.	108

Lista de Tabelas

2.1	Análise Comparativa de SLMs e Implantação em Dispositivos de Borda. .	24
3.1	Comparativo sintético das técnicas de quantização, correlacionando o momento de aplicação com a granularidade e o impacto na infraestrutura de <i>hardware</i>	31
3.2	Comparação de algoritmos de agrupamento considerando características-chave, como o tipo de método, a forma de determinação do número de grupos e a adequação a diferentes tipos de dados.	37
4.1	Exemplos representativos de pares de pergunta-resposta gerados a partir do manual técnico automotivo.	49
5.1	Espaço de busca de hiperparâmetros para definição da arquitetura base (Cenário de Referência).	68
5.2	Hiperparâmetros do processo de quantização vetorial evolutiva (Cenário 1).	69
5.3	Hiperparâmetros específicos dos algoritmos de agrupamento (EVQAT e EVQ-PTQ).	69
5.4	Microcontroladores considerados nos experimentos.	73
6.1	Resultados de implantação embarcada dos modelos FP32 selecionados no Cenário de Referência (pior caso e melhor desempenho).	80
6.2	Hiperparâmetros dos algoritmos de agrupamento evolutivo	82
6.3	Implantação dos modelos EVQAT	87
6.4	Implantação do modelo QAT-INT8	90
6.5	Implantação dos modelos EVQ-PTQ	95
6.6	Implantação dos modelos INT8-PTQ	100
6.7	Comparação entre métodos de quantização e modelo de referência (FP32).	103

Lista de Símbolos e Abreviaturas

ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
AIoT	Artificial Intelligence of Things
ARM	Advanced RISC Machines
ASPICE	Automotive Software Performance Improvement and Capacity dEtermination
AVR	Alf and Vegard's RISC
AWQ	Activation-Aware Weight Quantization
BERT	Bidirectional Encoder Representations from Transformers
BF16	Brain Floating Point 16-bit
BLEU	Bilingual Evaluation Understudy
CAN	Controller Area Network
CPU	Central Processing Unit
ECG	Electrocardiogram
EM	Exact Match
EVQ-PTQ	Quantização Pós-Treinamento Evolutiva
EVQAT	Evolving Vector Quantization-Aware Training (Treinamento Consciente de Quantização Vetorial Evolutiva)
FP16	Floating Point 16-bit
FP32	Floating Point 32-bit (Precisão de Ponto Flutuante de 32 bits)
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
GDPR	General Data Protection Regulation

GELU	Gaussian Error Linear Unit
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
IIoT	Internet das Coisas Industrial
INT2	Integer 2-bit
INT4	Integer 4-bit
INT8	Integer 8-bit
INT8-PTQ	Quantização Pós-Treinamento INT8
IoT	Internet das Coisas
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KDE	Kernel Density Estimation
LGPD	Lei Geral de Proteção de Dados
LLM	Large Language Model
LoRA	Low-Rank Adaptation
MAC	Multiply-Accumulate
MCU	Microcontroller Unit (Microcontrolador)
ML	Machine Learning
MLP	Multilayer Perceptron
MMLU	Massive Multitask Language Understanding
MoE	Mixture of Experts
NAS	Neural Architecture Search
NLP	Natural Language Processing
NPU	Neural Processing Unit
PPL	Perplexity
PTQ	Post-Training Quantization

QA	Question Answering
QAT	Quantization-Aware Training
QAT-INT8	Treinamento Consciente de Quantização para Inteiros de 8 bits
QLoRA	Quantized Low-Rank Adaptation
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction, Multiple Data
SLM	Small Language Model
SoC	System on Chip
SRAM	Static Random Access Memory
SVD	Singular Value Decomposition
TDP	Thermal Design Power
TinyML	Tiny Machine Learning
VQ	Vector Quantization
VQA	Visual Question Answering
WCET	Worst-Case Execution Time
β_1, β_2	Hiperparâmetros do otimizador Adam
η	Taxa de aprendizado (learning rate)
γ, β	Parâmetros de escala e deslocamento na normalização de camada
$\mathbf{c}_j$	Centróide ou palavra-código j
$\mathbf{E}_{\text{pos}}$	Matriz de embeddings posicionais
$\mathbf{E}_{\text{token}}$	Matriz de embeddings de tokens
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	Matrizes de Query, Key e Value
$\mathbf{W}$	Matriz de pesos
$\mathbf{x}_i$	Vetor de entrada i

$\mathcal{C}$	Codebook (conjunto de vetores prototípicos)
$\mathcal{L}$	Função de perda (loss function)
θ	Conjunto de parâmetros do modelo
d_k	Dimensionalidade das chaves (keys) no mecanismo de atenção
d_{model}	Dimensionalidade do embedding do modelo

Capítulo 1

Introdução

A crescente incorporação de Inteligência Artificial (IA) em sistemas embarcados tem redefinido o papel do processamento local de dados em aplicações críticas, especialmente, no domínio automotivo (Lasi et al. 2014, Thangaraju et al. 2024). Paralelamente ao avanço expressivo do Processamento de Linguagem Natural (PLN), impulsionado por arquiteturas profundas e modelos de larga escala (Vaswani et al. 2017, Brown et al. 2020, Naveed et al. 2025), observa-se um desalinhamento estrutural entre o poder computacional desses modelos e as severas restrições de recursos impostas por dispositivos de borda (Strubell et al. 2019, Schizas et al. 2022). Esta tese aborda esse desafio com uma abordagem metodológica inovadora: o *Evolving Vector Quantization-Aware Training* (EVQAT). O EVQAT representa uma mudança de paradigma, tratando a compressão não como uma etapa posterior, mas como um processo contínuo e integrado ao treinamento de modelos de aprendizado de máquina. Este capítulo estabelece as bases conceituais, científicas e metodológicas do trabalho, contextualizando o problema, motivações, hipóteses e objetivos, além de delinear as contribuições esperadas e a estrutura do documento.

1.1 Contextualização

A evolução recente do PLN foi marcada pela consolidação da arquitetura Transformer, que possibilitou o surgimento dos *Large Language Models* (LLMs) (Vaswani et al. 2017). Esses modelos demonstram desempenho notável em tarefas complexas, mas seu tamanho e consumo energético os tornam inviáveis para execução local em dispositivos com recursos limitados, isto é, plataformas embarcadas com RAM e *Flash* na ordem de kilobytes, baixo poder de processamento e limitação energética. (Strubell et al. 2019, Brown et al. 2020, Naveed et al. 2025). Estudos recentes demonstram que mesmo modelos de 7B parâmetros, quando quantizados para inteiro de 4 bits (INT4), enfrentam severas restrições de largura de banda de memória em dispositivos de borda como Raspberry Pi e Jetson Orin, com *throughput* limitado a poucos *tokens* por segundo (Dhar et al. 2024).

Setores como o automotivo, em plena transformação digital, demandam soluções de IA embarcadas que operem com baixa latência, alta confiabilidade e total privacidade de dados (Fleming 2008, Kim et al. 2024, Thangaraju et al. 2024, Lasi et al. 2014). A crescente complexidade dos sistemas veiculares modernos, com múltiplos domínios ele-

trônicos integrados, amplifica a necessidade de processamento inteligente local para diagnóstico, assistência ao motorista e interação natural com o usuário (Tami et al. 2024). No entanto, os microcontroladores (MCUs) típicos desses sistemas possuem apenas alguns megabytes de memória Flash e poucos kilobytes de RAM, criando um abismo tecnológico em relação aos requisitos dos modelos modernos. Mesmo os *Small Language Models* (SLMs), na faixa de 1-4B parâmetros, que emergem como alternativas mais leves, ainda demandam gigabytes de memória mesmo após a aplicação de quantização convencional. Dessa forma, permanecem inviáveis para execução local em microcontroladores, cuja memória, capacidade de processamento e orçamento energético são severamente limitados, exigindo o emprego de técnicas de compressão mais agressivas, que vão além da simples redução de precisão numérica e impõem reduções estruturais substanciais no modelo para alcançar ordens de grandeza de economia de memória. (Islam et al. 2024, Zhou et al. 2025, Zhang et al. 2024, Gross et al. 2026, Lu et al. 2025, Pham et al. 2025, Schizas et al. 2022).

Superar esse abismo exige técnica onde a eficiência e a adaptabilidade da compressão são mais críticas. Nesse contexto, esta tese apresenta o *Evolving Vector Quantization-Aware Training* (EVQAT). Diferentemente das técnicas estáticas de pós-treinamento, o EVQAT integra algoritmos de quantização vetorial evolutiva, como AutoCloud e Mean-Shift, diretamente ao ciclo de treinamento (Chen et al. 2025). Os pesos da rede são tratados como fluxos de dados contínuos, permitindo que os *codebooks*, definidos como conjuntos finitos de vetores prototípicos que representam e substituem os pesos originais, se ajustem dinamicamente à medida que o modelo aprende, minimizando o ruído de quantização e preservando a informação aprendida, isto é, reduzindo o erro introduzido quando valores contínuos (FP32) são aproximados por representações de menor precisão (Zheng et al. 1997). Essa abordagem difere fundamentalmente de métodos tradicionais, como o K-means estático, e se alinha conceitualmente a avanços recentes em quantização consciente de ativação e compressão adaptativa por camada, ao mesmo tempo em que introduz a novidade da evolução contínua dos *codebooks* durante o treinamento (Sanyal et al. 2024, Yu et al. 2024, Wu et al. 2019).

A metodologia EVQAT foi inicialmente concebida e validada em uma arquitetura fundamental, a *Multilayer Perceptron* (MLP), conforme documentado em (Flores, Costa & Silva 2025, Flores, Medeiros, Silva, Costa & Silva 2025). Essa validação preliminar em uma arquitetura mais simples confirmou a viabilidade dos princípios fundamentais da quantização vetorial evolutiva integrada ao treinamento. Esta tese avança significativamente a partir dessa base, focando na aplicação e avaliação do EVQAT em um cenário complexo e de alto impacto. O cerne desta investigação reside na aplicação da metodologia EVQAT a uma arquitetura Transformer, especificamente o *Generative Pre-trained Transformer 2* (GPT-2), utilizando como estudo de caso para validação um sistema de consulta a manual técnico automotivo. Esta transição de uma MLP para um Transformer no contexto de um problema real de borda representa o salto de complexidade e contribuição prática central deste trabalho, demonstrando a escalabilidade e a eficácia prática do método proposto, em linha com trabalhos recentes que exploram a implantação de *Transformers* miniaturizados em MCUs (Yang et al. 2023, Scherer, Rutishauser, Cavigelli & Benini 2024, Jung et al. 2025).

1.2 Motivação

Diversas motivações práticas e tecnológicas impulsionam esta pesquisa. Primeiramente, a experiência do usuário no domínio automotivo é frequentemente prejudicada pela dificuldade de acesso a informações técnicas. Manuais de veículos são documentos extensos e complexos, cuja consulta em situações críticas, como uma falha no veículo, requer uma solução rápida, local e que não dependa de conectividade instável. Estudos de caso recentes demonstram o potencial de LLMs para suporte a desenvolvimento e diagnóstico de sistemas embarcados, mas essas soluções tipicamente requerem recursos computacionais além das capacidades de MCUs automotivos convencionais (Cao et al. 2025, Englhardt et al. 2024).

Em segundo lugar, as soluções atuais baseadas em nuvem para assistentes de linguagem no automotivo apresentam limitações intrínsecas (Junior et al. 2024). A dependência de conectividade introduz latência incompatível com cenários de segurança crítica, torna o sistema indisponível em áreas remotas e levanta sérias preocupações quanto à privacidade e segurança dos dados (Yang et al. 2024, He et al. 2025). Em um contexto onde veículos processam dados sensíveis do usuário e informações de localização, a execução local de modelos de linguagem emerge como um requisito fundamental para conformidade com regulamentações como Lei Geral de Proteção de Dados (LGPD) e *General Data Protection Regulation* (GDPR). Soluções recentes que utilizam LLMs para interagir com manuais ainda dependem de conectividade ou recursos não disponíveis em dispositivos de borda, não atendendo aos requisitos de privacidade e disponibilidade offline essenciais para aplicações automotivas críticas (Park et al. 2024, Lee et al. 2023).

Terceiro, embora os SLMs surjam como uma alternativa mais leve, sua implantação direta em microcontroladores ainda é inviável sem técnicas de compressão profundas (Huang et al. 2024, Zhang et al. 2024, Gross et al. 2026, Liu et al. 2024, Hu et al. 2024). As avaliações comparativas recentes revelam que mesmo modelos compactos como TinyLlama (1.1B) e Phi-3-mini (3.8B) requerem 2-4GB de memória após quantização INT4, ordens de grandeza acima da capacidade típica de MCUs automotivos (1-8MB *Flash*, 256KB-2MB RAM). Métodos convencionais de quantização, como inteiro de 8 bits (INT8) e INT4, frequentemente não são suficientes para os requisitos de memória limitado (Kuzmin et al. 2022, Masud et al. 2024, Lu et al. 2025, Islam et al. 2024). Trabalhos pioneiros demonstraram a viabilidade de *Transformers* miniaturizados (263KB-6K parâmetros) em MCUs para tarefas específicas como reconhecimento de gestos e classificação de ECG (Yang et al. 2023, Busia et al. 2025), mas a transição para modelos de linguagem generativos com capacidade de raciocínio sobre conhecimento textual complexo permanece um desafio em aberto.

A Quantização Vetorial (VQ) tradicional promete taxas de compressão mais altas através da representação de múltiplos pesos por índices de *codebook*, isto é, um conjunto finito de vetores representativos, mas suas implementações clássicas baseadas em K-means são estáticas, sensíveis à inicialização e podem degradar significativamente a acurácia quando aplicadas a arquiteturas complexas (Gao, Li, Chen & Wang 2024, Wu et al. 2019, Gao, Li, Chen & Wang 2024). Estudos recentes sobre quantização de SLMs revelam que modelos menores apresentam sensibilidade fundamentalmente diferente à

compressão comparados a LLMs maiores, necessitando de abordagens adaptadas (Zhou et al. 2025). Técnicas avançadas como *Activation-Aware Weight Quantization* (AWQ) e quantização mista demonstraram compressão quase sem perdas para 4 bits em modelos grandes, mas sua eficácia para compressão ultra-agressiva (<2 bits) em SLMs ainda é limitada. A validação inicial do EVQAT em MLPs sugeriu que uma abordagem adaptativa e evolutiva poderia superar essas limitações, motivando sua aplicação ao desafio maior da compressão de *Transformers* para cenários reais (Flores, Costa & Silva 2025, Flores, Medeiros, Silva, Costa & Silva 2025).

Quarto, a transparência e a auditabilidade dos processos de compressão são críticas para domínios regulados, como o automotivo. Muitos *frameworks* modernos de otimização de modelos, como o TensorRT e o OpenVINO, adotam transformações proprietárias implementadas em *runtimes* opacos, entendidos como ambientes de execução que encapsulam decisões internas de otimização, mapeamento para hardware e gerenciamento de recursos, sem expor plenamente seus mecanismos ao desenvolvedor (NVIDIA Corporation 2024, Intel Corporation 2024). Embora efetivas em termos de desempenho, essas abordagens comprometem a rastreabilidade exigida por normas como a ISO 26262 e o ASPICE (*ISO 26262: Road Vehicles – Functional Safety* 2018). Em contraste, a geração de código C++ puro e autossuficiente pelo EVQAT, sem dependências de bibliotecas externas ou de *runtimes* opacos, alinha-se aos requisitos de certificação e facilita a auditoria por terceiros, um aspecto frequentemente negligenciado em trabalhos focados exclusivamente em desempenho (Scherer, Rutishauser, Cavigelli & Benini 2024, Liang et al. 2023).

Portanto, a motivação central desta tese é estender e aplicar a metodologia EVQAT, previamente validada em arquiteturas simples, ao domínio dos *Transformers*, avaliando sua capacidade de viabilizar modelos de linguagem compactos, eficientes e funcionais para um estudo de caso real no setor automotivo, priorizando a operação local, a privacidade, a transparência do processo e a conformidade com padrões industriais de segurança e qualidade.

1.3 Problemática e Hipótese

A implantação de modelos de linguagem baseados em arquitetura *Transformer* em microcontroladores (MCUs) de sistemas embarcados enfrenta uma contradição fundamental: a demanda computacional e de memória desses modelos é incompatível com os recursos limitados de hardware de borda típico do setor automotivo. Por exemplo, um modelo como o GPT-2, com 124 milhões de parâmetros, consome centenas de megabytes em precisão nativa em ponto flutuante de 32 bits (FP32), permanecendo inviável para os poucos megabytes de memória *Flash* disponíveis em MCUs mesmo após quantização convencional para INT8.

Técnicas de compressão agressiva são, portanto, indispensáveis. Contudo, métodos como a quantização escalar uniforme para baixíssima precisão (4 bits ou menos) frequentemente resultam em degradação severa da qualidade do modelo. Abordagens mais sofisticadas, como a Quantização Vetorial (VQ), oferecem maior potencial de compressão ao representar grupos de pesos por meio de índices que referenciam um dicionário compacto

(*codebook*). A eficácia da VQ tradicional é limitada pela natureza estática dos *codebooks* gerados por algoritmos como K-means, que não se adaptam à distribuição dinâmica e complexa dos parâmetros de uma rede neural durante o seu treinamento.

Esta tese aborda essa lacuna por meio do EVQAT, uma metodologia que integra algoritmos de agrupamento evolutivo diretamente no ciclo de otimização, permitindo que os *codebooks* se adaptem continuamente à distribuição dos pesos do modelo. Embora validada previamente em arquiteturas mais simples (MLPs), sua aplicação e eficácia em *Transformers* – arquiteturas que incluem mecanismos de atenção multi-cabeça, normalização de camada e *embeddings* de alta dimensão – constituem um problema de pesquisa em aberto. Adicionalmente, a maior parte das pesquisas sobre *Transformers* para MCUs foca em tarefas discriminativas (classificação), deixando uma lacuna significativa na implantação de modelos generativos de linguagem, os quais exigem a preservação de conhecimento semântico complexo e capacidade de geração coerente.

Para investigar sistematicamente essa problemática, os experimentos foram estruturados em cinco cenários que comparam técnicas de representação e compressão. Com base nessa estrutura, as seguintes questões de pesquisa (QPs) foram formuladas:

- **QP1 (Escalabilidade):** A metodologia EVQAT é escalável e eficaz quando aplicada à compressão de modelos baseados na arquitetura *Transformer* (GPT-2) para implantação em dispositivos de borda? Especificamente, os algoritmos evolutivos conseguem gerar *codebooks* adaptativos que preservam as representações aprendidas sob compressão?
- **QP2 (Desempenho versus Compressão):** Qual é a relação de compromisso entre taxa de compressão e preservação da acurácia em uma tarefa prática de *question-answering* quando se comparam as abordagens EVQAT (durante e pós-treinamento) com as técnicas de quantização INT8 (durante e pós-treinamento) e com o modelo de referência em FP32?
- **QP3 (Viabilidade de Implantação):** O *pipeline* EVQAT, que gera código C++ autossuficiente, viabiliza a implantação prática do modelo comprimido em MCUs representativos, atendendo aos limites rígidos de memória, tempo de resposta (latência e taxa de processamento) e consumo energético?
- **QP4 (Transparência e Conformidade):** Como a transparência e a auditabilidade inerentes ao *pipeline* EVQAT (quantização documentada, código-fonte aberto, ausência de dependências proprietárias) se comparam aos requisitos de certificação para sistemas automotivos críticos (ISO 26262, ASPICE)?

Com base nessas questões, e alinhadas com os cenários experimentais delineados, as seguintes hipóteses são propostas:

- **H1 (Escalabilidade e Eficácia):** A aplicação do EVQAT a um modelo GPT-2 durante o treinamento produzirá uma compressão eficiente com menor degradação de desempenho (perplexidade, acurácia em QA) em comparação com a aplicação da mesma técnica em modo pós-treinamento estático e com a quantização INT8, para taxas de compressão equivalentes.

- **H2 (Preservação da Acurácia):** Para uma taxa de compressão agressiva (superior a 10 vezes), o modelo comprimido com EVQAT (Cenário 2) manterá um nível de acurácia (medido por *Exact Match* e F1-score no conjunto de teste de QA) significativamente superior ao dos modelos comprimidos com INT8 e comparável a uma fração definida (pelo menos 85%) da acurácia do modelo de referência em FP32 (Cenário de Referência).
- **H3 (Viabilidade Prática):** O código C++ gerado para o modelo EVQAT permitirá sua execução dentro dos limites de memória (*Flash* e RAM) de MCUs como o Arduino Portenta H7 e ESP32, com latência de primeiro *token* inferior a 1 segundo e taxa de processamento compatível com interatividade básica (mais de 5 *tokens* por segundo), validando a viabilidade do estudo de caso.
- **H4 (Transparência):** A documentação completa do processo de quantização, a geração de código-fonte aberto e as ferramentas de análise proporcionadas pelo *framework* EVQAT oferecerão um nível de rastreabilidade e auditabilidade superior ao de *toolchains* proprietárias, facilitando a avaliação de conformidade com padrões de certificação.

1.4 Objetivos Gerais e Específicos

1.4.1 Objetivo Geral

Desenvolver e validar uma metodologia de treinamento consciente de quantização vetorial evolutiva (EVQAT) para viabilizar a compressão e a implantação de modelos de linguagem pequenos em microcontroladores com severas restrições de recursos computacionais, preservando a capacidade preditiva e oferecer transparência, auditabilidade e conformidade com requisitos de sistemas críticos.

1.4.2 Objetivos Específicos

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

1. Conceber e implementar um *framework* de quantização vetorial evolutiva integrado ao ciclo de treinamento de modelos baseados em arquitetura Transformer, permitindo a adaptação dinâmica de *codebooks* em sincronia com a otimização dos parâmetros do modelo, com o objetivo de mitigar o ruído de quantização e preservar representações semânticas relevantes.
2. Avaliar os algoritmos de agrupamento evolutivo e métricas de distância quanto à sua eficácia na geração de *codebooks* adaptativos, analisando o impacto da granularidade de quantização na capacidade de generalização e no desempenho preditivo dos modelos.

3. Comparar a metodologia proposta com técnicas consolidadas de quantização, incluindo treinamento consciente de quantização e quantização pós-treinamento, caracterizando os compromissos entre compressão, desempenho linguístico, custo computacional e consumo energético.
4. Desenvolver um pipeline automático de geração de código C++ determinístico, autossuficiente e portátil, capaz de traduzir modelos quantizados em implementações executáveis diretamente em microcontroladores, sem dependência de bibliotecas externas ou aceleradores proprietários.
5. Validar experimentalmente a implantação dos modelos comprimidos em diferentes arquiteturas de microcontroladores, mensurando latência de inferência, ocupação de memória, consumo energético e taxa de processamento, de modo a comprovar a viabilidade prática em ambientes com recursos restritos.
6. Demonstrar a aplicabilidade do *framework* proposto por meio de um estudo de caso no domínio automotivo, envolvendo a construção de um sistema de pergunta-resposta executável localmente, evidenciando a generalização da metodologia para cenários reais e regulados.

1.5 Contribuições

Esta tese apresenta contribuições científicas, metodológicas e tecnológicas que avançam o estado da arte na compressão de modelos de linguagem para sistemas embarcados sujeitos a restrições severas de memória, energia e latência. As contribuições abrangem desde a proposição de novas metodologias de quantização até a validação experimental em hardware real, integrando aspectos teóricos, experimentais e de engenharia, com ênfase em transparência, reprodutibilidade e aplicabilidade prática.

No âmbito metodológico, é proposta a metodologia de Treinamento Consciente de Quantização Vetorial Evolutiva, que integra algoritmos de agrupamento evolutivo diretamente ao ciclo de treinamento de modelos baseados em arquiteturas *Transformer*. Nessa abordagem, os pesos da rede neural são tratados como fluxos contínuos de dados, permitindo a adaptação dinâmica dos *codebooks* em sincronia com a retropropagação dos gradientes. Adicionalmente, é introduzida uma estratégia de subamostragem baseada em quantis, capaz de reduzir significativamente o custo computacional dos algoritmos de agrupamento sem comprometer a representatividade estatística dos parâmetros do modelo.

Do ponto de vista técnico e de engenharia, a tese apresenta um *pipeline* automatizado de geração de código-fonte em C++ autossuficiente, portátil e determinístico, capaz de converter modelos de linguagem quantizados em implementações executáveis diretamente em microcontroladores genéricos, sem dependências de bibliotecas externas, *runtimes* proprietários ou sistemas operacionais específicos. É também desenvolvida uma infraestrutura de *software* de código aberto que suporta todo o ciclo de vida de modelos de linguagem compactos, desde a aquisição e estruturação do conhecimento até a validação final em dispositivos embarcados. A viabilidade prática da abordagem é demonstrada por

meio da implantação em múltiplas arquiteturas de microcontroladores, com caracterização detalhada de latência, consumo energético e ocupação de memória.

No campo científico, a tese estabelece uma análise comparativa sistemática entre quantização durante o treinamento e quantização pós-treinamento estática, evidenciando diferenças relevantes na capacidade de generalização de modelos de linguagem compactos. São apresentadas análises do comportamento de diferentes algoritmos de agrupamento evolutivo e métricas de distância aplicados à quantização de pesos, estabelecendo relações entre granularidade dos *codebooks*, preservação semântica e custo computacional. Além disso, são fornecidas evidências empíricas de que a quantização pode atuar como um mecanismo de regularização implícita, contribuindo para a redução do sobreajuste mesmo quando aplicada após a convergência do modelo.

Em termos de aplicação, a tese valida a proposta em um estudo de caso no domínio automotivo, envolvendo a extração de conhecimento de documentação técnica e a construção de um sistema de *question–answering* executável localmente, atendendo a requisitos de privacidade de dados, operação *offline* e baixa latência. Embora validada nesse contexto, a arquitetura proposta é concebida de forma agnóstica ao domínio de aplicação, permitindo sua adoção em áreas como Internet das Coisas industrial, dispositivos médicos, sistemas educacionais embarcados e interfaces homem–máquina inteligentes.

Por fim, a tese contribui para a reprodutibilidade científica por meio da disponibilização pública de todo o *framework* desenvolvido, incluindo algoritmos de quantização, geradores de código C++, *scripts* de avaliação e ferramentas de análise. O trabalho também documenta o processo de geração de conjuntos de dados sintéticos para tarefas de *question–answering* a partir de documentação técnica, além de estabelecer métricas de referência de desempenho, latência, uso de memória e consumo energético para diferentes configurações de quantização e famílias de microcontroladores, fornecendo bases objetivas para trabalhos futuros em *TinyML* generativo.

1.6 Metodologia da Pesquisa

A definição de uma metodologia de pesquisa clara e rigorosa é essencial para assegurar a validade, a confiabilidade e a reprodutibilidade dos resultados obtidos. Esta pesquisa é caracterizada quanto à sua natureza, abordagem, objetivos e procedimentos técnicos, estruturando-se de forma a oferecer respostas empíricas e práticas às questões de pesquisa formuladas.

1.6.1 Natureza da Pesquisa

Esta pesquisa é de natureza aplicada, com foco na resolução de um problema concreto: a viabilização da implantação de um modelo *Transformer* para tarefas de *question–answering* em microcontroladores automotivos com recursos limitados. Parte de uma validação anterior do método EVQAT em arquiteturas MLP e estende sua aplicação para o domínio mais complexo dos *Transformers*, alinhando-se com os avanços recentes em modelos de linguagem compactos e sua execução em hardware de borda. A investigação

está diretamente orientada para os cenários de avaliação delineados, que comparam diferentes estratégias de quantização (EVQAT e INT8, durante e pós-treinamento) contra um modelo de referência sem compressão.

1.6.2 Abordagem Metodológica

A abordagem adotada é predominantemente quantitativa, fundamentada na mensuração objetiva e na comparação estatística de métricas de desempenho. Estas métricas incluem acurácia (perplexidade, F1-score, *Exact Match*), eficiência (consumo de memória *Flash* e RAM, latência e taxa de processamento) e consumo energético. A análise comparativa é realizada entre os cinco cenários experimentais estabelecidos, seguindo protocolos de avaliação consolidados na literatura para oferecer a comparabilidade dos resultados. Uma dimensão qualitativa complementar é incorporada para analisar a viabilidade prática de implantação, a usabilidade do código gerado e a transparência do processo de compressão, contextualizando os achados no domínio automotivo e em requisitos de conformidade regulatória.

1.6.3 Objetivos da Investigação

A pesquisa possui um duplo caráter: explicativo e de desenvolvimento tecnológico. É explicativa ao buscar estabelecer e validar empiricamente as relações entre a aplicação da metodologia EVQAT (com seus algoritmos evolutivos de geração de *codebooks*) e a preservação do desempenho em modelos *Transformer* submetidos a compressão. É de desenvolvimento tecnológico ao produzir artefatos validados: um *pipeline* funcional de compressão EVQAT para *Transformers*, um modelo GPT-2 comprimido e ajustado para *question-answering* automotivo, e código C++ executável, otimizado e auditável, pronto para implantação em MCUs.

1.6.4 Procedimentos Técnicos

A estratégia técnica combina uma revisão bibliográfica sistemática com uma pesquisa experimental configurada como um estudo de caso instrumental. A pesquisa bibliográfica, detalhada no Capítulo 2 (Trabalhos Relacionados), fundamentou o estado da arte e identificou as lacunas de pesquisa por meio de buscas estruturadas em bases de dados multidisciplinares (Scopus, IEEE Xplore, ACM Digital Library, arXiv).

A pesquisa experimental é estruturada como um estudo de caso único e instrumental, em que:

- **O Caso:** A implantação de um modelo de linguagem para consulta a manual técnico automotivo em um microcontrolador com restrições típicas do setor (2 a 8 MB de *Flash*, 256 KB a 2 MB de RAM);
- **A Unidade de Análise:** A aplicação da metodologia EVQAT a este caso, abrangendo todo o *pipeline* desde o ajuste fino do modelo até a geração de código executável e a validação de desempenho;

- **A Estratégia Experimental:** Implementação do conjunto de etapas do EVQAT para o modelo *Transformer* alvo (GPT-2). A estratégia envolve a condução de experimentos controlados que comparam os cinco cenários de avaliação, variando hiperparâmetros específicos (como algoritmo de agrupamento e granularidade de quantização). As medições sistemáticas cobrem desempenho (acurácia no domínio) e viabilidade de implantação (análise estática de memória, execução em *hardware*). Os resultados quantitativos e qualitativos são comparados com técnicas estabelecidas e emergentes do estado da arte;
- **Validade e Reprodutibilidade:** Para assegurar a validade interna, são utilizados conjuntos de dados de validação independentes, e múltiplas execuções com diferentes sementes aleatórias avaliam a variabilidade. Para oferecer a validade externa e a reprodutibilidade, todo o código-fonte, conjuntos de dados processados, modelos treinados e *scripts* de análise são disponibilizados publicamente com documentação detalhada, seguindo os princípios da ciência aberta e das boas práticas de pesquisa reproduzível em Aprendizado de Máquina.

1.7 Estrutura da Tese

Esta tese está organizada em sete capítulos, além desta introdução, os quais estão estruturados de forma a conduzir o leitor desde a contextualização do problema até a apresentação dos resultados e conclusões da pesquisa.

No Capítulo 1, é apresentada a introdução do trabalho, na qual são discutidos o contexto, a motivação, a problemática e as hipóteses da pesquisa, bem como os objetivos e a metodologia adotada, estabelecendo as bases conceituais e científicas da investigação.

O Capítulo 2 reúne os trabalhos relacionados, contextualizando o estado da arte sobre SLMs, técnicas de compressão, quantização e implantação de modelos de aprendizado de máquina em dispositivos embarcados e de borda. Ao final do capítulo, são identificadas as principais lacunas de pesquisa que motivam a proposta desta tese.

No Capítulo 3, é apresentado o referencial teórico que fundamenta o trabalho, abrangendo os conceitos de quantização vetorial, métodos de agrupamento evolutivo, arquiteturas *Transformer* e modelos de linguagem, com ênfase nos aspectos relevantes para compressão e implantação em sistemas embarcados.

O Capítulo 4 descreve detalhadamente a metodologia proposta, incluindo a aquisição e o pré-processamento dos dados, a arquitetura do modelo, os procedimentos de treinamento, as estratégias de quantização durante e após o treinamento, bem como o processo de geração de código e implantação em dispositivos de borda.

No Capítulo 5, é apresentado o planejamento experimental da pesquisa, no qual são definidos os objetivos dos experimentos, os cenários de avaliação, os hiperparâmetros explorados, as métricas adotadas e o ambiente experimental, permitindo a reprodução e a análise sistemática dos resultados.

O Capítulo 6 apresenta e discute os resultados obtidos nos diferentes cenários experimentais, analisando o impacto das técnicas de quantização propostas em termos de

desempenho, eficiência computacional, consumo energético e viabilidade de implantação em microcontroladores.

Por fim, o Capítulo 7 apresenta as conclusões da pesquisa, sintetizando as principais contribuições do trabalho, as publicações associadas, bem como as atividades futuras para a continuidade e consolidação da pesquisa.

Capítulo 2

Trabalhos Relacionados

Este capítulo contextualiza o uso de Inteligência Artificial aplicada a modelos de linguagem sob a perspectiva da eficiência computacional, reunindo e discutindo trabalhos que abordam técnicas de compressão, otimização e adaptação de modelos de linguagem para ambientes com recursos limitados. O foco recai sobre produções científicas que investigam estratégias para viabilizar inferência local e determinística de modelos de linguagem, com ênfase em *Small Language Models* (SLMs) e arquiteturas derivadas de *Large Language Models* (LLMs) projetadas para dispositivos de borda e microcontroladores.

A revisão prioriza estudos que exploram métodos estruturais e algorítmicos de redução de complexidade, incluindo quantização, poda, destilação de conhecimento, compartilhamento de parâmetros e redesenho arquitetural de *Transformers*, bem como trabalhos que analisam os impactos dessas técnicas em termos de uso de memória, latência, consumo energético e precisão. O capítulo inicia com a descrição do método de busca e seleção da literatura e, em seguida, apresenta a revisão temática organizada em cinco eixos principais: (i) técnicas de compressão e eficiência em modelos de linguagem; (ii) quantização, poda e aproximações de baixa ordem aplicadas a LLMs; (iii) arquiteturas compactas e redesenho de *Transformers* para *Edge AI*; (iv) implantação de modelos de linguagem em dispositivos embarcados e microcontroladores; e (v) síntese do capítulo e identificação das lacunas de pesquisa.

2.1 Metodologia de Pesquisa Bibliográfica

Para fundamentar esta pesquisa no estado da arte atual, foi realizada uma busca sistemática na literatura científica com foco em métodos de compressão, otimização e adaptação de LLMs para ambientes computacionalmente restritos. A base de dados Scopus foi adotada como principal fonte, em virtude de sua ampla cobertura multidisciplinar e reconhecimento pela comunidade científica. As estratégias de busca foram concebidas para identificar trabalhos situados na interseção entre modelos de linguagem compactos, técnicas de redução de complexidade e implantação eficiente em dispositivos de borda e microcontroladores.

A formulação das expressões de busca contemplou termos relacionados a compressão de modelos, eficiência computacional e inferência embarcada, incluindo palavras-chave como “model compression”, “quantization”, “pruning”, “knowledge distillation”, “low-

rank approximation”, “parameter sharing”, “neural architecture search”, “tiny language models”, “small language models”, “edge LLMs”, “on-device NLP” e “embedded transformers”, bem como suas variações e sinônimos. Esses termos foram combinados por meio de operadores booleanos e aplicados aos campos de título, resumo e palavras-chave, de modo a capturar estudos que abordassem tanto otimizações em nível de arquitetura quanto estratégias de compressão pós-treinamento.

```
TITLE-ABS-KEY (
  (
    ("large language model*" OR "LLM*" OR
     "small language model*" OR "SLM*" OR
     "tiny language model*" OR "compact language model*")
  )
  AND
  (
    "model compression" OR "model optimization" OR
    "quantization" OR "low-bit quantization" OR
    "pruning" OR "structured pruning" OR
    "knowledge distillation" OR
    "low-rank approximation" OR
    "evolving quantization" OR
    "neural architecture search" OR
    "efficient transformer*" OR
    "lightweight transformer*"
  )
  AND
  (
    "edge AI" OR "edge computing" OR
    "embedded system*" OR "microcontroller*" OR
    "resource-constrained device*" OR
    "on-device inference"
  )
)
```

Para complementar a busca principal, o Google Scholar foi utilizado com o objetivo de identificar publicações recentes, pré-prints e trabalhos ainda não indexados na Scopus, além de permitir o rastreamento de citações para mapear contribuições relevantes e tendências emergentes. Os manuscritos recuperados foram organizados utilizando a ferramenta Mendeley (Reiswig 2010) e submetidos a um processo de filtragem baseado em critérios de qualidade, relevância e aderência temática.

Na etapa inicial de seleção, foram excluídos trabalhos considerados de baixa qualidade, tais como estudos incompletos, mal documentados ou sem avaliação experimental consistente. Em seguida, os resumos e conclusões dos trabalhos remanescentes foram analisados de forma detalhada, com ênfase na identificação das técnicas de compressão empregadas e de seus impactos sobre memória, latência, consumo energético e precisão.

Técnicas de leitura dinâmica, como *skimming* e buscas direcionadas por palavras-chave, auxiliaram na eliminação de estudos fora do escopo, como aqueles focados exclusivamente em *hardware* especializado de alto desempenho ou em cenários dependentes de infraestrutura em nuvem.

A leitura integral dos trabalhos selecionados permitiu identificar contribuições centrais que exploram diferentes eixos de otimização, incluindo: (i) quantização agressiva (por exemplo, inteiros de 8 e 4 bits, e quantização híbrida), (ii) poda estrutural e não estrutural de parâmetros, (iii) destilação de conhecimento de modelos de grande porte para arquiteturas compactas, (iv) aproximações de baixa ordem e compartilhamento de pesos, e (v) redesenho arquitetural de *Transformers*, com ênfase em modelos rasos e estreitos, atenção eficiente e eliminação de componentes de alto custo computacional. Observou-se que, embora essas abordagens apresentem ganhos expressivos de eficiência, muitas dependem de *toolchains* complexos, quantização opaca ou aceleradores proprietários, o que pode comprometer a transparência e a reprodutibilidade em domínios sensíveis.

Por fim, a análise das listas de referências e de trabalhos correlatos dos autores mais relevantes permitiu ampliar a cobertura da revisão. Embora o levantamento tenha considerado um espectro amplo de técnicas de compressão para LLMs, este capítulo destaca prioritariamente os estudos mais recentes e diretamente relacionados à implantação de modelos de linguagem extremamente compactos em dispositivos de borda genéricos, ressaltando lacunas associadas à auditabilidade, portabilidade e execução em tempo real, aspectos que motivam a abordagem proposta neste trabalho.

2.2 Modelos de Linguagem em Sistemas Embarcados

A integração de LLMs e SLMs em dispositivos de borda com recursos restritos tornou-se uma área de pesquisa em rápida expansão, impulsionada pela necessidade de aplicações de Inteligência Artificial com baixa latência, preservação de privacidade e eficiência energética. A literatura pode ser agrupada em cinco direções principais: (i) avaliação comparativa e análise empírica, (ii) *frameworks* para implantação em microcontroladores, (iii) otimização a nível de sistema e *hardware*, (iv) aplicações inovadoras e adaptação *on-device*, e (v) técnicas de quantização e compressão.

2.2.1 Avaliação Comparativa e Caracterização Empírica

Estudos empíricos forneceram uma análise crítica sobre a viabilidade e relação de compromisso da execução de LLMs e SLMs em plataformas embarcadas. Recentes análises em 2024 demonstraram que, após quantizar modelos baseados em CPU para INT4 e utilizar *kernels* otimizados em assembly ARM, equipes reportaram latência de primeiro *token* abaixo de 1 segundo e velocidade de geração de aproximadamente 25 *tokens* por segundo, enquanto o modelo original em FP16 produzia menos de 1 *token* por segundo.

Dhar et al. (2024) implantaram LLaMA-2 (7B) usando quantização INT4 em dispositivos Raspberry Pi e Jetson Orin, demonstrando que a largura de banda de memória limitada é a principal restrição de desempenho. Seus experimentos revelaram que a taxa de processamento de geração de *tokens* é diretamente proporcional à largura de banda da

memória RAM, com o Raspberry Pi 4B alcançando apenas 2,3 *tokens*/segundo devido ao gargalo de sua interface LPDDR4-3200, enquanto o Jetson Orin Nano, equipado com LPDDR5-6400, atingiu 8,7 *tokens*/segundo para o mesmo modelo quantizado.

Islam et al. (2024) compararam TinyLlama (1,1B), Phi-3 (3,8B) e Llama-3 (8B) em vários dispositivos de borda, identificando relações de compromisso importantes entre latência e perplexidade. Seus achados enfatizaram a importância da seleção de modelo baseada nas capacidades do dispositivo-alvo. O estudo mostrou que o Phi-3, apesar de ter 3,4× mais parâmetros que o TinyLlama, apresentou apenas 1,8× maior latência em dispositivos com 8 GB de RAM, devido à sua arquitetura otimizada e melhor localidade de *cache*. Adicionalmente, o trabalho demonstrou que a perplexidade do TinyLlama (9,2 no conjunto de dados C4) era significativamente superior à do Phi-3 (6,1), sugerindo que arquiteturas bem projetadas superam ganhos puramente quantitativos de parâmetros.

Lu et al. (2025) conduziram um dos estudos de comparação mais abrangentes, avaliando mais de sessenta SLMs com contagens de parâmetros variando de 100 milhões a 5 bilhões. Seus resultados revelaram que modelos menores e bem otimizados podem superar arquiteturas maiores da classe 7B em várias tarefas, particularmente quando consideradas as métricas combinadas de precisão, latência e consumo energético. Este trabalho destacou inovações arquiteturais como mecanismos de atenção eficientes (por exemplo, *grouped-query attention*), tamanhos de vocabulário otimizados (32k *tokens* vs. 50k em modelos maiores) e funções de ativação especializadas como SwiGLU e GELUTanh, que reduzem operações computacionais em até 15% comparado à GELU padrão. Os autores observaram que modelos como o SmoLLM-360M alcançaram 52,3% de precisão na avaliação comparativa ARC-Easy enquanto consumiam apenas 180 mW durante inferência em um Jetson Nano, comparado a 2,8W do LLaMA-7B para 58,7% de precisão na mesma tarefa.

Weyssow et al. (2025) estenderam esta análise avaliando estratégias de ajuste fino eficiente em parâmetros (PEFT), incluindo *Low-Rank Adaptation* (LoRA), *Retrieval-Augmented Generation* (RAG) e destilação de conhecimento, em diferentes plataformas móveis e embarcadas. Seu estudo forneceu diretrizes práticas para adaptação de modelos embarcados, mostrando que LoRA alcança o melhor equilíbrio entre qualidade de adaptação e sobrecarga computacional para cenários com recursos limitados. Especificamente, a aplicação de LoRA com *rank* $r = 8$ em apenas 5% das camadas do modelo resultou em adaptação comparável ao *fine-tuning* completo, enquanto reduziu o consumo de memória em 94% e o tempo de treinamento em 87%. Para RAG, os autores demonstraram que índices vetoriais quantizados (*Product Quantization* com 8 bits) mantêm 98,2% da qualidade de recuperação enquanto reduzem o consumo de memória do índice em 32×.

Trabalhos recentes introduziram SLMQuant, como o desenvolvido por Zhou et al. (2025), uma avaliação comparativa sistemática especificamente projetado para avaliar técnicas de quantização para SLMs, revelando disparidades fundamentais na sensibilidade à quantização entre SLMs e LLMs que necessitam de abordagens de compressão adaptadas para modelos menores. O estudo identificou que camadas de atenção em SLMs com menos de 1B de parâmetros apresentam 3,2× maior sensibilidade a *outliers* de ativação comparado a LLMs de 7B+ parâmetros, exigindo técnicas como SmoothQuant ou AWQ mesmo para quantização de 8 bits. Adicionalmente, a avaliação comparativa re-

velou que esquemas de quantização por-canal são insuficientes para SLMs, necessitando granularidade por-token ou até por-head nas camadas de atenção para manter degradação de perplexidade abaixo de 5%.

2.2.2 Frameworks para Implantação em Microcontroladores

Microcontroladores impõem restrições extremas tanto em memória quanto em computação, exigindo *frameworks* especializados e *pipelines* de otimização. Yang et al. (2023) demonstraram execução de *Transformers* em MCUs através do desenvolvimento do Tiny-Transformer, um modelo de 263 kB otimizado com quantização INT8 e *kernels* eficientes especificamente projetados para processadores ARM Cortex-M. O modelo alcançou 87,3% de precisão em reconhecimento de 10 gestos com latência de 42 ms em um STM32H7 (480 MHz, 2 MB *Flash*, 1 MB RAM), representando uma redução de 15,8× em tamanho comparado à versão original em ponto flutuante. Ainda neste artigo, os autores implementaram técnicas de *loop unrolling* e vetorização SIMD (*Single Instruction, Multiple Data*) usando instruções ARM NEON, resultando em aceleração de 3,4× nas operações de multiplicação matriz-vetor.

Lin et al. (2023) propuseram MCUBERT, que utiliza aproximações de baixa ordem e busca de arquitetura neural, do inglês *Neural Architecture Search* (NAS), para comprimir BERT-mini para dispositivos classe MCU. Sua abordagem alcançou compressão de 4× com menos de 2% de perda de precisão em tarefas de compreensão textual. O método emprega decomposição SVD (*Singular Value Decomposition*) truncada nas matrizes de projeção de atenção, mantendo apenas os k maiores valores singulares onde k é determinado por um limiar de energia retida de 95%. A busca de arquitetura otimizou simultaneamente a profundidade do modelo (número de camadas), largura (dimensão oculta) e estrutura de atenção (número de cabeças), resultando em uma arquitetura final com 6 camadas, dimensão oculta de 256 e 4 cabeças de atenção, comparado ao BERT-mini original de 12 camadas, dimensão 512 e 8 cabeças.

Liang et al. (2023) introduziram MCUFormer, usando agendamento consciente de memória e NAS de disparo único para implantar *Transformers* de visão em dispositivos STM32 com apenas 2 MB de RAM. O *framework* implementa uma estratégia de *swap* inteligente entre SRAM e *Flash*, mantendo apenas os pesos da camada atualmente em execução na SRAM rápida (240 ciclos de acesso) enquanto os demais residem em *Flash* (7 ciclos de acesso). Esta técnica, combinada com recomputação seletiva de ativações intermediárias, permitiu executar um ViT (*Vision Transformer*) com 12 camadas e 192 canais, processando imagens 64×64 em 340 ms com pico de uso de SRAM de apenas 1,8 MB.

Scherer, Rutishauser, Cavigelli & Benini (2024) apresentaram Deeploy, uma infraestrutura de compilador para implantação de SLMs como TinyStories em MCUs heterogêneos com alocação estática de memória e estratégias otimizadas de *cache KV* (*key-value*). Deeploy suporta processadores RISC-V com aceleradores de unidade de processamento neural (NPU), alcançando acelerações significativas através de *co-design hardware-software*. O compilador realiza análise de vida útil de tensores em tempo de compilação, gerando um plano de alocação de memória que minimiza fragmentação e

elimina a sobrecarga de alocação dinâmica. Para *cache* KV, o sistema implementa quantização assimétrica de 4 bits nos valores K e V, reduzindo o uso de *cache* em 4× com degradação de perplexidade inferior a 0,3 pontos. Em um MCU Siracusa (RISC-V RV32 + NPU de 8 MACs), Deeploy alcançou a taxa de processamento de 18,7 *tokens*/segundo para um modelo TinyStories de 42M parâmetros.

Jung et al. (2025) e Busia et al. (2025) alcançaram melhorias adicionais de eficiência através de atenção com pesos fundidos (*fused-weight attention*) e técnicas de *tiling* em profundidade (*depth-first tiling*). Seu trabalho demonstrou inferência em tempo real com consumo energético mínimo em MCUs GAP9 e STM32H7, permitindo aplicações como classificação de arritmia em eletrocardiograma (ECG) diretamente em dispositivos eletrônicos vestíveis (*wearables*) médicos. A técnica de *fused-weight attention* combina as matrizes de projeção W_Q , W_K e W_V em uma única matriz de pesos W_{QKV} de dimensão $[d_{model}, 3 \times d_{model}]$, reduzindo operações de carregamento de memória em 66%. O *depth-first tiling* processa cada camada completamente antes de avançar para a próxima, maximizando reuso de *cache* L1 e minimizando transferências para SRAM. Em um GAP9 operando a 370 MHz, o sistema classificou 30 segundos de sinal ECG em 127 ms com consumo energético de 4,2 mJ, permitindo operação contínua por 18 dias com bateria de 250 mAh.

2.2.3 Otimização a Nível de Sistema e Hardware

Vários estudos focam em melhorar a sinergia entre design de modelo e execução em *hardware* através de estratégias de co-otimização. Huang et al. (2024) desenvolveram EdgeLLM, um acelerador heterogêneo CPU-FPGA que emprega precisão mista (FP16×INT4) e esparsidade estruturada para reduzir demandas computacionais. Seu sistema alcançou aceleração de 3× comparado à inferência apenas em Unidade Central de Processamento (CPU) para GLM-6B. A arquitetura utiliza uma estratégia de particionamento em que camadas de *embedding* e normalização executam em CPU (FP16 para preservar precisão), enquanto camadas *feed-forward* e de atenção são mapeadas para o *Field-Programmable Gate Array* (FPGA) com quantização INT4. A esparsidade estruturada é induzida através de poda 2:4 (mantém 2 valores a cada 4 consecutivos), alcançando 50% de esparsidade sem degradação significativa de qualidade. O sistema processa 1024 *tokens* em 387 ms consumindo 4,8 W, comparado a 1142 ms e 15 W na execução CPU-only.

Zhou et al. (2024) propuseram EdgeMoE, um motor de inferência para modelos esparsos de *Mixture-of-Experts* (MoE) que aplica quantização por especialista (*expert-wise*) e estratégias inteligentes de *cache*. Ao carregar apenas especialistas ativos na memória, EdgeMoE reduziu o uso de memória de pico em 60% mantendo a qualidade do modelo. O sistema implementa um preditor baseado em *look-ahead* que antecipa quais especialistas serão ativados nos próximos k *tokens* (tipicamente $k = 8$), iniciando pré-carregamento assíncrono da DRAM para SRAM enquanto os especialistas atuais ainda estão em execução. A quantização é aplicada de forma heterogênea, com especialistas mais frequentemente ativados mantidos em INT8 e especialistas raros quantizados para INT4, balanceando qualidade e eficiência. Em um Jetson TX2 com 8 GB RAM, EdgeMoE executou Mixtral-8×7B (47B parâmetros totais, 13B ativos) com pico de uso de memória de apenas 4,2 GB

e latência média de 143 ms/token.

Sanyal et al. (2024) introduziram EntroLLM, que combina quantização mista com codificação Huffman para reduzir sobrecarga de transferência de dados em *hardware* de borda, alcançando até 4× de compressão além da quantização INT4 padrão. O sistema analisa a distribuição de pesos quantizados e constrói árvores de Huffman adaptativas por camada, explorando o fato de que pesos quantizados exibem distribuições altamente não uniformes (por exemplo, 42% dos pesos INT4 concentram-se em apenas 3 valores dos 16 possíveis). A codificação é aplicada *offline* e os pesos são descomprimidos durante inferência usando *buffers* de *streaming* de 4 KB. Em um Jetson Orin Nano executando Phi-3-mini (3.8B), EntroLLM reduziu o requisito de memória do modelo de 2,1 GB (INT4) para 587 MB (INT4+Huffman), com custo computacional adicional de decodificação de apenas 8,3ms por camada.

Avanços recentes em aceleração baseada em FPGA mostraram resultados promissores para inferência de LLMs. Chen et al. (2024) demonstraram que a aceleração espacial em FPGA pode alcançar desempenho competitivo comparado a soluções GPU enquanto consome significativamente menos energia, tornando FPGAs atrativos para cenários de borda onde eficiência energética é fundamental. O estudo comparou FPGAs Xilinx Alveo U280 com GPUs NVIDIA A100 executando LLaMA-7B, mostrando que o FPGA alcançou 87% da taxa de processamento da GPU (34,7 vs. 39,9 *tokens/s*) enquanto consumiu apenas 38% da potência (48 W vs. 127 W), resultando em eficiência energética 2,3× superior (0,72 vs. 0,31 *tokens/J*).

2.2.4 Aplicações Inovadoras e Adaptação On-Device

Trabalhos emergentes estendem SLMs para domínios multimodais e adaptativos, demonstrando a versatilidade de modelos comprimidos. Yao et al. (2025) apresentaram MiniCPM-V, um modelo visão-linguagem que executa em *smartphones* com quantização de 4 bits e alcança desempenho competitivo em tarefas de resposta a perguntas visuais. Seu trabalho mostrou que capacidades multimodais podem ser preservadas mesmo sob compressão agressiva. O modelo utiliza um *encoder* de visão compacto baseado em SigLIP com apenas 78M parâmetros, processando imagens 448×448 em 96 pacotes de 64×64. O *decoder* de linguagem (2,4B parâmetros) é quantizado para 4 bits usando GPTQ, resultando em um consumo total de 1,8 GB. Em avaliações de desempenho de VQA (*Visual Question Answering*), MiniCPM-V alcançou 72,1% no VQAv2 e 58,3% no TextVQA, comparável ao LLaVA-7B (73,9% e 59,1%) que requer 5,2 GB de memória.

Tami et al. (2024) propuseram HazardNet, um modelo compacto para detecção de segurança no tráfego em tempo real, demonstrando a aplicabilidade de SLMs a cenários de borda críticos para segurança. O modelo baseia-se em Phi-2 (2,7B parâmetros) ajustado com *Quantized Low-Rank Adaptation* (QLoRA) usando um *dataset* de 127k descrições de cenários de tráfego rotuladas com níveis de perigo. A abordagem QLoRA utiliza adaptadores de *rank* $r = 16$ com quantização *Normal Float 4-bit* (NF4) do modelo base, resultando em apenas 58 MB de parâmetros treináveis adicionais. O sistema alcançou *F1-score* de 0,89 na detecção de situações de alto risco, processando descrições textuais ou transcrições de áudio em 34ms em um Jetson Orin Nano, adequado para alertas em

tempo real em sistemas avançados de assistência ao motorista (ADAS).

Yu et al. (2024) exploraram estratégias de compressão adaptativa para modelos baseados em LLaMA, introduzindo compressão unificada por camada (*layer-wise unified compression*) que balanceia sensibilidade por camada com alvos globais de compressão. O método implementa um *framework* de otimização multiobjetivo que minimiza simultaneamente $\mathcal{L}_{task}$ (perda na tarefa *downstream*), $\mathcal{L}_{size}$ (tamanho total do modelo) e $\mathcal{L}_{latency}$ (latência de inferência). Para cada camada ℓ , o sistema determina uma taxa de compressão c_ℓ através de busca baseada em gradiente, começando com todas as camadas em $c_\ell = 1.0$ e iterativamente aumentando c_ℓ das camadas menos sensíveis até atingir o objetivo de compressão global. Em LLaMA-7B, a abordagem alcançou compressão de 4,3× com degradação de apenas 1,8 pontos de perplexidade, superando compressão uniforme (5,4 pontos de degradação) e métodos baseados em magnitude (3,2 pontos).

Gao, Zhang, Zhang, Gong & Guo (2024) desenvolveram FedPT, um *framework* de ajuste fino federado que permite treinamento colaborativo em *hardware* limitado através de *proxy-tuning* e quantização heterogênea. Este trabalho abriu novas possibilidades para melhoria colaborativa de modelos com preservação de privacidade em dispositivos de borda distribuídos. O sistema utiliza um modelo *proxy* pequeno (tipicamente 350M parâmetros) que é treinado localmente em cada dispositivo, com gradientes agregados via federação para atualizar um modelo maior hospedado em servidor. A quantização heterogênea permite que dispositivos com diferentes capacidades participem: dispositivos de alto desempenho *high-end* executam o *proxy* em FP16, *mid-tier* em INT8, e *low-end* em INT4. Em experimentos com 100 dispositivos simulados ajustando LLaMA-7B em tarefas de pergunta-resposta médica, FedPT alcançou 94,3% da precisão de *fine-tuning* centralizado enquanto manteve toda a comunicação criptografada e evitou compartilhamento de dados brutos.

Cao et al. (2025) e Englhardt et al. (2024) destacaram o potencial de LLMs para suportar desenvolvimento e depuração de *software* embarcado, criando ambientes de desenvolvimento assistidos por IA que executam inteiramente em *hardware* de borda. O sistema EmbedAgent utiliza um SLM de 1,3B parâmetros ajustado em 42k exemplos de código embarcado e *traces* de depuração. O modelo é integrado com RAG sobre documentação de MCU, *datasheets* e fóruns técnicos, mantendo um índice vetorial de 18 GB comprimido para 2,1 GB via *product quantization*. Em testes com desenvolvedores, o sistema reduziu tempo médio de depuração de *bugs* complexos em 38% e forneceu sugestões de correção com taxa de aceitação de 67% quando executado em um Raspberry Pi 4 com 8 GB RAM.

2.2.5 Técnicas de Quantização e Compressão

A quantização emergiu como uma das técnicas de compressão mais eficazes para permitir a implantação de LLMs em dispositivos de borda. Métodos de quantização pós-treinamento (PTQ - *Post-Training Quantization*) avançaram significativamente, com técnicas como AWQ (*Activation-Aware Weight Quantization*) e SmoothQuant alcançando compressão quase sem perdas para precisão de 4 bits. Esses métodos abordam o desafio de ativações *outliers* que podem degradar significativamente a qualidade do modelo sob

quantização ingênua.

O método AWQ identifica canais de pesos mais importantes para a saída do modelo através de análise de sensibilidade baseada em ativação. Especificamente, para um tensor de pesos $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ e ativações $\mathbf{X} \in \mathbb{R}^{B \times d_{in}}$, AWQ computa fatores de escala por canal $\mathbf{s} \in \mathbb{R}^{d_{in}}$ que minimizam o erro de quantização ponderado pela importância das ativações. Os canais correspondentes a *outliers* de ativação (tipicamente 0,1-1% dos canais) mantêm precisão mais alta (INT8 ou FP16) enquanto os demais são quantizados para INT4. Em LLaMA-7B, AWQ alcançou perplexidade de 5,47 no WikiText-2 (vs. 5,68 do *baseline* FP16), superando quantização uniforme INT4 que degradou para 8,23.

SmoothQuant adota uma abordagem complementar, migrando a dificuldade de quantização das ativações para os pesos através de transformações matemáticas equivalentes. O método introduz fatores de escala $\mathbf{s}$ que suavizam *outliers* de ativação enquanto compensam nos pesos: $\mathbf{Y} = (\mathbf{X} \text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s})\mathbf{W})$. Como pesos podem ser quantizados *offline* com maior precisão (calibração cuidadosa e *fine-tuning*), esta transformação permite quantização INT8 tanto de pesos quanto ativações com degradação mínima. Em OPT-175B, SmoothQuant manteve 99,3% da precisão *zero-shot* enquanto reduziu o requisito de memória de 350 GB para 44 GB.

A destilação de conhecimento provou-se eficaz para criar modelos compactos que retêm grande parte da capacidade de suas contrapartes maiores. Trabalhos recentes mostraram que modelos com menos de 4B de parâmetros, treinados através de destilação cuidadosa e exposição massiva a *tokens* (3+ trilhões de *tokens*), podem alcançar desempenho competitivo com modelos muito maiores (Santos et al. 2025). A série Phi-3 da Microsoft exemplifica esta tendência, com o Phi-3-mini de 3,8B parâmetros alcançando 69% no *Massive Multitask Language Understanding* (MMLU) através de treinamento intensivo ao invés de pura escala. O modelo foi treinado em 3,3 trilhões de *tokens* cuidadosamente filtrados, com ênfase em dados de alta qualidade (livros técnicos, código filtrado, problemas matemáticos sintéticos) ao invés de simplesmente maximizar volume de dados. A destilação utilizou GPT-4 como *teacher* em 180M exemplos de *chain-of-thought reasoning*.

Técnicas de poda evoluíram de abordagens simples baseadas em magnitude para métodos sofisticados de poda estruturada que removem cabeças de atenção inteiras, neurônios *feed-forward* ou até camadas completas de *Transformer* (Han et al. 2015). Avanços recentes em poda adaptativa baseada em flutuação (*fluctuation-based adaptive pruning*) permitem compressão dinâmica baseada na importância de camadas durante a inferência, otimizando a relação entre precisão e eficiência em tempo real (An et al. 2024). O método analisa a variância de ativações em um conjunto de calibração, identificando neurônios com baixa flutuação (contribuição consistentemente pequena) como candidatos à remoção. Ao contrário de métodos baseados em magnitude que analisam apenas os pesos, a abordagem baseada em flutuação captura a importância contextual dos neurônios. Em LLaMA-2-7B, poda de 40% dos neurônios FFN usando critério de flutuação degradou perplexidade em apenas 0,8 pontos, comparado a 2,3 pontos com poda por magnitude.

Abordagens híbridas combinando múltiplas técnicas de compressão mostraram-se particularmente promissoras. Estudos reportam que a aplicação sequencial de poda (compressão de 4×), destilação e quantização (INT8) pode alcançar taxas de compressão gerais

de 14× ou superiores enquanto limita a degradação de precisão a 2-3 pontos em tarefas *downstream*. O *pipeline* típico consiste em: (i) poda estruturada removendo 50-60% dos neurônios FFN e 25-30% das cabeças de atenção, (ii) *fine-tuning* do modelo podado por 5-10% dos passos de treinamento originais, (iii) destilação do modelo podado usando o modelo original como *teacher*, e (iv) quantização PTQ para INT8 ou inteiro de 4 bits (INT4). Em GPT-2-medium (355M parâmetros), este processo produziu um modelo de 26 MB (vs. 1,42 GB original) mantendo 92,7% da precisão em tarefas de compreensão de linguagem.

Cao & Aref (2026) propuseram retreinamento parcial consciente de saliência para quantização ultra-baixa (inteiro de 2 bits (INT2) e binária), identificando subconjuntos críticos de pesos que requerem precisão mais alta. O método computa mapas de saliência através do produto de gradientes $\nabla_{\mathbf{W}} \mathcal{L}$ com os próprios pesos $\mathbf{W}$, identificando 5-10% de pesos mais salientes que são mantidos em INT8 enquanto os demais são quantizados para INT2. Retreinamento *quantization-aware training* é aplicado por 2-3% dos passos originais. Em Mistral-7B, a abordagem alcançou quantização média de 2,4 bits/peso (90% em INT2, 10% em INT8) com degradação de 3,2 pontos de perplexidade, comparado a 8,7 pontos com quantização uniforme INT2.

Fang et al. (2025) introduziram auto-destilação através do tempo, onde o modelo serve como próprio *teacher* em diferentes estágios de geração autoregressiva. Durante treinamento, o modelo es exposto a prefixos de diferentes comprimentos, aprendendo a prever não apenas o próximo *token* mas também distribuições de *tokens* futuros. Em inferência, esta técnica permite geração especulativa com maior taxa de aceitação, acelerando a taxa de processamento em até 2,3× sem degradação de qualidade. O método é particularmente eficaz em SLMs onde o custo computacional de manter múltiplas cópias do modelo é proibitivo.

Hao et al. (2024) propuseram construção de modelos de qualquer formato, permitindo extração de sub-redes de diferentes tamanhos de um único modelo super-rede treinado. A super-rede é treinada com *dropout* estruturado estocástico, onde em cada passo de treinamento uma configuração aleatória (número de camadas, largura, número de cabeças) é amostrada e atualizada. Após treinamento, sub-redes de qualquer configuração dentro dos limites da super-rede podem ser extraídas sem *fine-tuning* adicional. Esta abordagem permite implantação adaptativa: dispositivos potentes extraem sub-redes maiores, enquanto dispositivos restritos utilizam configurações menores, tudo a partir do mesmo modelo base.

2.2.6 Síntese e Lacunas de Pesquisa

A Tabela 2.1 resume estudos representativos abordando a implantação de SLMs em dispositivos de borda. A tabela consolida informações em várias categorias, incluindo tamanho do modelo, estratégia de compressão, *hardware*-alvo e domínio de aplicação. Ela fornece uma visão unificada do progresso atual, desde miniaturização de *Transformers* para reconhecimento de gestos e diagnósticos médicos até modelos multimodais capazes de realizar inferência e tomada de decisão localmente.

Esta perspectiva comparativa destaca a diversidade de abordagens enquanto também

revela várias lacunas críticas de pesquisa:

1. **Ausência de procedimentos de avaliação padronizados:** Diferentes estudos usam métricas incompatíveis, plataformas de *hardware* e metodologias de avaliação comparativa, dificultando comparação direta. Enquanto Islam et al. (2024) reportam latência de primeiro *token*, Lu et al. (2025) focam em taxa de processamento médio, e Dhar et al. (2024) enfatizam consumo energético. A ausência de um *framework* unificado de avaliação impede meta-análises e dificulta a identificação de técnicas genuinamente superiores versus otimizações específicas de *hardware*.
2. **Transparência limitada na quantização:** Muitas abordagens dependem de ferramentas proprietárias ou esquemas de quantização opacos que comprometem reprodutibilidade e auditabilidade, particularmente preocupante para aplicações críticas ou reguladas. *Frameworks* comerciais como TensorRT e OpenVINO implementam otimizações de difícil compreensão que, embora efetivas, impossibilitam verificação independente. Em domínios como dispositivos médicos ou sistemas automotivos, esta opacidade é incompatível com requisitos regulatórios de validação e rastreabilidade.
3. **Foco insuficiente em execução determinística:** A maioria dos trabalhos prioriza desempenho de caso médio, negligenciando garantias de tempo de execução de pior caso essenciais para sistemas embarcados de tempo real. Técnicas como *cache* dinâmico, quantização adaptativa e poda baseada em flutuação introduzem variabilidade de latência que pode violar restrições temporais em aplicações críticas (Zhou et al. 2024, Yu et al. 2024, An et al. 2024).
4. **Dependência de hardware:** Muitas otimizações estão fortemente acopladas a aceleradores específicos (TPUs, NPUs customizadas) ou requerem recursos de *hardware* indisponíveis em microcontroladores genéricos. EdgeLLM requer FPGAs Xilinx com suporte a precisão mista, Deeploy é otimizado para RISC-V com NPU, e MiniCPM-V depende de NPUs móveis (Samsung Exynos, Qualcomm Hexagon). A portabilidade para MCUs genéricos baseados em ARM Cortex-M ou RISC-V sem aceleradores permanece desafiadora.
5. **Modelagem de consumo energético:** Embora latência e memória sejam bem estudadas, a perfilagem energética abrangente ao longo de todo o processo de inferência permanece subexplorada. Estudos tipicamente reportam potência média ou energia total, mas ignoram componentes críticos como custo energético de transferências SRAM $\leftrightarrow$ Flash (0,6nJ/byte no STM32L4), sobrecarga de gerenciamento de *clock* (PLL lock time: 2-5ms consumindo 5-15mA), e estados de baixo consumo entre inferências.
6. **Portabilidade limitada:** Soluções otimizadas para uma família de processadores (por exemplo, ARM Cortex-M) frequentemente não são transferíveis para outras arquiteturas (RISC-V, Xtensa, ARC). *Kernels* otimizados em assembly ARM NEON não funcionam em RISC-V, técnicas de *tiling* otimizadas para hierarquia de *cache*

ARM (L1: 32KB I+D, L2: 256KB) são subótimas em RISC-V com *cache* unificado menor, e instruções SIMD específicas de ARM não têm equivalentes diretos em todas as extensões RISC-V (Yang et al. 2023) .

7. **Auditabilidade e explicabilidade:** Modelos altamente comprimidos tornam-se “caixas-pretas” ainda mais opacas, dificultando inspeção e validação em domínios regulados. Quantização agressiva altera distribuições de ativação, poda remove caminhos de raciocínio, e destilação pode introduzir vieses não presentes no modelo *teacher*. Técnicas de interpretabilidade (*attention visualization*, *saliency maps*, LIME) desenvolvidas para modelos FP32 completos frequentemente falham ou produzem resultados enganosos quando aplicadas a modelos comprimidos, especialmente sob quantização INT4 ou inferior.

A síntese apresentada na Tabela 2.1 fundamenta a motivação para nosso *framework* de quantização evolutiva proposto, que visa alcançar reprodutibilidade, portabilidade e compressão extrema sob restrições severas de recursos, ao mesmo tempo em que aborda as lacunas identificadas através de estratégias de otimização transparentes e agnósticas a *hardware*. Especificamente, nossa abordagem fornece: (i) método de avaliação padronizado com métricas uniformes, (ii) quantização totalmente documentada com código-fonte aberto, (iii) análise de *Worst-Case Execution Time* (WCET) e garantias determinísticas, (iv) implementação portátil em C++ sem dependências de *hardware* específico e (v) modelo energético detalhado validado com medições físicas.

Tabela 2.1: Análise Comparativa de SLMs e Implantação em Dispositivos de Borda.

Referência	Modelo	Compressão / Otimização	Aplicação	Hardware
(Yang et al. 2023)	TinyTransformer (263 kB)	Quantização INT8, <i>kernels</i> otimizados ARM NEON	Reconhecimento de gestos (10 classes, 87.3% acc.)	STM32H7 (480MHz, 1MB RAM), GAP8
(Dhar et al. 2024)	LLaMA-2 (7B)	Quantização INT4	Análise de pegada de recursos e <i>bandwidth</i>	Jetson Orin Nano (8.7 tok/s), Raspberry Pi 4B (2.3 tok/s)
(Islam et al. 2024)	TinyLlama (1.1B), Phi-3 (3.8B), Llama-3 (8B)	Avaliação comparativa (4-bit/FP16)	Caracterização de latência vs. perplexidade	Raspberry Pi 5, Jetson Orin, Mac mini M2
(Lu et al. 2025)	60+ SLMs (100M–5B)	Quantização 4-bit, análise arquitetural	avaliação comparativa de eficiência	Jetson Orin, Smartphones (Android/iOS)
(Weyssow et al. 2025)	Vários SLMs	LoRA (r=8), RAG (PQ-8bit), destilação	Avaliação de estratégias PEFT	Plataformas móveis e embarcadas
(Zhou et al. 2025)	Vários SLMs (<4B)	avaliação comparativa sistemática de quantização	Análise de sensibilidade PTQ	Dispositivos de borda (geral)
(Lin et al. 2023)	BERT-mini comprimido (4× redução)	SVD <i>low-rank</i> , NAS	Compreensão textual (<2% perda)	MCUs genéricos (2MB RAM)

Continua na próxima página

Tabela 2.1: Análise Comparativa de SLMs e Implantação em Dispositivos de Borda. (continuação)

Referência	Modelo	Compressão / Otimização	Aplicação	Hardware
(Liang et al. 2023)	MCUFormer (ViT 12L-192C)	NAS <i>one-shot</i> , agendamento SRAM/Flash	Vision <i>Transformer</i> (64×64, 340ms)	STM32 (2MB RAM)
(Scherer, Rutishauser, Cavigelli & Benini 2024)	TinyStories (42M params)	Quantização INT4-KV, alocação estática	Inferência auto-regressiva (18.7 tok/s)	Siracusa MCU (RISC-V + NPU 8-MAC)
(Jung et al. 2025)	<i>Tiny Transformers encoder</i>	<i>Fused-weight attention, depth-first tiling</i>	Inferência TinyML geral	STM32H7 (480MHz), GAP9 (370MHz)
(Busia et al. 2025)	Tiny Transformer (6k params)	Quantização 8-bit, execução paralela	Classificação ECG (30s em 127ms, 4.2mJ)	GAP9 MCU (8-core RISC-V)
(Huang et al. 2024)	GLM-6B	FP16×INT4 misto, esparsidade 2:4	Aceleração heterogênea (3× vs CPU)	CPU + Xilinx FPGA (4.8W)
(Chen et al. 2024)	Vários LLMs (7B+)	Aceleração espacial FPGA	Análise de eficiência energética	Xilinx Alveo U280 (2.3× tok/J vs GPU)
(Zhou et al. 2024)	Mixtral-8×7B (47B total)	Quantização per-expert, <i>cache</i> preditivo	Inferência MoE (60% redução RAM)	Jetson Nano (4.2GB RPi 4B), TX2 (pico),
(Sanyal et al. 2024)	smoLLM (1.7B), Phi-3 (3.8B)	INT4 + codificação Huffman (4× extra)	Compressão para transferência	Jetson Orin Nano (587MB vs 2.1GB)
(Yao et al. 2025)	MiniCPM-V (2.4B LM + 78M vision)	Quantização 4-bit GPTQ, SigLIP encoder	VQA multimodal (72.1% VQAv2)	<i>Smartphones</i> (1.8GB <i>footprint</i>)
(Yu et al. 2024)	LLaMA-7B	Compressão <i>layer-wise</i> unificada, poda	Adaptação <i>on-device</i> (4.3× comp., 1.8 PPL)	Jetson TX2, <i>smartphones</i> Android
(Gao, Zhang, Zhang, Gong & Guo 2024)	GPT-2 (1.5B), LLaMA (7–30B)	<i>Proxy-tuning</i> federado (350M proxy)	<i>Fine-tuning</i> colaborativo (94.3% acc.)	100 dispositivos heterogêneos
(Tami et al. 2024)	Phi-2 ajustado (2.7B)	QLoRA (r=16, NF4), 58MB adaptadores	Detecção de perigo no tráfego (F1=0.89)	Jetson Orin Nano (34ms/inferência)
(Cao et al. 2025)	SLM custom (1.3B) + RAG	<i>Fine-tuning</i> 42k exemplos, PQ índice	Assistente de código embarcado	Raspberry Pi 4 (8GB), ESP32
(Englhardt et al. 2024)	Vários SLMs (<3B)	Adaptação para <i>debugging</i>	Depuração assistida por LLM	Dispositivos de desenvolvimento
(Wang et al. 2025)	<i>Survey</i> (100+ modelos)	Poda, quantização, destilação, NAS	Revisão sistemática de técnicas	N/A (análise teórica)
(An et al. 2024)	LLaMA-2-7B	Poda baseada em flutuação (40% FFN)	Compressão dinâmica (0.8 PPL)	Plataformas de inferência gerais

Continua na próxima página

Tabela 2.1: Análise Comparativa de SLMs e Implantação em Dispositivos de Borda. (continuação)

Referência	Modelo	Compressão / Otimização	Aplicação	Hardware
(Cao & Aref 2026)	Mistral-7B	Retreinamento consciente de saliência	Quantização <i>ultra-low</i> (2.4 bits médio)	Dispositivos res- tritos
(Fang et al. 2025)	Vários LLMs	Auto-destilação temporal	Geração especu- lativa (2.3× acele- ração)	Plataformas gerais
(Hao et al. 2024)	Super-rede adaptativa	<i>Dropout</i> estruturado esto- cástico	Extração de sub- redes <i>any-shape</i>	Vários dispositi- vos <i>edge</i>
(Liu et al. 2024)	MobileLLM (125–350M)	Redesenho arquitetural, <i>weight sharing</i>	Compreensão e geração <i>mobile</i>	<i>Smartphones, ta- blets</i>
(Hu et al. 2024)	MiniCPM (2.4B)	Treinamento 3.3T <i>tokens</i> , destilação GPT-4	Geração de texto geral (69% MMLU)	Dispositivos mó- veis
(Cheng & Siu 2024)	Modelos híbridos	Particionamento <i>edge- cloud</i> , fusão	Inferência cola- borativa adapta- tiva	Sistema hí- brido (latência variável)
(Santos et al. 2025)	Neural networks (geral)	Destilação <i>teacher-student</i> (seminal)	<i>Framework</i> teó- rico de destilação	N/A (trabalho fundamental)
(Han et al. 2015)	AlexNet, VGG-16	Poda + quantização + Huff- man (35–49×)	Compressão de CNNs (seminal)	CPU, GPU
Nossa proposta	GPT otimizado (<i>Decoder-only</i>)	Quantização transparente, WCET	Implantação certificável em MCU	ESP32, Portenta H7, 6 Pico, Nano 33 Ble Sense

Capítulo 3

Fundamentação Teórica

Este capítulo estabelece os fundamentos teóricos que sustentam a presente pesquisa. Considerando que este trabalho está situado na interseção de várias áreas, nomeadamente a quantização vetorial, o processamento de linguagem natural, a compressão de modelos de aprendizado profundo e a computação embarcada de baixo consumo, uma revisão abrangente dos conceitos nucleares de cada um desses domínios se torna necessária. O objetivo deste capítulo é fornecer o substrato teórico essencial para a compreensão da metodologia proposta, do desenho experimental e da interpretação dos resultados obtidos. A exposição está estruturada para apresentar, em primeiro lugar, a taxonomia de técnicas de quantização, em seguida são apresentados os conceitos da quantização vetorial e dos algoritmos de agrupamento evolutivo que a viabilizam. Por fim, são detalhadas as definições centrais da arquitetura *Transformer* e dos modelos de linguagem de grande escala, com foco específico no GPT-2.

3.1 Taxonomia de Técnicas de Quantização

A quantização de redes neurais constitui um mecanismo fundamental para viabilizar a inferência em cenários de *Edge AI*, nos quais restrições de energia, memória e capacidade computacional são particularmente severas. Diferentemente de abordagens clássicas de compressão de dados, a quantização em *Deep Learning* configura-se como um problema de otimização numérica, cujo objetivo central é reduzir a precisão dos parâmetros do modelo minimizando, ao mesmo tempo, a degradação da acurácia.

Nesse contexto, a quantização não deve ser compreendida como uma técnica monolítica, mas como um conjunto amplo de estratégias que diferem quanto ao momento de aplicação, ao tipo de representação numérica, ao nível de granularidade e à forma como interagem com a arquitetura de hardware subjacente. A literatura recente pode ser organizada por meio de uma taxonomia multidimensional, estruturada em oito eixos principais: nível de precisão, consciência de arquitetura, dependência de execução, momento de aplicação, representação numérica, granularidade, dinâmica de escala e objeto quantizado.

As subseções a seguir detalham cada um desses eixos, fornecendo uma visão sistemática das decisões de projeto envolvidas no processo de quantização.

3.1.1 Nível de Precisão

O nível de precisão estabelece como os valores numéricos associados aos tensores de pesos e ativações são representados internamente. Essa escolha impacta diretamente a largura de banda de memória, o consumo energético e a complexidade dos circuitos aritméticos, sendo, portanto, um dos fatores mais determinantes para o desempenho prático do modelo quantizado.

De maneira geral, as abordagens existentes podem ser organizadas em um espectro que vai desde representações de ponto flutuante reduzido até esquemas de quantização extrema, nos quais os parâmetros assumem apenas alguns poucos valores discretos. As principais categorias são descritas a seguir:

- **Ponto Flutuante Reduzido (FP16, BF16):** Mantém a estrutura da notação científica composta por sinal, expoente e mantissa, porém com menor largura de palavra. O formato *Brain Floating Point* (BF16) destaca-se por preservar os 8 bits de expoente do FP32, garantindo o mesmo alcance dinâmico, ainda que à custa de menor precisão na mantissa. Essa característica reduz problemas de *overflow* e *underflow*, favorecendo a estabilidade numérica em aceleradores especializados, como TPUs (Wang & Rubio-González 2024).
- **Ponto Fixo e Inteiro (INT8, INT4):** Converte valores reais contínuos em um conjunto discreto de inteiros por meio de uma transformação linear, simétrica ou assimétrica, definida por um fator de escala (S) e um ponto zero (Z), conforme a relação $r = S(q - Z)$. Entre essas representações, o formato INT8 tornou-se predominante em inferência, em virtude da elevada eficiência das unidades lógicas e aritméticas (ALUs) inteiras quando comparadas às de ponto flutuante (Jacob et al. 2018).
- **Baixa Precisão (Binária/Ternária):** Restringe os parâmetros do modelo a conjuntos extremamente reduzidos, como $\{-1, +1\}$ na quantização binária ou $\{-1, 0, +1\}$ na quantização ternária. Essa simplificação permite substituir operações de multiplicação por operações lógicas do tipo *XNOR* e contagem de bits (*popcount*), reduzindo drasticamente a complexidade computacional. Em contrapartida, tais abordagens impõem desafios significativos à preservação da acurácia (Rastegari et al. 2016).

3.1.2 Consciência da Arquitetura (*Hardware-Awareness*)

Embora a redução de precisão seja, em princípio, benéfica, o ganho real de desempenho de um modelo quantizado depende fortemente do alinhamento entre a técnica de quantização e as características do hardware alvo. Diferenças na disponibilidade de instruções vetoriais, aceleradores dedicados e hierarquia de memória podem tornar uma mesma estratégia eficiente em um dispositivo e ineficaz em outro.

Nesse sentido, as abordagens de quantização podem ser classificadas quanto ao grau de consciência da arquitetura de execução:

- **Hardware-Unaware (Agnóstica):** Otimiza métricas abstratas, como o número de operações de ponto flutuante (FLOPs) ou o tamanho do modelo em memória. Por não considerar restrições físicas, como alinhamento de dados ou suporte a instruções SIMD, tais abordagens podem não se traduzir em ganhos efetivos de latência.
- **Hardware-Aware (Consciente):** Incorpora restrições do hardware diretamente no processo de otimização. Técnicas baseadas em *Neural Architecture Search* (NAS) utilizam a latência medida no dispositivo como função objetivo, ajustando dinamicamente a precisão de cada camada conforme a capacidade das unidades de processamento disponíveis, como *tensor cores* ou DSPs (Wang et al. 2019).

3.1.3 Transparência e Execução

Outro aspecto relevante diz respeito à forma como o modelo quantizado é interpretado e executado no ambiente de destino. Essa escolha afeta a portabilidade do modelo, a sobrecarga de execução e a dependência de bibliotecas externas. Sob essa perspectiva, distinguem-se duas abordagens principais:

- **Runtime-Dependente:** Requer um interpretador especializado, como TFLite ou OpenVINO, que implemente operadores quantizados otimizados. Embora facilite o *deploy*, essa abordagem impõe dependência direta entre o formato do modelo e a versão do *runtime* (Jacob et al. 2018).
- **Compilação Direta (Transparente):** Utiliza compiladores de *Deep Learning*, como TVM ou MLIR, para gerar código de máquina específico para o hardware alvo. Essa estratégia elimina a sobrecarga de interpretação e permite otimizações mais agressivas (Chen et al. 2018).

3.1.4 Momento da Quantização

O momento em que a quantização é aplicada influencia diretamente a capacidade do modelo de acomodar o erro introduzido pela redução de precisão. De modo geral, quanto mais cedo esse erro é incorporado ao processo de otimização, maior tende a ser a robustez do modelo resultante. As principais abordagens são:

- **Pós-Treinamento (PTQ):** Aplica-se a modelos previamente treinados.
 - *Static:* Determina os parâmetros de quantização (S, Z) a partir de um conjunto de dados de calibração, fixando os intervalos dinâmicos antes da inferência.
 - *Dynamic:* Calcula as escalas das ativações em tempo de execução, melhorando a precisão em modelos com alta variabilidade dinâmica, como RNNs, ao custo de maior latência (Rokh et al. 2023).

- **Durante o Treinamento (QAT):** Insere operações de quantização simulada no grafo computacional durante o treinamento. O erro de arredondamento é considerado no cálculo do gradiente por meio do *Straight-Through Estimator*, permitindo que a rede ajuste seus pesos para mitigar os efeitos da baixa precisão (Jacob et al. 2018).

3.1.5 Representação Numérica

A representação numérica distingue-se pela cardinalidade do mapeamento entre valores reais e valores quantizados, influenciando diretamente o grau de compressão alcançável.

- **Quantização Escalar:** Trata cada elemento do tensor de forma independente. É amplamente utilizada em formatos INT8, devido à sua simplicidade e compatibilidade direta com hardware.
- **Quantização Vetorial:** Mapeia vetores de parâmetros para entradas em um dicionário (*codebook*), explorando correlações estruturais entre os pesos. Inclui variantes como *Product Quantization* (Jégou et al. 2011) e *Residual Quantization*.

3.1.6 Granularidade e Dinâmica

A granularidade define o escopo de compartilhamento dos parâmetros de quantização. Abordagens *per-tensor* utilizam uma única escala global, sendo computacionalmente eficientes, porém sensíveis a *outliers*. Já abordagens *per-channel*, comuns em CNNs, atribuem escalas distintas a cada filtro, isolando distribuições estatísticas divergentes.

Quanto à dinâmica, os parâmetros de quantização podem ser estáticos, definidos após calibração, ou adaptativos, ajustando-se à entrada ou sendo aprendidos durante o treinamento (Gongyo et al. 2025).

3.1.7 Objeto Quantizado

Por fim, a quantização pode ser aplicada a diferentes componentes do processo de aprendizado:

- **Pesos:** Reduz o tamanho de armazenamento do modelo (ROM) (Ge et al. 2017).
- **Ativações:** Reduz o consumo de memória RAM e tráfego de dados entre memória e processador. Essencial para aceleração, pois ativações de baixa precisão permitem o uso de unidades *Multiply-Accumulate* (MAC) inteiras (Jacob et al. 2018, Kim et al. 2021).
- **Gradientes:** Focado em acelerar o *treinamento* distribuído, reduzindo a largura de banda necessária para a comunicação entre nós em *clusters* de treinamento (Seide et al. 2014, Wen et al. 2017).

Dettmers et al. (2022) focaram especificamente na quantização de ativações e pesos em Transformers de larga escala, identificando o surgimento de *outliers* sistemáticos em ativações que dificultam a quantização INT8 tradicional. Para resolver este problema, propuseram uma decomposição de precisão mista que isola dimensões de *outliers* em multiplicação matricial de 16 bits enquanto mais de 99.9% dos valores são multiplicados em 8 bits.

A Tabela 3.1 sintetiza as principais características discutidas, contrastando as abordagens quanto ao momento de aplicação, granularidade e impacto sobre a infraestrutura de hardware. Observa-se que métodos escalares, como PTQ e QAT, predominam em cenários nos quais a latência é crítica, devido à sua correspondência direta com instruções inteiras suportadas pelo *hardware*.

Tabela 3.1: Comparativo sintético das técnicas de quantização, correlacionando o momento de aplicação com a granularidade e o impacto na infraestrutura de *hardware*.

Característica	Static PTQ	QAT (Padrão)	HAWQ / NAS	Quant. Vetorial
Momento	Pós-treinamento	Durante o treino	Híbrido	Durante o treino
Representação	Escalar (INT8)	Escalar (INT4/8)	Mista (Heterogênea)	Vetorial (Índices)
Granularidade Típica	Per-tensor / Channel	Per-channel	Per-block / Layer	Bloco/Vetorial
Foco de Otimização	Simplicidade de <i>deploy</i>	Acurácia em baixos bits	Latência real	Taxa de compressão
Custo Computacional	Baixo (Calibração)	Alto (Re-treino)	Muito Alto (Busca)	Alto (Decodificação)

Embora a quantização escalar ofereça um caminho direto para a aceleração, ela trata cada peso como uma variável independente, ignorando a estrutura correlacionada dos parâmetros da rede. Para superar esse limite, torna-se necessário explorar técnicas de Quantização Vetorial, nas quais a representação por *codebooks* e centróides permite atingir taxas de bits fracionárias por peso, redefinindo o compromisso entre compactação e fidelidade de representação.

3.2 Quantização Vetorial

A Quantização Vetorial, do inglês *Vector Quantization* (VQ), é uma técnica amplamente utilizada em compressão de dados e codificação de sinais, cujo objetivo é representar um conjunto de vetores de alta dimensionalidade por meio de um subconjunto reduzido de vetores, denominados palavras-código, que compõem um livro de códigos (*codebook*) C (Vasuki & Vanathi 2006). A função primária da VQ consiste em mapear vetores contínuos em representações discretas, possibilitando uma compressão eficiente dos dados. Nesse contexto, ao invés de armazenar ou transmitir diretamente os vetores de entrada, cada vetor é representado pela palavra-código mais próxima no livro de códigos, reduzindo significativamente a quantidade de informação a ser processada ou transmitida.

3.2.1 Definição Formal

Considere um conjunto de vetores de entrada $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, onde

$$\mathbf{x}_i \in \mathbb{R}^d, \quad i = 1, \dots, N. \quad (3.1)$$

O objetivo da Quantização Vetorial é determinar um livro de códigos $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_K\}$, tal que

$$\mathbf{c}_j \in \mathbb{R}^d, \quad j = 1, \dots, K, \quad (3.2)$$

com $K \ll N$. Cada vetor de entrada $\mathbf{x}_i$ é associado à palavra-código mais próxima $\mathbf{c}_j$, segundo uma métrica de distância $d(\mathbf{x}, \mathbf{c})$, usualmente a distância Euclidiana:

$$d(\mathbf{x}_i, \mathbf{c}_j) = \|\mathbf{x}_i - \mathbf{c}_j\|_2. \quad (3.3)$$

Define-se a função de quantização como

$$Q: \mathbb{R}^d \rightarrow \mathcal{C}, \quad (3.4)$$

a qual associa a cada vetor de entrada $\mathbf{x}_i$ à palavra-código do livro de códigos que minimiza a distância Euclidiana:

$$Q(\mathbf{x}_i) = \mathbf{c}_{j^*}, \quad \text{onde } j^* = \arg \min_{1 \leq j \leq K} \|\mathbf{x}_i - \mathbf{c}_j\|_2. \quad (3.5)$$

Dessa forma, cada vetor $\mathbf{x}_i \in \mathbb{R}^d$ é aproximado por sua respectiva palavra-código $Q(\mathbf{x}_i) \in \mathcal{C}$, promovendo a compressão dos dados originais. O erro de quantização, também denominado distorção média, é definido como:

$$E = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - Q(\mathbf{x}_i)\|_2^2. \quad (3.6)$$

O objetivo do processo de quantização vetorial consiste em minimizar a distorção E mantendo um livro de códigos de cardinalidade reduzida, estabelecendo um compromisso entre taxa de compressão e fidelidade da representação.

3.2.2 Processo de Treinamento e Codificação

O processo de treinamento da VQ envolve a geração do livro de códigos $\mathcal{C}$ a partir de um conjunto de vetores de treinamento. Uma vez que o livro de códigos tenha sido gerado, o processo de codificação consiste em associar cada vetor $\mathbf{x}_i$ ao índice j de sua palavra-código correspondente $\mathbf{c}_j$, resultando na compressão de dados. A decodificação é o processo inverso, no qual os índices j são utilizados para recuperar as palavras-código $\mathbf{c}_j$, oferecendo uma aproximação dos vetores originais.

Esta etapa crítica do processo de treinamento baseia-se no uso de algum método de agrupamento. Em resumo, agrupamento é uma técnica fundamental utilizada para agrupar pontos de dados similares em grupos com base em critérios específicos ou medidas de similaridade.

Existem muitos algoritmos de agrupamento na literatura, cada um com suas vantagens e desvantagens. Portanto, selecioná-los adequadamente é de importância primordial para

o processo de treinamento como um todo. Neste ponto, embora nossa abordagem não dependa de um algoritmo particular, selecionamos quatro métodos evolutivos populares diferentes para avaliar seu desempenho ao suportar o processo de quantização pretendido.

3.3 Métodos de Agrupamento

O objetivo principal dos algoritmos de agrupamento é particionar um conjunto de dados em grupos distintos, onde os pontos de dados dentro do mesmo grupo são mais similares entre si do que aqueles em outros grupos. Os métodos de agrupamento podem ser amplamente categorizados em vários tipos, incluindo abordagens baseadas em densidade, baseadas em centróides e baseadas em modelos, cada uma com seus próprios pontos fortes e aplicações únicos (Ezugwu et al. 2022).

Nesta seção, são apresentados e discutidos diferentes métodos de agrupamento amplamente utilizados na literatura, com ênfase em seus princípios de funcionamento, aplicações e principais vantagens. São considerados os métodos AutoCloud, uma abordagem baseada em densidade capaz de ajustar automaticamente o número de grupos em um conjunto de dados; Mean-Shift, um algoritmo não paramétrico que identifica agrupamentos a partir da estimativa da densidade; Affinity Propagation, que realiza a seleção de exemplares por meio da propagação de mensagens; *Density-Based Stream Clustering* (DBStream), desenvolvido para o processamento de fluxos de dados dinâmicos em tempo real.

Cada um desses métodos adota estratégias distintas para a identificação de grupos, as quais variam conforme a natureza dos dados e os requisitos específicos da tarefa de análise. Considerando que a abordagem proposta neste trabalho depende fortemente de técnicas de agrupamento, a escolha de métodos adequados torna-se um aspecto central, motivando a apresentação detalhada dos algoritmos adotados como referência.

3.3.1 AutoCloud

AutoCloud é um algoritmo de agrupamento baseado em densidade projetado para determinar automaticamente o número de grupos em um conjunto de dados sem exigir um número predefinido. O que diferencia o AutoCloud é sua capacidade de detectar grupos em distribuições de dados complexas modelando cada grupo como uma “nuvem” e explorando a densidade local dos pontos de dados (Bezerra et al. 2020).

Uma das etapas-chave no AutoCloud é a estimativa da densidade local em cada ponto de dados usando uma abordagem de estimação de densidade por *kernel*, do inglês *Kernel Density Estimation* (KDE). A função de densidade é definida como:

$$f(x_i) = \frac{1}{nh^d} \sum_{j=1}^n K\left(\frac{x_i - x_j}{h}\right) \quad (3.7)$$

onde:

- $f(x_i)$ é a densidade estimada no ponto x_i ;
- n é o número total de pontos de dados;

- h é o parâmetro de largura de banda (*bandwidth*), que controla a suavização;
- d é a dimensionalidade dos dados;
- K é a função kernel, tipicamente Gaussiana.

AutoCloud não apenas identifica regiões de alta densidade, mas também ajusta dinamicamente os parâmetros para detectar grupos de tamanhos e formas variados. Adicionalmente, o algoritmo pode utilizar métodos como análise de máxima verossimilhança para determinar o número ideal de grupos, minimizando uma função de custo. Outra característica notável do AutoCloud é sua capacidade de separar grupos sobrepostos, uma vez que analisa independentemente a densidade das nuvens, prevenindo a influência de grupos vizinhos.

Este método é particularmente útil para dados espaciais, onde os agrupamentos podem ter formas irregulares ou densidades variadas.

3.3.2 Mean-Shift

Mean-Shift é um algoritmo de agrupamento não-paramétrico, o que significa que não requer que o número de grupos seja especificado antes da execução. Baseado em uma técnica de KDE, o Mean-Shift detecta modos de densidade em um espaço de características e agrupa pontos de dados ao redor desses modos (Cheng 1995). Em cada iteração, ele calcula o centróide de um conjunto de pontos dentro de uma janela de vizinhança e desloca esta janela para uma região de maior densidade até que o processo convirja.

Em cada iteração, a posição de um ponto x_i é atualizada de acordo com a seguinte equação de média ponderada:

$$m(x_i) = \frac{\sum_{j=1}^n K(x_i - x_j) x_j}{\sum_{j=1}^n K(x_i - x_j)} \quad (3.8)$$

onde:

- $m(x_i)$ é o novo ponto central para a próxima iteração;
- K é a função kernel, que pode ser uma função Gaussiana: $K(x_i - x_j) = e^{-\frac{\|x_i - x_j\|^2}{2h^2}}$;
- h é o parâmetro de suavização (largura de banda).

O Mean-Shift move progressivamente os pontos de dados em direção a regiões de maior densidade até que atinjam os picos de densidade local, que se tornam os centros dos grupos. Este algoritmo é particularmente útil em espaços de características complexos e é aplicado em várias áreas como segmentação de imagens, rastreamento de objetos e análise de dados multivariados.

Do ponto de vista computacional, o algoritmo Mean-Shift pode apresentar elevado custo computacional, especialmente em conjuntos de dados de grande porte, uma vez que, a cada iteração, é necessária a avaliação dos vizinhos de cada amostra. Ainda assim,

sua robustez ao ruído e sua capacidade de identificar grupos com formas arbitrárias tornam o método particularmente adequado para dados com distribuições complexas e não gaussianas.

3.3.3 Affinity Propagation

Affinity Propagation é um algoritmo de agrupamento baseado em medidas de similaridade que não requer a definição prévia do número de grupos. Diferentemente dos métodos tradicionais, o algoritmo utiliza um mecanismo de troca de mensagens para selecionar automaticamente os exemplares, os quais atuam como representantes centrais de cada grupo (Gan et al. 2007). A similaridade entre dois pontos de dados x_i e x_j é representada pela matriz S , sendo definida pela seguinte medida:

$$S(i, j) = -\|x_i - x_j\|^2 \quad (3.9)$$

O algoritmo mantém duas mensagens: responsabilidade $r(i, k)$, que reflete quão apropriado é para o ponto k ser o exemplar para i , e disponibilidade $a(i, k)$, que indica quão apropriado é para i escolher k como seu exemplar. As responsabilidades e disponibilidades são atualizadas iterativamente usando as seguintes equações:

$$r(i, k) \leftarrow S(i, k) - \max_{k' \neq k} \{a(i, k') + S(i, k')\} \quad (3.10)$$

$$a(i, k) \leftarrow \min \left(0, r(k, k) + \sum_{i' \notin \{i, k\}} \max(0, r(i', k)) \right) \quad (3.11)$$

Essas mensagens são trocadas iterativamente entre os pontos até que o algoritmo atinja a convergência, resultando na seleção de exemplares que passam a representar os centros dos grupos. Diferentemente de algoritmos como o K-Means, o Affinity Propagation determina automaticamente o número de grupos a partir dos próprios dados, não sendo necessária a definição prévia dessa quantidade.

3.3.4 DBStream

O DBStream é um algoritmo de agrupamento projetado especificamente para o processamento de fluxos de dados dinâmicos e contínuos. Fundamentado em métodos baseados em densidade, o algoritmo estende os princípios do DBSCAN de modo a lidar com a evolução temporal dos dados, empregando uma combinação de microgrupos e uma função de decaimento temporal (Hahsler & Bolaños 2016).

Cada microgrupo C_i no DBStream é caracterizado por um centro c_i , um peso w_i e uma função de decaimento $f(t)$, responsável por reduzir gradualmente a relevância dos pontos ao longo do tempo. Essa função pode ser modelada como:

$$f(t) = e^{-\lambda(t-t_0)} \quad (3.12)$$

em que:

- t representa o tempo atual;
- t_0 corresponde ao instante em que o ponto foi incorporado ao microgrupo;
- λ é o parâmetro de decaimento, que controla a taxa de redução da relevância temporal.

À medida que novos dados são recebidos, o DBStream atualiza continuamente os microgrupos, que representam agrupamentos locais de pontos. Microgrupos cuja densidade decai abaixo de um determinado limiar são removidos, permitindo que o modelo se adapte a mudanças no padrão dos dados. A formação de grupos de maior escala ocorre por meio da agregação de microgrupos, definida com base em critérios de conectividade e proximidade.

Devido a essas características, o DBStream é particularmente adequado para cenários que envolvem fluxos contínuos de dados, tais como sistemas de recomendação, monitoramento em tempo real e detecção de anomalias. O algoritmo combina eficiência computacional e flexibilidade, viabilizando o processamento contínuo de grandes volumes de dados com adaptação dinâmica a padrões em constante evolução.

Para proporcionar uma visão mais clara dos métodos de agrupamento discutidos, a Tabela 3.2 apresenta uma síntese comparativa de suas principais características, princípios de funcionamento e aplicações. A comparação evidencia as diferentes estratégias adotadas por cada algoritmo para a formação de grupos, auxiliando na identificação do método mais adequado para uma determinada tarefa de agrupamento evolutivo.

Esta tabela comparativa evidencia a diversidade de abordagens existentes entre os métodos de agrupamento analisados. AutoCloud e Mean-Shift baseiam-se em estimativas de densidade, embora diferenciem-se quanto aos seus mecanismos internos e à sensibilidade aos parâmetros adotados. O Affinity Propagation destaca-se por empregar uma estratégia fundamentada na troca de mensagens entre pontos de dados, enquanto o DBStream foi concebido especificamente para lidar com os desafios impostos por fluxos de dados contínuos e evolutivos. Nesse contexto, a escolha do método mais apropriado depende fortemente da natureza dos dados, de suas dinâmicas temporais e das características desejadas dos grupos resultantes.

3.4 LLM e Transformers

As LLMs representam uma revolução no PLN, sendo capazes de compreender, gerar e manipular texto de forma surpreendentemente sofisticada. Estes modelos são baseados na arquitetura *Transformer*, introduzida por (Vaswani et al. 2017) no trabalho seminal “Attention is All You Need”, que substituiu arquiteturas recorrentes tradicionais por mecanismos de atenção, permitindo processamento paralelo e captura de dependências de longo alcance de forma mais eficiente.

Arquitetura Transformer

A arquitetura *Transformers* é composta por dois componentes principais: o *encoder* (codificador) e o *decoder* (decodificador). No entanto, muitos LLMs modernos, como

Tabela 3.2: Comparação de algoritmos de agrupamento considerando características-chave, como o tipo de método, a forma de determinação do número de grupos e a adequação a diferentes tipos de dados.

Característica	AutoCloud	Mean-Shift	Affinity Propagation	DBStream
Tipo de Método	Baseado em Densidade	Baseado em Densidade	Baseado em Similaridade	Baseado em Densidade
Número de <i>clusters</i>	Automático	Automático	Automático	Dinâmico
Princípio-Chave	KDE; Modelagem de Nuvens	KDE; Busca de Modos	Troca de Mensagens	<i>Microclusters</i> ; Decaimento
Parâmetros	Largura de Banda (h)	Largura de Banda (h)	Similaridade; <i>Damping</i>	Decaimento (λ)
Densidade Variável	Sim	Sim	Sensível	Sim (adaptativo)
Formas Arbitrárias	Sim	Sim	Limitado	Sim
Tipo de Dados	Estático; Espacial	Estático; Espacial	Estático; Pequeno/médio	<i>Streams</i> ; Evolutivo
Pontos Fortes	Detecção automática; Sobreposição; Ruído	Sem k ; Robusto; Garantias teóricas	Exemplares; Não-métricas; Qualidade	Tempo real; <i>Concept drift</i> ; Memória
Limitações	Bandwidth; Gaussiano; $O(n^2)$	Pesado; <i>Tuning</i> ; Não hierárquico	$O(n^2)$; Convergência; Preferências	<i>Thresholds</i> ; <i>Tuning</i> ; <i>Management</i>

o *Generative Pre-trained Transformer 2* (GPT-2) (Radford et al. 2019), utilizam apenas a parte decodificadora da arquitetura, focando na geração autorregressiva de texto. A Figura 3.1 ilustra essa distinção arquitetural, contrastando o modelo completo original com a estrutura *Decoder-only* adotada pelos modelos da família GPT.

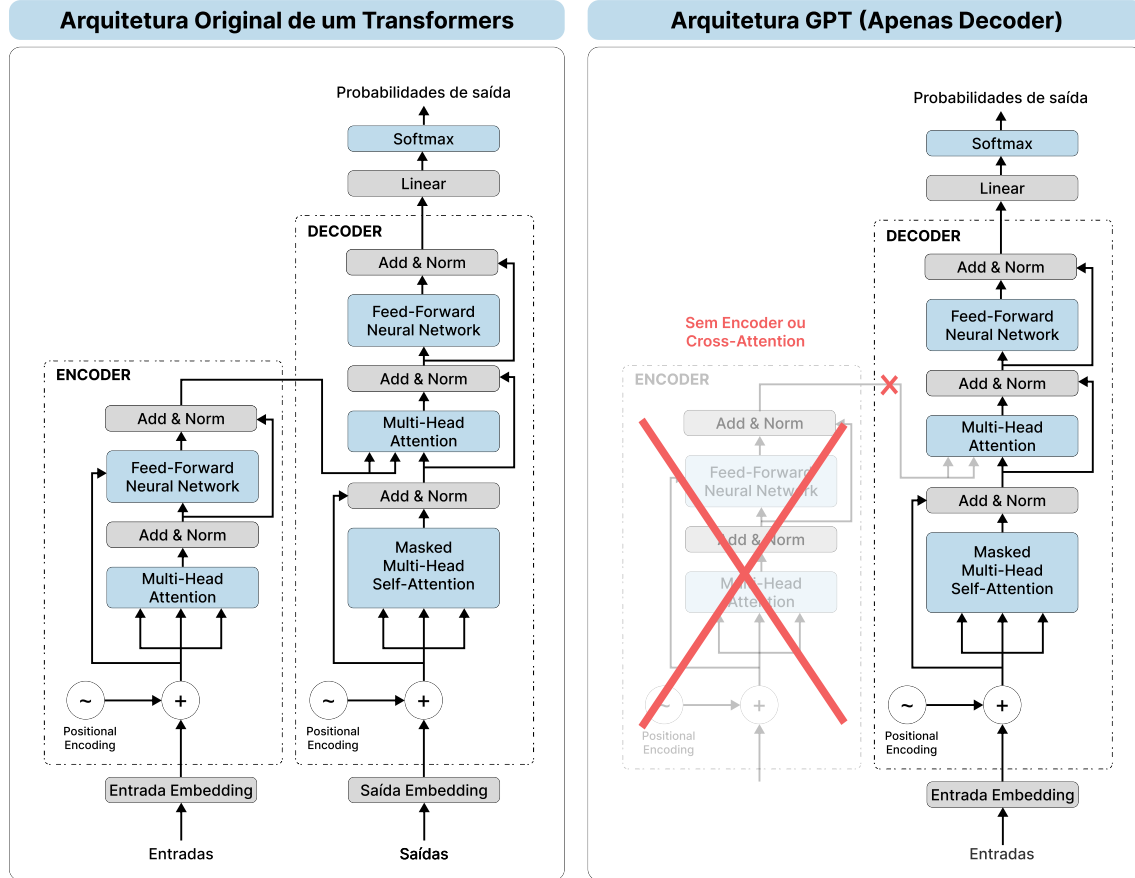


Figura 3.1: Comparativo de Arquitetura (*Encoder-Decoder* vs. *Decoder-Only*).

O componente fundamental dos *Transformers* é o mecanismo de atenção, que permite ao modelo ponderar a importância de diferentes partes da entrada ao processar cada elemento. O fluxo de dados deste mecanismo, especificamente a *Scaled Dot-Product Attention*, é detalhado na Figura 3.2, demonstrando visualmente as operações matriciais descritas pela equação:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.13)$$

onde:

- Q (*Query*) é a matriz de consultas com dimensão $[n \times d_k]$;
- K (*Key*) é a matriz de chaves com dimensão $[m \times d_k]$;
- V (*Value*) é a matriz de valores com dimensão $[m \times d_v]$;

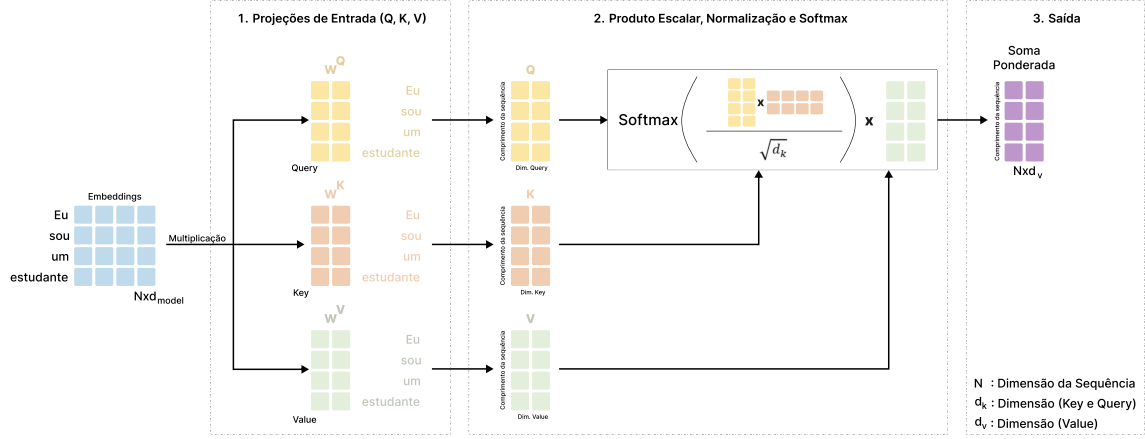


Figura 3.2: Detalhamento do Mecanismo de Atenção (*Scaled Dot-Product Attention*).

- d_k é a dimensionalidade das chaves (usado para normalização);
- n é o comprimento da sequência de consulta;
- m é o comprimento da sequência de chave/valor.

O fator de escala $\frac{1}{\sqrt{d_k}}$ é utilizado para evitar que o produto escalar cresça excessivamente para grandes valores de d_k , o que poderia resultar em gradientes muito pequenos após a aplicação da função *softmax*.

A atenção multi-cabeça permite que o modelo capture diferentes tipos de relações simultaneamente, sendo definida como:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (3.14)$$

onde cada cabeça de atenção é calculada como:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (3.15)$$

Aqui, $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ e $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ são matrizes de projeção aprendidas, e h é o número de cabeças de atenção.

Em modelos autorregressivos como o GPT-2, é essencial que cada posição na sequência atenda apenas às posições anteriores, impedindo que o modelo “veja o futuro” durante o treinamento. Isso é implementado através da atenção causal mascarada:

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \quad (3.16)$$

onde M é uma matriz de máscara com valores $M_{ij} = 0$ se $i \geq j$ (permitindo atenção) e $M_{ij} = -\infty$ se $i < j$ (bloqueando atenção futura).

Após a camada de atenção, cada posição passa por uma rede *feed-forward* de posição específica (*position-wise feed-forward network*), que é aplicada identicamente a cada posição:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3.17)$$

onde $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, e d_{ff} é tipicamente $4 \times d_{\text{model}}$. A função $\max(0, \cdot)$ representa a ativação ReLU. No GPT-2, utiliza-se a função GELU (*Gaussian Error Linear Unit*) (Hendrycks & Gimpel 2016):

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2} \left[1 + \text{erf} \left(\frac{x}{\sqrt{2}} \right) \right] \quad (3.18)$$

onde $\Phi(x)$ é a função de distribuição cumulativa da distribuição normal padrão e erf é a função erro.

Cada sub-camada (atenção e *feed-forward*) é envolvida por uma conexão residual seguida de normalização de camada (*layer normalization*) (Zhang & Sennrich 2019):

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \quad (3.19)$$

A normalização de camada é definida como:

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (3.20)$$

onde:

- $\mu = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} x_i$ é a média;
- $\sigma^2 = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} (x_i - \mu)^2$ é a variância;
- γ e β são parâmetros aprendíveis;
- ϵ é um termo pequeno para estabilidade numérica;
- $\odot$ denota multiplicação elemento a elemento.

3.4.1 GPT-2: Arquitetura e Componentes

O GPT-2, desenvolvido pela OpenAI (Radford et al. 2019), é um modelo *Transformer decoder-only* com até 1,5 bilhão de parâmetros. Sua arquitetura consiste em:

A representação inicial de cada *token* é obtida pela soma do embedding do token e do embedding posicional. Seja x o índice do token no vocabulário e p sua posição na sequência, com $x \in \{1, \dots, |\mathcal{V}|\}$ e $p \in \{1, \dots, L\}$. Os embeddings são definidos como funções em $\mathbb{R}^{d_{\text{model}}}$, resultando em:

$$\mathbf{E}_{\text{token}} \in \mathbb{R}^{|\mathcal{V}| \times d_{\text{model}}}, \quad \mathbf{E}_{\text{pos}} \in \mathbb{R}^{L \times d_{\text{model}}}. \quad (3.21)$$

onde:

- $\mathbf{E}_{\text{token}} \in \mathbb{R}^{V \times d_{\text{model}}}$ é a matriz de *embeddings* de *tokens*, com V sendo o tamanho do vocabulário;

- $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{L \times d_{\text{model}}}$ é a matriz de embeddings posicionais, com L sendo o comprimento máximo da sequência;
- x é o índice do *token*;
- p é a posição do *token* na sequência.

O GPT-2 empilha N blocos *Transformer* idênticos (12 para GPT-2 small, 48 para GPT-2 large). Cada bloco realiza:

$$\mathbf{a}^{(l)} = \text{LayerNorm}(\mathbf{h}^{(l-1)}) \quad (3.22)$$

$$\mathbf{m}^{(l)} = \mathbf{h}^{(l-1)} + \text{MultiHeadAttention}(\mathbf{a}^{(l)}) \quad (3.23)$$

$$\mathbf{f}^{(l)} = \text{LayerNorm}(\mathbf{m}^{(l)}) \quad (3.24)$$

$$\mathbf{h}^{(l)} = \mathbf{m}^{(l)} + \text{FFN}(\mathbf{f}^{(l)}) \quad (3.25)$$

onde l indica o índice da camada.

Após o último bloco *Transformer*, uma normalização de camada final é aplicada, seguida por uma projeção linear para o espaço do vocabulário:

$$\mathbf{y} = \text{softmax}(\mathbf{h}^{(N)} \mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}) \quad (3.26)$$

onde $\mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times V}$ é a matriz de pesos de saída (frequentemente compartilhada com $\mathbf{E}_{\text{token}}^T$) e $\mathbf{y} \in \mathbb{R}^V$ representa a distribuição de probabilidade sobre o vocabulário.

O GPT-2 é treinado usando o objetivo de modelagem de linguagem causal, que maximiza a probabilidade logarítmica dos *tokens* dados os *tokens* anteriores:

$$\mathcal{L}(\theta) = - \sum_{i=1}^T \log P(x_i | x_{<i}; \theta) \quad (3.27)$$

onde:

- θ representa todos os parâmetros do modelo;
- T é o comprimento da sequência;
- $x_{<i}$ denota todos os *tokens* anteriores a x_i .

Durante o treinamento, utiliza-se tipicamente o algoritmo de otimização Adam (Kingma & Ba 2014) com taxa de aprendizado adaptativa e *gradient clipping* para estabilidade:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad (3.28)$$

onde:

- $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$ é a estimativa corrigida do primeiro momento;
- $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$ é a estimativa corrigida do segundo momento;

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} \mathcal{L}$ é a média móvel exponencial dos gradientes;
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} \mathcal{L})^2$ é a média móvel exponencial dos gradientes ao quadrado;
- η é a taxa de aprendizado;
- β_1, β_2 são hiperparâmetros de decaimento (tipicamente 0,9 e 0,999).

Durante a inferência, o GPT-2 gera texto de forma autorregressiva, selecionando o próximo *token* com base na distribuição de probabilidade condicional:

$$x_{t+1} \sim P(x \mid x_1, x_2, \dots, x_t) \quad (3.29)$$

Diversas estratégias de decodificação podem ser empregadas para a geração do próximo *token*, conforme descrito a seguir.

No *Greedy Decoding*, o *token* com maior probabilidade condicional é selecionado em cada passo:

$$x_{t+1} = \arg \max_x P(x \mid x_{<t+1}) \quad (3.30)$$

No *Top-k Sampling*, a distribuição de probabilidade é restrita aos k *tokens* mais prováveis, e o próximo *token* é amostrado desse subconjunto:

$$x_{t+1} \sim P_k(x \mid x_{<t+1}) \quad (3.31)$$

onde P_k redistribui a probabilidade apenas entre os k *tokens* mais prováveis.

O *Nucleus Sampling*, ou *Top-p Sampling*, considera o menor conjunto de *tokens* cuja probabilidade cumulativa excede um limiar p :

$$x_{t+1} \sim P_p(x \mid x_{<t+1}) \quad (3.32)$$

em que P_p é definida sobre esse conjunto adaptativo de *tokens*.

A técnica de *Temperature Scaling* ajusta a distribuição de probabilidade por meio de um parâmetro de temperatura T :

$$P(x_i \mid x_{<i}) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \quad (3.33)$$

onde z_i representa os *logits* do modelo. Valores de $T < 1$ tornam a distribuição mais concentrada, resultando em saídas mais determinísticas, enquanto valores de $T > 1$ produzem distribuições mais uniformes e gerações mais aleatórias.

O GPT-2 demonstrou capacidades expressivas em diversas tarefas de Processamento de Linguagem Natural sem a necessidade de ajuste fino (*zero-shot* e *few-shot learning*), incluindo tradução automática, sumarização, pergunta-resposta e geração criativa de texto. No entanto, o modelo apresenta limitações relevantes:

- **Alucinações:** tendência a gerar informações factualmente incorretas com elevado grau de confiança;

- **Viés:** reprodução de vieses presentes nos dados de treinamento;
- **Contexto limitado:** janela de contexto finita, restrita a 1024 *tokens* no GPT-2;
- **Raciocínio:** dificuldades em tarefas que exigem raciocínio lógico complexo ou matemática simbólica;
- **Custo computacional:** necessidade de recursos computacionais significativos para treinamento e inferência.

Apesar dessas limitações, o GPT-2 estabeleceu as bases para modelos subsequentes ainda mais avançados, como o GPT-3, GPT-4 e outros modelos de LLMs modernos, consolidando a arquitetura *Transformer* como o paradigma dominante no processamento de linguagem natural.

Capítulo 4

Metodologia

Este capítulo descreve detalhadamente a metodologia desenvolvida para viabilizar o treinamento, a compressão e a implantação de *Small Language Models* (SLMs) em sistemas embarcados com recursos computacionais restritos. A abordagem proposta, denominada *Evolving Vector Quantization-Aware Training* (EVQAT), integra técnicas de quantização vetorial evolutiva diretamente ao ciclo de treinamento de modelos baseados na arquitetura *Transformer*, permitindo comprimir e adaptar os parâmetros do modelo. O fluxo metodológico é organizado de forma sequencial, abrangendo desde a aquisição e preparação dos dados até a geração automática de código otimizado para implantação em microcontroladores. As etapas principais incluem: extração de conhecimento a partir de manuais técnicos, pré-processamento e tokenização dos dados, definição de uma arquitetura de modelo leve, treinamento com integração de técnicas de quantização (evolutiva e INT8), exportação dos parâmetros e geração de código C++ autossuficiente. Esta metodologia foi projetada para atender aos rigorosos requisitos de memória, latência e consumo energético típicos de dispositivos de borda, priorizando a operação local, a privacidade dos dados e a transparência do processo de compressão.

4.1 Visão Geral

A metodologia proposta materializa um framework completo para o treinamento, a quantização e a implantação de SLMs em sistemas embarcados com recursos computacionais restritos. O fluxo de trabalho é estruturado de forma sequencial e modular, abrangendo explicitamente todas as etapas e alternativas representadas na Figura 4.1, cobrindo integralmente o ciclo de vida do modelo, desde a aquisição dos dados brutos até a inferência em tempo de execução no dispositivo de borda.

Conforme ilustrado na Figura 4.1, o processo inicia-se com a aquisição de dados, na qual manuais de usuários de veículos, em formato PDF, são coletados como principal fonte de conhecimento. Em seguida, a etapa de extração de conhecimento constrói pares estruturados de perguntas e respostas. Esses dados são então submetidos ao pré-processamento e à tokenização, que incluem procedimentos de limpeza, normalização e conversão para representações numéricas adequadas ao treinamento do modelo.

Na etapa subsequente, é definida a arquitetura do modelo, baseada em um *Transformer* leve, projetado especificamente para atender às restrições de memória e capacidade

de processamento de microcontroladores. O modelo é então treinado utilizando o conjunto de dados preparado. Conforme explicitado na figura, o *framework* suporta múltiplas estratégias de representação numérica dos parâmetros após o treinamento, incluindo o uso padrão em ponto flutuante de 32 bits (FP32), a quantização pós-treinamento em INT8 e abordagens conscientes de quantização com mecanismos evolutivos, nas quais os *codebooks* são ajustados dinamicamente para preservar a distribuição dos pesos sob compressão.

Independentemente da opção selecionada, todos os parâmetros e configurações do modelo treinado são exportados de forma estruturada em arquivos JSON. Esses arquivos alimentam automaticamente a etapa de conversão do modelo, na qual os parâmetros são traduzidos para código-fonte C++ otimizado. Por fim, o *framework* contempla explicitamente a implantação do modelo no dispositivo embarcado e sua execução em modo de inferência, permitindo o processamento de consultas em tempo real de forma totalmente local, conforme representado no fluxo completo da Figura 4.1.

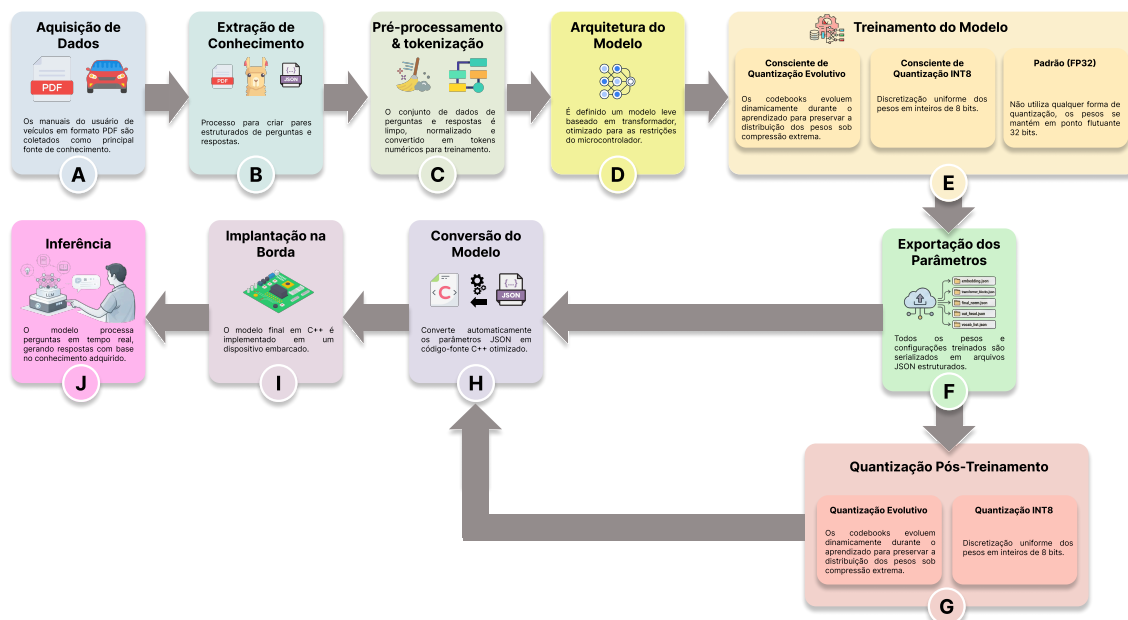


Figura 4.1: Visão geral da metodologia proposta desde o treinamento até a implantação de um SLM em dispositivos de baixa potência.

4.2 Aquisição de Dados

A primeira etapa consiste na extração de conhecimento técnico a partir de documentos textuais, definidos como a fonte primária de dados do sistema. No contexto deste estudo, embora sejam utilizados manuais de veículos, a metodologia proposta não é restrita a um domínio específico de aplicação, podendo ser igualmente empregada na análise de documentos técnicos de outras áreas, tais como indústria, energia, saúde, telecomunicações ou sistemas embarcados em geral.

4.3 Extração de Conhecimento

O processo de geração do conjunto de dados sintético seguiu um *pipeline* metodológico rigoroso, ilustrado na Figura 4.2, composto por quatro etapas principais: (i) extração de conteúdo textual de manuais técnicos em formato PDF, (ii) segmentação semântica em fragmentos de até 100 caracteres, (iii) geração sintética automatizada de pares pergunta–resposta via modelo de linguagem, e (iv) estruturação e armazenamento final em formato JSON. Cada etapa foi cuidadosamente projetada para transformar documentação técnica bruta em um formato estruturado adequado ao treinamento de SLMs para implantação em ambientes de recursos limitados.



Figura 4.2: Fluxo da ingestão do manual técnico (PDF), segmentação semântica controlada, geração de pares instrução–resposta via LLaMA 3.2 e estruturação final em JSON para treinamento do modelo.

4.3.1 Entrada

Inicialmente, o conteúdo textual foi extraído de manuais técnicos automotivos em PDF. Esse formato foi escolhido devido à sua ampla utilização na documentação técnica industrial e à disponibilidade de ferramentas robustas capazes de analisar documentos com estruturas complexas e elementos gráficos estruturados.

4.3.2 Segmentação Semântica

Após a extração, o texto bruto passou por uma etapa de segmentação, que ao contrário de abordagens simplistas baseadas na divisão por comprimento fixo, o algoritmo implementado respeitou os limites naturais dos parágrafos presentes no material de origem. Parágrafos consecutivos foram agrupados de maneira iterativa até que o limite de 100 caracteres fosse atingido, momento em que um novo fragmento (*chunk*) era iniciado. Essa estratégia de segmentação semântica assegura que cada fragmento preserve uma unidade informacional autoconclusiva, como uma descrição procedimental, uma definição técnica ou uma especificação.

A limitação rígida de 100 caracteres por fragmento constitui uma decisão de projeto crítica para a viabilidade da solução em microcontroladores de recursos ultralimitados. Matematicamente, esta restrição no comprimento da sequência de entrada (N) impacta diretamente múltiplas dimensões da eficiência do modelo:

1. **Redução de memória:** O tamanho da matriz de *embeddings* e das ativações intermediárias armazenadas na memória SRAM escala linearmente com o comprimento da sequência de entrada N ;
2. **Complexidade computacional:** Em arquiteturas baseadas em atenção, a limitação mitiga a complexidade quadrática $O(N^2)$ das operações matriciais de *self-attention*;
3. **Latência de inferência:** O número de operações de ponto flutuante (FLOPs) por predição permanece dentro dos limites operacionais do hardware alvo, viabilizando inferência em tempo real.

Tal otimização é indispensável para garantir que o pico de uso de memória e a carga computacional permaneçam compatíveis com as restrições de SRAM (tipicamente 256 a 512 KB) e frequência de *clock* (80–240 MHz) dos microcontroladores considerados neste trabalho.

4.3.3 Geração Sintética de Perguntas e Respostas

Para cada fragmento de texto segmentado, um par pergunta-resposta foi gerado automaticamente por meio de um sistema de *Retrieval-Augmented Generation* (RAG), construído sobre o modelo LLaMA 3.2 (Grattafiori et al. 2024). O *framework* RAG foi empregado exclusivamente na fase *offline* de geração do conjunto de dados; o modelo assistente final opera inteiramente de forma autônoma, sem qualquer componente de recuperação em tempo de inferência.

A Engenharia de *Prompts* foi elaborada para garantir perguntas concisas, objetivas e tecnicamente precisas, derivadas estritamente do conteúdo de cada fragmento. O *prompt* utilizado para geração dos pares instrução-resposta segue a estrutura apresentada no Código 4.1.

Listagem 4.1: *Prompt* utilizado para geração automatizada de pares pergunta-resposta a partir de fragmentos de 100 caracteres do manual técnico.

```
prompt = f"""
You are a data generator for fine-tuning or training a language
model.

Given the content below, create:
1. A short and objective question (instruction), under 10 words,
   based only on the content.
2. A direct and concise answer (response), under 10 words,
   based only on the content.

Do not add extra explanations. Use the following format:

INSTRUCTION: <short question>
RESPONSE: <short answer>
```

```
Content :
\" \" \"
{ chunk }
\" \" \"
\" \" \"
```

Esse *prompt* direcionado resultou em pares pergunta-resposta caracterizados por alta densidade informacional e rigoroso alinhamento semântico com o conteúdo de origem. Após a geração automatizada, realizou-se uma curadoria manual nos 130 pares produzidos, com revisão detalhada para validar a consistência técnica das informações, corrigir possíveis imprecisões factuais e garantir fidelidade absoluta ao manual técnico original. Essa etapa de validação humana, essencial para aplicações críticas em sistemas embarcados automotivos, assegurou que todas as respostas refletissem com precisão o conhecimento dominal presente nos fragmentos processados.

O processo de geração e curadoria produziu um conjunto final de 130 pares pergunta-resposta, quantidade deliberadamente limitada para manter um escopo controlado nos experimentos iniciais de avaliação de desempenho. Essa dimensão reduzida do conjunto de dados permite analisar de forma sistemática o desempenho de diferentes arquiteturas de modelos em múltiplas famílias de microcontroladores, isolando os efeitos da escala dos dados de outros fatores experimentais.

4.3.4 Segmentação e Armazenamento dos dados

Os pares pergunta-resposta resultantes foram estruturados no formato JSON, onde cada linha representa um objeto JSON independente contendo os campos pergunta e resposta. Esta escolha de formato facilita o processamento incremental durante o treinamento, reduz o consumo de memória durante o carregamento de dados e assegura a reprodutibilidade dos experimentos.

Os dados gerados foram particionados em dois subconjuntos disjuntos, sendo 100 pares destinados ao treinamento (77%) e 30 pares reservados para teste (23%). Essa divisão possibilita a avaliação da capacidade de generalização do modelo a partir de exemplos não observados durante o processo de treinamento. Exemplos representativos desses pares são apresentados na Tabela 4.1, evidenciando a variedade de informações capturadas, incluindo simbologia de segurança, componentes estruturais, localização de sensores e funcionalidades do veículo.

Tabela 4.1: Exemplos representativos de pares de pergunta-resposta gerados a partir do manual técnico automotivo.

Pergunta	Resposta
What does the symbol DANGER represent?	Warning or threat indicator
What is under the Volkswagen badge?	Radar sensor for assist systems
What are the first two parts of a car?	Engine compartment and windscreen
Where is the rain and light sensor located?	Near the interior mirror
What features are included in this vehicle?	Sensors for assist and jacking points

4.4 Pré-processamento e Tokenização

O conjunto de dados é submetido a etapas sistemáticas de limpeza textual, normalização e tokenização, com o objetivo de transformar os pares pergunta–resposta em sequências discretas de *tokens*. Esse pré-processamento reduz variações ortográficas e sintáticas irrelevantes, além de garantir uma representação consistente do conteúdo textual para o treinamento do modelo.

Para estruturar adequadamente as sequências de entrada e saída, são introduzidos símbolos especiais com funções bem definidas, a saber: `<pad>` para preenchimento de comprimento, `<sos>` para indicar o início da sequência, `<eos>` para marcar o término da sequência, `<unk>` para representar termos fora do vocabulário e `<sep>` para separar explicitamente a pergunta da resposta. Esses *tokens* especiais auxiliam o modelo a aprender o alinhamento entre os elementos de entrada e saída durante o treinamento supervisionado, contribuindo para maior estabilidade no processo de otimização e melhor generalização.

4.5 Arquitetura do Modelo

O modelo proposto adota uma adaptação leve da arquitetura *Transformer* baseada em codificador do tipo GPT, otimizada para execução em ambientes de borda. Conforme ilustrado na Figura 4.3, ele consiste em camadas de *embedding*, múltiplos blocos *Transformer* e uma camada de projeção. Esta estrutura mantém os principais benefícios arquiteturais do GPT-2 (Gross et al. 2026), reduzindo simultaneamente a sobrecarga computacional.

A sequência de entrada é tokenizada e projetada em um espaço vetorial denso de dimensão C através de uma matriz de *embedding*, onde T é o comprimento da sequência e V o tamanho do vocabulário. *Embeddings* posicionais são adicionados para codificar informações de ordem. As representações resultantes passam por L blocos *Transformer* empilhados, cada um contendo normalização de camada (LayerNorm), atenção multi-cabeça mascarada, *dropout*, conexões residuais e uma rede *feed-forward* com ativação GeLU. Após o bloco final, uma LayerNorm e uma projeção linear mapeiam as características de volta ao espaço do vocabulário para predição autorregressiva.

Esta arquitetura preserva capacidades essenciais de modelagem de linguagem mantendo-se computacionalmente eficiente, tornando-a adequada para implantação em dispositivos com recursos limitados.

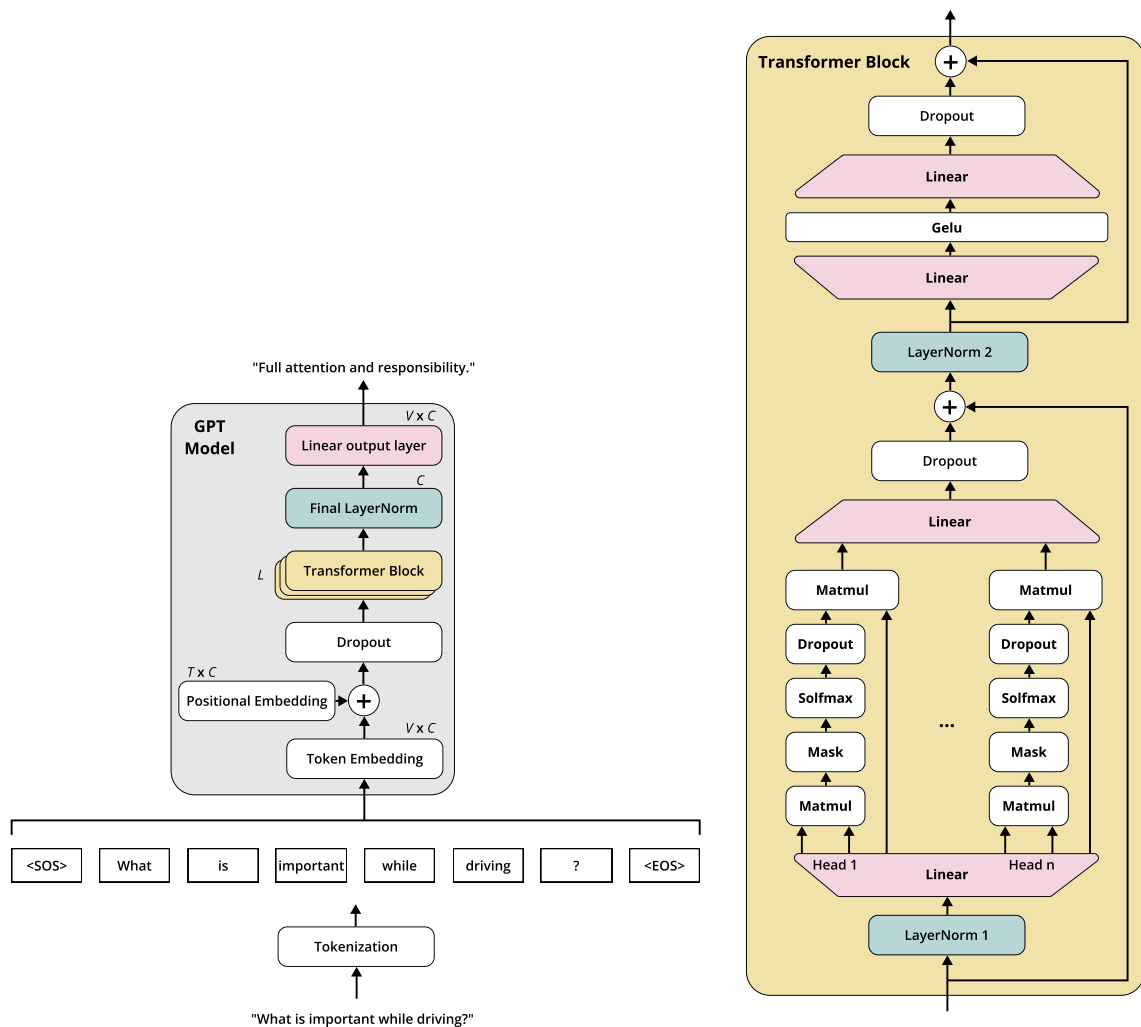


Figura 4.3: Visão geral da arquitetura do modelo baseado em GPT proposto, mostrando o fluxo de processamento da consulta de entrada desde a tokenização até a geração de saída. O painel direito ilustra a estrutura interna de um Bloco *Transformer* com atenção multi-cabeça mascarada, normalização e sub-camadas *feed-forward*.

4.6 Treinamento do Modelo

O processo de treinamento do modelo foi concebido de forma flexível e modular, permitindo a avaliação de diferentes estratégias de quantização integradas ao ciclo de aprendizado. O *framework* de treinamento suporta três modos distintos: (i) treinamento padrão em precisão total (FP32); (ii) treinamento consciente de quantização vetorial evolutiva, com subamostragem dos parâmetros; e (iii) treinamento consciente de quantização INT8. Cada modo estabelece um compromisso específico entre precisão numérica, consumo de memória e eficiência computacional, aspectos críticos para a implantação do modelo em microcontroladores de baixa potência.

Conforme ilustrado na Figura 4.4, o processo tem início com a definição dos *embeddings* de *tokens* e posicionais, bem como dos parâmetros dos blocos *Transformer*, todos inicialmente representados em precisão FP32. Em seguida, o método de treinamento é selecionado, estabelecendo o fluxo adotado ao longo das épocas de treinamento. O procedimento completo de treinamento, incluindo a atualização dos parâmetros e a aplicação das estratégias de quantização durante o aprendizado, é formalizado no Algoritmo 1.

No modo de treinamento padrão, os parâmetros do modelo permanecem integralmente em precisão total, não sendo aplicadas quaisquer técnicas de quantização. Esse modo é utilizado como referência de desempenho e estabelece a linha de base para comparação com os demais métodos avaliados.

No treinamento consciente de quantização vetorial evolutiva, descrito de forma detalhada no Algoritmo 1 e no Algoritmo 2, os parâmetros dos blocos *Transformer*, da camada de saída e das normalizações são periodicamente achatados e submetidos a um processo de subamostragem baseada em quantis. Essa etapa reduz a complexidade do espaço de busca e viabiliza a aplicação de algoritmos de agrupamento para o cálculo de centroides representativos. Em seguida, os pesos originais são substituídos pelos centroides mais próximos, incorporando a quantização diretamente ao ciclo de treinamento.

De forma alternativa, no treinamento consciente de quantização INT8, conforme também especificado no Algoritmo 1, os parâmetros são escalonados por meio de um mapeamento mínimo-máximo (min-max) para uma representação inteira de baixa precisão. Posteriormente, aplica-se o escalonamento inverso, permitindo que os valores quantizados retornem ao domínio de ponto flutuante e viabilizando a continuidade do processo de retropropagação e atualização dos pesos.

Independentemente do modo selecionado, o treinamento segue o fluxo padrão de modelos baseados em *Transformers*, envolvendo a tokenização das entradas, a consulta aos *embeddings*, a propagação direta, o cálculo da função de perda, a retropropagação dos gradientes e a otimização dos parâmetros por mini-lotes. Ao final de cada época, o contador de iterações é incrementado e o processo prossegue até o atendimento dos critérios de parada definidos.

4.6.1 Treinamento Padrão

Na configuração de Treinamento Padrão, os parâmetros do modelo são preservados em sua representação original (FP32), abrangendo camadas de normalização, a cabeça



Figura 4.4: Fluxograma do processo de treinamento da metodologia proposta, detalhando as etapas de inicialização, os três modos de quantização (Padrão, Vetorial Evolutiva e INT8) e o ciclo de otimização com atualização dos parâmetros mestres.

Algoritmo 1: Procedimento de Treinamento para LLM baseado em GPT com Estratégias de Quantização Durante o Treinamento

Input: Dataset de treinamento $\mathcal{D} = \{(\mathbf{X}_b, \mathbf{y}_b)\}$

Output: Modelo GPT treinado

```

1 Inicializar parâmetros do GPT: Embeddings de token  $\mathbf{E}$  e embeddings posicionais  $\mathbf{P}$ ;
2 Parâmetros dos blocos Transformer  $\{\mathbf{W}_{\text{attn}}^{(l)}, \mathbf{W}_{\text{mlp}}^{(l)}, \gamma^{(l)}, \beta^{(l)}\}$ ;
3 Normalização final  $\gamma^{\text{final}}, \beta^{\text{final}}$  e cabeça de saída  $\mathbf{W}_{\text{out}}$ ;
4 for cada época  $t$  do
5   if modo de treinamento = padrão then
6     | Utilizar todos os parâmetros em precisão FP32;
7   end
8   if modo = quantização vetorial evolutiva e  $t$  pertence à fase de quantização then
9     for cada bloco Transformer  $l$  do
10      | Achatar os parâmetros do bloco em um vetor  $\mathbf{w}$ ;
11      | Aplicar subamostragem baseada em quantis:
12      |
13      |  $\hat{\mathbf{w}} \leftarrow \text{Subsample}(\mathbf{w}, \text{fração} = 0.1, \text{n\_bins} = 50)$ 
14      |
15      | Aplicar quantização vetorial conforme Algoritmo 2 para obter centroides;
16      | Substituir cada elemento  $x \in \mathbf{w}$  pelo centroide mais próximo  $c_j$ ;
17      | Remodelar os parâmetros para a estrutura original do bloco  $l$ ;
18    end
19    Aplicar o mesmo procedimento a  $\mathbf{W}_{\text{out}}$  e aos parâmetros de normalização;
20  end
21  if modo = quantização Int8 e  $t$  pertence à fase de quantização then
22    for cada bloco Transformer  $l$  do
23      | Calcular  $\min(\mathbf{W}^{(l)})$  e  $\max(\mathbf{W}^{(l)})$ ;
24      | Quantizar os parâmetros por escalonamento min-max:
25      |
26      |  $q = \text{round}\left(\frac{x - \min}{\max - \min} \cdot 255\right), \quad q \in [0, 255]$ 
27      |
28      | Desquantizar para FP32:
29      |
30      |  $\hat{x} = \frac{q}{255} \cdot (\max - \min) + \min$ 
31      | Substituir os parâmetros originais por  $\hat{x}$ ;
32    end
33    Aplicar o mesmo procedimento à camada de saída e aos parâmetros de
34    normalização;
35  end
36  for cada mini-lote  $(\mathbf{X}_b, \mathbf{y}_b)$  do
37    | Propagação direta: embeddings  $\rightarrow$  blocos Transformer  $\rightarrow$  logits;
38    | Calcular a função de perda  $\mathcal{L}(\mathbf{y}_b, \hat{\mathbf{y}}_b)$ ;
39    | Retropropagação para obtenção dos gradientes  $\nabla \theta$ ;
40    | Atualizar os parâmetros:
41    |
42    |  $\theta \leftarrow \theta - \eta \cdot \nabla \theta$ 
43  end
44  Avaliar opcionalmente a perda de validação  $\mathcal{L}_{\text{val}}$ ;
45 end

```

Algoritmo 2: Procedimento de Quantização Vetorial com Subamostragem**Input:** Vetor de parâmetros $\mathbf{x} = (x_1, \dots, x_N)$ **Output:** Vetor quantizado $\mathbf{x}$

1 Aplicar subamostragem baseada em quantis:

$$\hat{\mathbf{x}} \leftarrow \text{Subsample}(\mathbf{x}, \text{fração} = 0.1, \text{n_bins} = 50)$$

2 Aplicar algoritmo de agrupamento (ex.: Autocloud, Mean-Shift, DBStream, Affinity Propagation) a $\hat{\mathbf{x}}$ para obter o conjunto de centroides C ;3 **for** cada elemento $x_i \in \mathbf{x}$ **do**

4 Determinar o centroide mais próximo:

$$c_j = \arg \min_{c \in C} \|x_i - c\|$$

 Substituir $x_i \leftarrow c_j$;5 **end**6 Remodelar $\mathbf{x}$ para as dimensões originais da camada;

de saída e todos os blocos *Transformer*. Esta configuração adere à metodologia convencional para arquiteturas do tipo GPT, priorizando máxima precisão preditiva sem impor restrições de memória ou computacionais.

Embora esta abordagem garanta alta precisão numérica, ela incorre em demandas substanciais de armazenamento e energia, tornando-a impraticável para implantação em dispositivos com recursos limitados. O Algoritmo 1 descreve este modo na condição em que o modo de treinamento é configurado como “Padrão”.

4.6.2 Treinamento Consciente de Quantização Vetorial Evolutiva

O Treinamento Consciente de Quantização Vetorial Evolutiva introduz a quantização diretamente no processo de otimização, integrando um mecanismo de agrupamento dinâmico dentro do loop de treinamento. Antes do agrupamento, uma etapa de subamostragem é aplicada para reduzir o número de parâmetros preservando sua distribuição estatística. Esta redução seletiva aumenta a eficiência computacional sem distorcer a distribuição geral dos parâmetros. O procedimento completo é apresentado no Algoritmo 1 e detalhado pelo Algoritmo 2, sendo resumido da seguinte forma:

- **Subamostragem de parâmetros:** Antes do agrupamento, um subconjunto representativo de parâmetros é extraído por meio de amostragem estratificada baseada em quantis. Os parâmetros são particionados em um número predefinido de *bins* de acordo com seus quantis, e uma fração (tipicamente 10%) é amostrada aleatoriamente de cada *bin*. Esse procedimento contribui para que a subamostra preserve a distribuição estatística do conjunto completo de parâmetros, ao mesmo tempo em que reduz substancialmente o custo computacional associado à etapa de agrupamento.

- **Agrupamento de parâmetros:** Os parâmetros subamostrados das camadas de normalização, cabeça de saída e blocos *Transformer* são concatenados em um vetor unificado. Um algoritmo de agrupamento como Autocloud, Mean-Shift, DBStream ou Affinity Propagation é então aplicado a este conjunto reduzido para particionar o espaço de parâmetros em grupos (*clusters*). Cada grupo é representado por um centroide, que serve como aproximação quantizada dos valores originais dos parâmetros. Esta etapa reduz significativamente os requisitos de armazenamento enquanto preserva a distribuição global dos pesos, conforme descrito no Algoritmo 2.
- **Substituição baseada em centroide:** Cada parâmetro no modelo é substituído por seu centroide mais próximo. Isso impõe uma representação quantizada em todas as camadas mantendo coerência estrutural e reduzindo redundância dentro do espaço de parâmetros. A proximidade entre parâmetros e centroides pode ser computada usando várias métricas de distância, tais como Euclidiana (L2) (Gower 1985), Manhattan (L1) (Suwanda et al. 2020), Minkowski (Xu et al. 2019), Chebyshev (L ∞) (Gultom et al. 2018), Similaridade de Cosseno (Li 2019), Hamming (Hamming 1950), Dissimilaridade de Bray–Curtis (Bray & Curtis 1957), Jaccard (Chierichetti et al. n.d.), Wasserstein (Vasershtein 1969), Dynamic Time Warping (DTW) (Sakoe & Chiba 1978) e Mahalanobis (Mahalanobis 1936).
- **Propagação direta sob quantização:** O modelo realiza inferência usando parâmetros quantizados. Embora a quantização introduza pequenas perturbações nos valores dos parâmetros, esta abordagem alinha a dinâmica de treinamento com as condições reais de implantação encontradas em hardware embarcado de baixa potência.
- **Propagação reversa e atualização de parâmetros:** Os gradientes são computados em relação aos pesos quantizados, mas as atualizações são aplicadas aos parâmetros de precisão completa. Os centroides são então ajustados iterativamente conforme o treinamento progride, permitindo que evoluam em alinhamento com o objetivo de otimização.

Ao incorporar subamostragem baseada em quantis e adaptação evolutiva de centroides, esta abordagem alcança um equilíbrio efetivo entre precisão, pegada de memória e eficiência computacional, viabilizando assim a implantação de modelos do tipo GPT em dispositivos de borda com recursos limitados.

4.6.3 Treinamento Consciente de Quantização INT8

A estratégia de Treinamento com Quantização INT8 introduz uma representação uniforme de inteiro de 8 bits através de escalonamento mínimo–máximo. Diferentemente do agrupamento baseado em centroide descrito no Algoritmo 2, esta abordagem mapeia parâmetros de ponto flutuante para o intervalo discreto $[0, 255]$, conforme especificado no Algoritmo 1. O procedimento é o seguinte:

- Identificar parâmetros quantizáveis, especificamente pesos de camadas lineares e de normalização (camadas de *embedding* são excluídas para preservar fidelidade representacional);
- Computar os valores mínimo e máximo de cada conjunto de parâmetros para definir o intervalo de quantização;
- Normalizar e escalonar linearmente os valores dos parâmetros para o domínio de inteiro de 8 bits;
- Desquantizar valores de volta para FP32 para treinamento diferenciável, garantindo compatibilidade com otimização baseada em gradiente.

Durante o treinamento, o intervalo dinâmico é continuamente atualizado, permitindo que a quantização se adapte à distribuição evolutiva dos parâmetros. Embora a precisão reduzida introduza ruído de quantização, o modelo mantém sua capacidade preditiva aprendendo a compensar erros de discretização. Esta metodologia fornece um compromisso favorável entre precisão, eficiência de memória e velocidade computacional, tornando-a particularmente adequada para implantação em microcontroladores e outras plataformas com restrições de *hardware*.

4.7 Incremento e Condição de Término

Ao final de cada época, o índice de treinamento é incrementado em um, seguido pela avaliação da condição de término. O treinamento é interrompido quando o número máximo predefinido de épocas é alcançado. Se a condição não for atendida, o processo continua, alternando entre fases quantizadas e não quantizadas dependendo da configuração estabelecida no Algoritmo 1.

A integração de quantização periódica no processo de treinamento permite um equilíbrio flexível entre precisão e eficiência. Ao restringir dinamicamente a precisão dos parâmetros, a metodologia proposta favorece que o modelo resultante seja computacionalmente tratável e implantável em dispositivos de borda, onde eficiência de memória e energia são fundamentais.

4.8 Quantização Pós-Treinamento

Além da quantização durante o treinamento descrita no Algoritmo 1, o *framework* proposto suporta Quantização Pós-Treinamento (PTQ), uma técnica na qual a discretização de parâmetros é aplicada após o modelo ter sido completamente treinado em precisão de ponto flutuante. Diferentemente da quantização em tempo de treinamento, a PTQ não altera a dinâmica de otimização; ao invés disso, transforma os pesos e *embeddings* aprendidos em uma representação de menor precisão durante uma fase de pós-processamento.

Esta abordagem é computacionalmente mais simples e evita a necessidade de rotinas especializadas de treinamento com quantização, tornando-a atrativa quando implantação

rápida do modelo é necessária. Dentro do *framework* proposto, duas metodologias de PTQ são implementadas: (i) Quantização Vetorial Evolutiva e (ii) Quantização INT8. Ambos os métodos podem ser aplicados seletivamente aos parâmetros do modelo treinado, aos *embeddings*, ou a ambos os componentes simultaneamente, resultando em três configurações: "*Embedding*", "Modelo" e "Completo".

4.8.1 Quantização Pós-Treinamento Vetorial Evolutiva

Na configuração Quantização Pós-Treinamento Vetorial Evolutiva (EVQ-PTQ), as técnicas de agrupamento são aplicadas aos parâmetros do modelo após convergência do treinamento. Especificamente, os pesos e vieses de ponto flutuante dos blocos *Transformer*, camadas de normalização e cabeça de saída são achatados em um vetor de parâmetros unificado. Um algoritmo de agrupamento, como Autocloud, Mean-Shift, DBStream ou Affinity Propagation, é então aplicado para particionar o espaço de parâmetros em grupos, cada um representado por um centroide, de forma similar ao processo descrito no Algoritmo 2. Os valores originais são substituídos pelo centroide mais próximo, produzindo assim uma representação quantizada dos parâmetros treinados.

Três configurações do EVQ-PTQ são suportadas:

- ***Embedding***: apenas as matrizes de *embedding* são quantizadas, enquanto os parâmetros treinados do modelo permanecem em precisão completa (FP32);
- **Modelo**: apenas os parâmetros treinados do modelo são quantizados usando agrupamento evolutivo, enquanto os *embeddings* são preservados em precisão completa (FP32);
- **Completo**: tanto os parâmetros do modelo treinado quanto os *embeddings* são quantizados simultaneamente, proporcionando o mais alto grau de compressão.

Este procedimento pós-treinamento permite que o modelo se beneficie de requisitos reduzidos de memória sem retreinamento, ao custo de potencial degradação de precisão dependendo da configuração escolhida. A variante `evo_model` é particularmente efetiva quando os *embeddings* devem preservar fidelidade semântica, enquanto `evo_full` maximiza a compressão para implantações em dispositivos com recursos limitados.

4.8.2 Quantização Pós-Treinamento INT8

O método de Quantização Pós-Treinamento INT8 transforma os parâmetros em ponto flutuante em uma representação inteira de 8 bits por meio de uma abordagem de escalonamento mínimo-máximo, similar ao processo descrito no Algoritmo 1 para quantização durante o treinamento. Após a extração dos valores máximo e mínimo em todo o conjunto de parâmetros alvo, os valores são linearmente escalados para o intervalo discreto $[0, 255]$ e subsequentemente desquantizados para FP32. Esse procedimento contribui para que a representação quantizada permaneça compatível com *pipelines* de inferência padrão, ao mesmo tempo em que reduz significativamente o tamanho de armazenamento.

Similarmente à estratégia de agrupamento evolutivo, três configurações PTQ estão disponíveis:

- **Modelo:** apenas os parâmetros treinados do modelo (por exemplo, blocos *Transformer*, camadas de normalização e cabeça de saída) são quantizados para inteiros de 8 bits, enquanto os *embeddings* permanecem em precisão de ponto flutuante;
- **Embedding:** a quantização é aplicada exclusivamente às matrizes de *embedding*, preservando os parâmetros do modelo em precisão completa;
- **Completo:** tanto os parâmetros treinados quanto os *embeddings* são quantizados simultaneamente, produzindo a representação mais compacta.

Esta abordagem fornece uma alternativa leve e compatível com *hardware* ao agrupamento evolutivo. Embora possa introduzir maior ruído de quantização em comparação, a simplicidade do método e sua compatibilidade nativa com aceleradores de inferência modernos o tornam particularmente adequado para implantações no mundo real em dispositivos embarcados.

4.9 Exportação dos Parâmetros do Modelo

Após a conclusão da fase de treinamento seguindo os procedimentos descritos no Algoritmo 1, o conjunto completo de parâmetros do modelo foi sistematicamente serializado e exportado em múltiplos arquivos JSON estruturados:

- **config.json:** Define os hiperparâmetros centrais e configuração arquitetural, incluindo o número de cabeças de atenção, camadas *Transformer*, dimensionalidade do *embedding* e tamanho do vocabulário.
- **vocab_list.json:** Enumera o vocabulário completo como uma lista sequencial de *tokens* em formato *string* que constituem o domínio linguístico do modelo.
- **embedding.json:** Contém os tensores de peso para as camadas de *embedding* de *token* e posicional.
- **transformer_blocks.json:** Encapsula o conjunto completo de parâmetros para cada camada *Transformer*, incluindo todos os pesos e vieses associados aos mecanismos de atenção multi-cabeça e componentes da rede *feed-forward*.
- **final_norm.json:** Armazena os parâmetros de escalonamento e deslocamento para a operação de normalização de camada aplicada após o bloco *Transformer* final.
- **out_head.json:** Contém a matriz de peso da camada de projeção linear final que gera previsões de saída.

Além da geração de código C++ otimizado descrita na Seção 4.10, o *framework* desenvolvido foi concebido com flexibilidade para permitir a recriação e validação do modelo em ambientes de alto nível. Para isso, disponibiliza a funcionalidade de carregar os arquivos JSON gerados durante o processo de serialização para reconstruir integralmente a arquitetura do SLM e realizar inferência diretamente em Python. Esta capacidade é suportada para todos os casos de quantização contemplados pela metodologia, sejam eles aplicados durante o treinamento (Evolutivo ou INT8, conforme Algoritmo 1) ou como uma etapa de pós-treinamento. Tal funcionalidade é crucial para a validação cruzada dos resultados, depuração, análise de precisão e prototipagem rápida, garantindo a consistência do comportamento do modelo entre o ambiente de desenvolvimento e o dispositivo embarcado final.

O fluxo de trabalho de exportação suporta explicitamente duas estratégias distintas para gerenciamento de precisão de parâmetros:

- **Precisão completa de ponto flutuante:** Como método de exportação padrão, esta abordagem serializa todos os tensores como valores de ponto flutuante de 32 bits. Esta preservação de precisão numérica completa é indispensável para manter máxima precisão do modelo e garantir execução idêntica bit a bit, o que é particularmente crucial para procedimentos de verificação e aplicações críticas de segurança.
- **Exportação quantizada:** A infraestrutura de código em linguagem Python implementa técnicas de quantização que reduzem a largura de bits dos parâmetros para minimizar a pegada de memória e acelerar a inferência em dispositivos com recursos limitados:
 - **Quantização INT8:** Esta abordagem implementa quantização de inteiro de 8 bits mapeando os valores originais do tensor de ponto flutuante para um intervalo de inteiro de 8 bits, conforme o processo descrito no Algoritmo 1. O processo preserva tanto os valores quantizados quanto os fatores de escalonamento necessários e metadados de ponto zero requeridos para desquantização precisa durante a inferência, tornando-a particularmente adequada para MCUs com limitações de memória.
 - **Quantização Evolutiva:** O *framework* também suporta técnicas de quantização mais sofisticadas, incluindo algoritmos de agrupamento de pesos baseados em metodologias evolutivas similares ao Algoritmo 2. Nestas implementações, um *codebook* de valores centroide é gerado e armazenado para referência, enquanto os pesos do modelo são convertidos em índices que apontam para os vetores centroide mais próximos, alcançando assim compressão significativa do espaço de parâmetros.

A coleção resultante de arquivos JSON constitui uma arquitetura autocontida e definitiva do modelo treinado, prontamente adaptável às restrições específicas de tamanho e desempenho de sistemas embarcados alvo, mantendo a integridade original do modelo de SLM.

4.10 Geração de Código

No contexto de TinyML, a geração automática de código C++ emerge como uma estratégia vantajosa em relação às metodologias tradicionais baseadas em interpretadores, como o TensorFlow Lite Micro, e à geração manual de código, uma vez que possibilita a produção de binários altamente otimizados, com baixo consumo de memória e sem a sobrecarga de *runtime* associada a metadados de modelo ou ao despacho dinâmico de operadores (Banbury et al. 2021). Embora interpretadores facilitem o compartilhamento de código entre diferentes modelos, eles impõem restrições significativas em microcontroladores devido ao consumo adicional de RAM para estruturas de controle e resolução de operadores em tempo de execução, frequentemente ultrapassando os limites de 256 kB disponíveis em dispositivos como ESP32 ou STM32 (Warden & Situnayake 2022). A geração manual de código, por sua vez, apesar de permitir otimizações *ad hoc*, é suscetível a erros humanos, apresenta elevado custo de desenvolvimento e mostra-se pouco escalável para arquiteturas complexas, como *Transformers*, conforme apontado por revisões recentes que ressaltam a necessidade de automação para reduzir o tempo de implantação e ampliar a portabilidade (Banbury et al. 2021). Estudos recentes indicam que a geração de código estático explora o conhecimento específico do modelo para realizar *unroll* de *loops*, vetorização SIMD e alocação em memória Flash via PROGMEM, alcançando reduções de até 50% no tamanho do binário e acelerações de inferência quando comparadas a abordagens baseadas em interpretadores, sem dependências externas (Urbann et al. 2020). Essa abordagem favorece a auditabilidade do código gerado, a aderência a normas como a ISO 26262 no contexto automotivo e a compatibilidade com compiladores ANSI C em plataformas AVR e ARM, consolidando-se como um elemento relevante no ciclo de vida de modelos de aprendizado de máquina implantados na borda (Chen et al. 2021).

O estágio de geração de código foi desenvolvido para traduzir automaticamente os parâmetros do SLM em código-fonte C++ otimizado, adequado para execução em microcontroladores. Este estágio estabelece uma conexão direta entre o ambiente de treinamento de alto nível e a implantação embarcada de baixo nível.

O processo começa com um conjunto de arquivos JSON exportados da fase de treinamento, contendo as informações do modelo: pesos do arquitetura *Transformer*, parâmetros de normalização, metadados de configuração e a lista de vocabulário. Estes arquivos são primeiramente carregados e validados para garantir consistência interna e completude. Após validação, o vocabulário é reconstruído, o comprimento máximo de *token* é determinado, e os principais parâmetros arquiteturais como número de camadas, cabeças de atenção, dimensão de incorporação e comprimento de contexto são extraídos. *Tokens* especiais que definem o início, fim e separação de sequências também são identificados e mapeados para seus identificadores numéricos correspondentes, preservando a lógica de geração de texto do modelo.

Um gerador dedicado baseado em Python realiza a conversão destes artefatos JSON exportados em dois arquivos-fonte C++. Dessa forma, os parâmetros do modelo são transformados em *arrays* C++ estáticos representando pesos, vieses e constantes de normalização. Os *arrays* são formatados usando precisão de ponto flutuante de 32 bits ou

representações quantizadas de inteiro de 8 bits, dependendo da configuração selecionada durante o treinamento (conforme Algoritmo 1) ou pós-treinamento. No modo quantizado, os pesos são codificados como valores inteiros compactos referenciando um *codebook*, o que habilita desquantização em tempo real durante a inferência. Este design reduz significativamente o consumo de memória mantendo fidelidade numérica, um aspecto essencial em ambientes embarcados com recursos limitados.

O arquivo de implementação gerado inclui todas as rotinas matemáticas necessárias para a inferência, tais como multiplicação matriz-vetor, normalização de camada, funções de ativação e demais operações computacionais. A arquitetura *Transformer*, que integra mecanismos de atenção multi-cabeça, camadas *feed-forward* e conexões residuais, conforme ilustrado na Figura 4.3, é implementada diretamente em C++. Essa tradução direta contribui para a execução em tempo real e para a manutenção de consistência entre a inferência embarcada e o comportamento observado no modelo de referência implementado em Python.

O arquivo de cabeçalho acompanhante define constantes arquiteturais como tamanho do vocabulário e número de camadas. Para manter compatibilidade entre diferentes arquiteturas de microcontrolador, o código gerado emprega diretivas de compilação condicional e usa a palavra-chave `PROGMEM` para alocar grandes *arrays* de dados na memória Flash ao invés de RAM. Esta abordagem habilita implantação tanto em plataformas baseadas em Alf and Vegard's RISC (AVR) quanto em Advanced RISC Machines (ARM) sem a necessidade de bibliotecas externas.

No estágio final, uma função completa de inferência *forward* é montada para computar *logits* de saída para predição de *token*, complementada por rotinas para geração e decodificação de *tokens*. O gerador adapta automaticamente o código de saída dependendo de se a quantização foi aplicada durante o treinamento (Algoritmo 1) ou pós-treinamento, garantindo execução eficiente tanto em modos de inferência de ponto flutuante quanto inteiro. O resultado é um projeto C++ autocontido, pronto para compilação, que reproduz precisamente as etapas de inferência do SLM em microcontroladores, habilitando implantação eficiente de modelos de linguagem dentro de ambientes de computação de borda.

4.11 Implantação de Borda e Inferência

A estratégia de implantação adota uma abordagem totalmente autocontida e orientada a *hardwares* de baixa potência, permitindo que SLMs sejam executados de forma autônoma em microcontroladores. Toda a arquitetura é encapsulada em um conjunto unificado de arquivos-fonte que contém a lógica de inferência, os pesos pré-treinados e os *buffers* de memória estática. Essa integração define constantes arquiteturais, como o tamanho do vocabulário, a dimensionalidade dos *embeddings*, o número de camadas e o comprimento do contexto, bem como as declarações de tensores e de vetores auxiliares. Para preservar a RAM limitada, essas estruturas de dados são explicitamente alocadas na memória não volátil *Flash*. A implementação fornece todas as rotinas matemáticas necessárias para os mecanismos de atenção, normalização e funções de ativação, permitindo que o modelo seja executado sem dependências externas ou *runtime frameworks*. Esse projeto assegura comportamento em tempo real, computação verificável e operação totalmente

offline, sendo adequado a ambientes de borda com recursos restritos.

Em tempo de execução, o ciclo de inferência inicia-se com a ingestão da entrada do usuário na forma de uma cadeia de caracteres. O texto bruto é tokenizado em identificadores inteiros por meio das funções de vocabulário embarcadas, definidas nos arquivos de cabeçalho. Cada *token* é mapeado para seu índice correspondente no arranjo interno de vocabulário do modelo. Em seguida, a sequência inteira resultante é ajustada para corresponder ao tamanho da janela de contexto previamente definida, por meio de truncamento ou preenchimento (*padding*), garantindo compatibilidade estrita com os requisitos de entrada fixa do modelo.

O processo de inferência prossegue por uma série de estágios computacionais que replicam a arquitetura *Transformer*. Inicialmente, cada identificador de *token* é convertido em um vetor denso pela soma de seu *embedding* semântico com a codificação posicional correspondente. Esse processo gera uma matriz que captura tanto o significado quanto a ordem sequencial da entrada. Essas representações são então processadas por uma pilha de blocos *Transformer*, cada um composto por um mecanismo de atenção multi-cabeça (*multi-head attention*) e uma rede *feed-forward*. Em cada bloco de atenção, as matrizes de *query*, *key* e *value* são computadas, e uma operação de produto interno escalonado (*scaled dot-product*) é aplicada com mascaramento causal, de modo a preservar a propriedade autorregressiva do modelo. As saídas são projetadas linearmente, combinadas com a entrada original por meio de conexões residuais e, em seguida, normalizadas. A rede *feed-forward* aplica transformações lineares separadas por funções de ativação, reproduzindo exatamente a configuração estrutural utilizada durante a fase de treinamento.

Após a conclusão das camadas *Transformer*, a representação oculta resultante passa por uma etapa final de normalização, utilizando parâmetros de escala e deslocamento previamente armazenados. Esse vetor normalizado é então multiplicado pela matriz de projeção de saída, produzindo *logits* sobre todo o vocabulário e representando, de forma efetiva, a distribuição de probabilidade predita para o próximo *token* da sequência.

O processo de geração de texto é realizado de maneira iterativa por meio de um laço autorregressivo. O modelo seleciona o *token* mais provável utilizando uma estratégia de decodificação gulosa (*greedy decoding*), o anexa à sequência existente e repete o ciclo de inferência até que um *token* específico de fim de sequência seja produzido ou que um limite de geração previamente definido seja alcançado. A saída decodificada é reconstruída em formato de texto legível por meio das funções de mapeamento embarcadas, permitindo a exibição ou transmissão direta da resposta gerada. Para tarefas interativas, como pergunta–resposta, uma rotina interna gerencia simultaneamente a predição de *tokens* e a reconstrução da cadeia de caracteres, ao mesmo tempo em que otimiza o uso dos *buffers* de memória.

Na etapa final de integração, o *firmware* compilado é incorporado de forma transparente ao ambiente do microcontrolador. O código principal de controle inicializa os periféricos de *hardware*, carrega os dados do modelo a partir da memória *Flash* e gerencia os canais de comunicação serial para interação com o usuário. A tarefa de inferência é executada integralmente no dispositivo, apoiando-se exclusivamente em parâmetros embarcados e *buffers* alocados estaticamente. Essa metodologia de implantação demonstra a viabilidade de executar modelos de linguagem compactos baseados em *Transformers* em

MCUs, viabilizando processamento de linguagem natural em tempo real, interpretável e totalmente *offline* em cenários de computação de borda.

Capítulo 5

Planejamento dos Experimentos

Após a definição formal da metodologia *Evolving Vector Quantization-Aware Training* (EVQAT) e da descrição dos algoritmos de compressão e geração de código no capítulo anterior, torna-se imperativo estabelecer um protocolo experimental rigoroso para validar as hipóteses levantadas. A transição de modelos teóricos de Aprendizado de Máquina para implantações físicas em dispositivos de recursos restritos exige não apenas a verificação da convergência algorítmica, mas também uma análise sistêmica que considere as restrições de *hardware* e a integridade semântica do modelo.

Este capítulo detalha o delineamento dos experimentos, projetados para isolar variáveis críticas e fornecer uma comparação entre a abordagem proposta e o estado da arte. A estrutura experimental foi concebida para cobrir todo o ciclo de vida do modelo: desde a busca por hiperparâmetros ótimos e o treinamento sob restrições de quantização, até a avaliação de desempenho em inferência e a validação física em microcontroladores comerciais.

Nas seções subsequentes, são apresentadas as questões de pesquisa refinadas, os cenários de avaliação comparativa, a configuração detalhada dos hiperparâmetros, as métricas de sucesso (que abrangem eficácia linguística e eficiência energética) e a especificação do ambiente computacional utilizado. Este planejamento visa garantir a reprodutibilidade dos resultados e fundamentar as evidências que serão discutidas no Capítulo de Resultados.

5.1 Questões de Pesquisa

O fluxo metodológico apresentado no Capítulo 4 fornece a fundamentação teórica e a estrutura operacional para responder às questões de pesquisa (QP):

- **QP1 (Escalabilidade e Viabilidade Técnica):** A metodologia EVQAT, previamente validada em arquiteturas simples (MLPs), demonstra escalabilidade ao ser aplicada à compressão de modelos de linguagem baseados em *Transformers* (GPT-2)? Os algoritmos evolutivos conseguem convergir e gerar *codebooks* estáveis durante o treinamento de uma arquitetura complexa com mecanismos de atenção?
- **QP2 (Compromisso Desempenho-Compressão):** Em que medida a compressão via EVQAT afeta a qualidade semântica (medida por Perplexidade e BLEU) em

comparação com a quantização escalar uniforme (INT8) e o modelo de referência (FP32)? Existe um ponto de equilíbrio onde a redução de memória não inviabiliza a utilidade do modelo para tarefas de recuperação de informação técnica?

- **QP3 (Generalização e Adaptação):** Como a adaptação dinâmica dos *codebooks* durante o treinamento (EVQAT) influencia a capacidade de generalização do modelo em comparação com a quantização estática pós-treinamento (PTQ)? A metodologia proposta é capaz de mitigar a degradação de desempenho em dados não vistos (conjunto de teste), comum em abordagens de quantização estática agressiva?
- **QP4 (Viabilidade de Implantação na Borda):** O código C++ gerado automaticamente permite a execução do modelo *Transformer* dentro dos limites físicos de microcontroladores de baixo custo (e.g., ESP32, Raspberry Pi Pico, STM32), atendendo a requisitos de latência (< 1s para interatividade) e uso de memória RAM/Flash, sem depender de sistemas operacionais ou interpretadores externos?
- **QP5 (Auditabilidade e Segurança Funcional):** O *framework* proposto oferece rastreabilidade e determinismo superiores às soluções de caixa-preta de mercado? A transparência do processo de quantização e a geração de código fonte aberto fornecem as evidências necessárias para suportar processos de certificação em sistemas críticos (conforme ISO 26262 e ASPICE), incluindo a previsibilidade do tempo de execução (WCET)?

Dessa forma, a proposta desta tese busca não apenas mensurar a taxa de compressão, mas investigar a natureza do aprendizado sob restrições severas e a adequação da solução para o ciclo de vida de desenvolvimento de software automotivo seguro.

5.2 Cenários de Avaliação

Para endereçar sistematicamente às questões de pesquisa estabelecidas (QP1 a QP5) e isolar as variáveis de impacto no desempenho dos SLMs, o planejamento experimental foi estruturado em cinco cenários distintos. Esta organização permite confrontar a metodologia proposta (EVQAT) contra *baselines* de mercado (INT8) e avaliar o impacto do momento da quantização (durante ou após o treinamento).

Os cenários foram desenhados para cobrir o espectro entre a máxima precisão numérica e a máxima eficiência de *hardware*, conforme descrito a seguir:

- **Cenário de Referência - Precisão completa (FP32):** Consiste no treinamento convencional do modelo *Transformers* (GPT-2) em precisão completa de ponto flutuante (32 bits), sem aplicação de compressão. Este cenário define o limite superior de desempenho semântico (perplexidade e BLEU) e o limite inferior de eficiência (maior consumo de memória), servindo como referência absoluta para responder à QP2.

- **Cenário 1 - Treinamento Consciente de Quantização Vetorial Evolutiva (EVQAT):** Representa a contribuição central da tese e investiga a **QP1**. Neste cenário, algoritmos de agrupamento evolutivo (AutoCloud, Mean-Shift, Affinity Propagation e DBStream) são integrados ao ciclo de *backpropagation*. O objetivo é validar se a adaptação dinâmica dos *codebooks* preserva a capacidade de representação do modelo sob compressão severa, mitigando a perda de generalização.
- **Cenário 2 - Treinamento Consciente de Quantização INT8 (QAT-INT8):** Utiliza a quantização uniforme escalar simulada durante o treinamento. Este cenário atua como o principal competidor do EVQAT, permitindo verificar se a complexidade adicional dos algoritmos evolutivos se traduz em ganho de qualidade semântica frente a métodos padronizados da indústria, endereçando a análise de compromisso da **QP2**.
- **Cenário 3 - Quantização Pós-Treinamento Evolutiva (EVQ-PTQ):** Aplica os mesmos algoritmos de agrupamento do Cenário 1, porém de forma estática, sobre os parâmetros congelados do modelo de referência. A comparação direta entre o Cenário 1 (dinâmico) e o Cenário 3 (estático) é fundamental para responder à **QP3**, isolando o ganho de generalização obtido pela evolução dos *codebooks* em contrapartida à degradação típica da quantização pós-treinamento.
- **Cenário 4: Quantização Pós-Treinamento INT8 (INT8-PTQ):** Aplica quantização escalar uniforme direta (escalamento *min-max*) ao modelo treinado. Este cenário representa a abordagem de menor custo computacional de desenvolvimento e maior determinismo, servindo como base para as análises de viabilidade de implantação (**QP4**) e eficiência ambiental (**QP5**).

A matriz de avaliação cruzada entre estes cenários fundamenta as respostas para as hipóteses levantadas:

- **Análise de Generalização (EVQAT vs. PTQ):** Ao contrastar os resultados de perda nos conjuntos de treino e teste entre os cenários de treinamento consciente (1 e 2) e pós-treinamento (3 e 4), avalia-se a capacidade do método proposto de mitigar o *overfitting* induzido pela discretização.
- **Análise de Eficiência e Sustentabilidade:** A comparação do custo energético de treinamento (elevado no Cenário 1, reduzido no Cenário 4) contra a eficiência de inferência na borda permite traçar o ciclo de vida completo do modelo, respondendo à nova questão de eficiência global.
- **Viabilidade de Borda e Auditabilidade:** A geração de código C++ para todos os cenários permite validar fisicamente o desempenho em microcontroladores (latência e uso de RAM/Flash) e verificar a transparência do código gerado frente a requisitos de certificação (**QP5**).

5.3 Hiperparâmetros do Modelo

Para garantir a reprodutibilidade dos experimentos e a validade estatística das respostas às questões de pesquisa (QP1 a QP5), os hiperparâmetros de arquitetura e treinamento foram rigorosamente controlados. Esta seção detalha as configurações para cada um dos cinco cenários de avaliação, partindo da busca por uma arquitetura ótima no cenário de referência e fixando-a para os cenários subsequentes de compressão.

5.3.1 Cenário de Referência (FP32)

O Cenário de Referência estabelece o limite superior de desempenho semântico. Para determinar a arquitetura *Transformer* (GPT-2) mais eficiente para o corpus automotivo, foi realizada uma busca em grade (*grid search*) explorando variações no número de cabeças de atenção, camadas e dimensão dos embeddings.

A Tabela 5.1 resume o espaço de busca. Os hiperparâmetros de otimização foram mantidos constantes: otimizador AdamW ($\eta = 0.01$), função de perda *Cross Entropy*, *batch size* de 32 e *dropout* de 0.01 para regularização, utilizando a ativação GeLU.

Tabela 5.1: Espaço de busca de hiperparâmetros para definição da arquitetura base (Cenário de Referência).

Parâmetro	Valores Explorados
<i>Heads</i>	[4, 3, 2, 1]
<i>Layers</i>	[4, 3, 2, 1]
<i>Embedding Dimension</i> (d_{model})	[24, 20, 16, 12]
Função de ativação	GeLU
Otimizador	AdamW
Função Perda	Cross Entropy
Batch size	32
Learning rate	0.01
Drop rate	0.01

Após a execução deste cenário, a configuração que apresentou o melhor compromisso entre perda de validação e estabilidade foi: 4 cabeças de atenção, 1 camada *Transformer* e dimensão de *embedding* de 24. Esta arquitetura foi fixada como o modelo base para todos os cenários de quantização subsequentes, garantindo que as variações de desempenho observadas sejam decorrentes exclusivamente dos métodos de compressão.

5.3.2 Cenário 1: Treinamento Consciente de Quantização Vetorial Evolutiva

Este cenário investiga a viabilidade técnica (QP1) e o desempenho (QP2) da proposta central da tese. A arquitetura ($d_{embedding} = 24$, $n_{head} = 4$ e $n_{layer} = 1$) é submetida ao

processo EVQAT, onde os parâmetros de quantização são críticos. A Tabela 5.2 apresenta os algoritmos de agrupamento e métricas de distância avaliados para a formação dos *codebooks*.

Uma etapa de subamostragem (*sub sample*) de 10% baseada em quantis foi aplicada para viabilizar o custo computacional do agrupamento iterativo durante o treinamento.

Tabela 5.2: Hiperparâmetros do processo de quantização vetorial evolutiva (Cenário 1).

Parâmetro	Configuração Selecionada
Métodos de Agrupamento	[Affinity Propagation, Mean-Shift, AutoCloud, DBStream]
Métricas de Distância	[Euclidean, Manhattan, Minkowski, Chebyshev, Cosine Similarity, Hamming, Bray-Curtis, Jaccard, Wasserstein, Mahalanobis]
Otimizador Base	AdamW ($\eta = 0.01$)
Taxa de Subamostragem	0.1 (10%)

A Tabela 5.3 detalha os hiperparâmetros intrínsecos de cada algoritmo de agrupamento. A seleção destes valores buscou equilibrar a granularidade dos grupos (impactando a precisão) e a velocidade de convergência (impactando o tempo de treino).

Tabela 5.3: Hiperparâmetros específicos dos algoritmos de agrupamento (EVQAT e EVQ-PTQ).

Algoritmo	Parâmetro	Valores Explorados
AutoCloud	Threshold (m)	[1,5, 1,75, 2, 2,25]
Mean-Shift	Bandwidth (b)	[0,001, 0,003, 0,005, 0,01]
	Maximum iterations	300
	Bin seeding	True
Affinity Propagation	Damping (d)	[0,5, 0,7, 0,85, 0,95]
	Maximum iterations	500
	Convergence iterations	20
DBStream	Clustering threshold (τ)	[0,02, 0,05, 0,1, 0,2]
	Fading factor (λ)	0,05
	Cleanup interval	4
	Intersection factor	0,5
	Minimum weight	1

5.3.3 Cenário 2: Treinamento Consciente de Quantização INT8

Para responder à QP2 sobre as relações de compromisso frente ao estado da arte, o Cenário 2 aplica a quantização escalar uniforme simulada. A arquitetura permanece fixa

($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$ e $n_{\text{layer}} = 1$) e os hiperparâmetros de treinamento são idênticos aos do Cenário 1.

A única variável distinta é o método de quantização: utiliza-se uma transformação linear *min-max* para o intervalo $[0, 255]$ aplicada aos pesos em cada passo de *forward*, com cálculo de gradientes via *Straight-Through Estimator* (STE) durante o *backward*. Não há hiperparâmetros adicionais de agrupamento neste cenário.

5.3.4 Cenário 3: Quantização Pós-Treinamento Evolutiva

O Cenário 3 avalia a eficácia da quantização vetorial quando dissociada do treinamento (QP3). O ponto de partida é o modelo FP32 final ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$ e $n_{\text{layer}} = 1$) obtido no Cenário de Referência.

Os algoritmos de agrupamento e seus respectivos hiperparâmetros são **idênticos** aos listados na Tabela 5.3. A diferença metodológica é que o agrupamento é aplicado uma única vez sobre os pesos congelados. As avaliações foram segmentadas em três subconfigurações para isolar a sensibilidade das camadas:

- **"Modelo"**: Quantização aplicada apenas às camadas lineares e de atenção;
- **"Embedding"**: Quantização aplicada apenas à matriz de *embeddings*;
- **"Completo"**: Quantização completa do modelo.

5.3.5 Cenário 4: Quantização Pós-Treinamento INT8 (INT8-PTQ)

Finalmente, o Cenário 4 serve como *baseline* de eficiência para implantação na borda (QP4 e QP5). Assim como no Cenário 3, utiliza-se o modelo $d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$, $n_{\text{layer}} = 1$ pré-treinado. A técnica aplicada é o escalonamento estático *min-max* para 8 bits, sem necessidade de calibração iterativa. As mesmas subconfigurações ("Modelo", "Embedding", "Completo") foram adotadas para manter a paridade na comparação.

5.4 Protocolo e Métricas de Avaliação

A fim de assegurar uma avaliação holística do SLM proposto, adotamos um protocolo de avaliação em duas etapas. Esta abordagem captura tanto as capacidades linguísticas teóricas dos modelos quanto sua viabilidade prática quando implantados em hardware com recursos limitados.

Durante as fases de treinamento e validação, o foco reside na minimização do objetivo de otimização, maximizando simultaneamente a qualidade gerativa. As métricas são detalhadas a seguir, separadas conforme a natureza da avaliação.

5.4.1 Métricas de Otimização e Incerteza

- **Perda de Entropia Cruzada (*Cross-Entropy Loss*)**: A função de perda primária atua como o alvo direto de otimização estocástica. Para uma sequência de tokens, a

perda de entropia cruzada quantifica a divergência entre a distribuição de probabilidade empírica e a distribuição modelada pela rede neural (Bengio et al. 2003). É definida formalmente como:

$$\mathcal{L}_{CE} = - \sum_{i=1}^V y_i \log(\hat{y}_i) \quad (5.1)$$

Onde y_i representa o rótulo verdadeiro e $\hat{y}_i$ a probabilidade predita pelo modelo para o i -ésimo token do vocabulário V .

- **Perplexidade (*Perplexity*):** Métrica canônica para avaliação de modelos de linguagem, a Perplexidade (PPL) é derivada da exponenciação da entropia cruzada média. Ela mensura a incerteza do modelo na predição do próximo token, sendo interpretada como o fator de ramificação efetivo (*weighted branching factor*) da linguagem modelada (Jurafsky & Martin 2009):

$$PPL = \exp \left(\frac{1}{N} \sum_{i=1}^N \mathcal{L}_{CE}(x_i) \right) \quad (5.2)$$

5.4.2 Precisão Linguística e Qualidade Semântica

- **Acurácia e Correspondência Exata (*Exact Match*):** Avalia-se a capacidade de recuperação de informação do modelo através da métrica *Exact Match* (EM), amplamente utilizada em tarefas de *Question Answering*. A métrica atribui pontuação máxima apenas se a sequência gerada for idêntica à referência, garantindo rigor sintático e factual (Rajpurkar et al. 2016).

$$EM = \frac{1}{M} \sum_{j=1}^M \mathbb{I}(\hat{S}_j = S_j) \quad (5.3)$$

- **Qualidade Semântica (BLEU):** Para avaliar a fluência e a fidelidade semântica, utiliza-se o *BiLingual Evaluation Understudy* (BLEU). Esta métrica computa a sobreposição de n-gramas entre a saída do modelo e um conjunto de referências humanas, penalizando sentenças excessivamente curtas para evitar vieses de precisão (Papineni et al. 2002).

5.4.3 Eficiência Computacional e Sustentabilidade

A avaliação da eficiência e do impacto ambiental do treinamento de modelos de Inteligência Artificial tornou-se crucial, dado o crescimento exponencial da demanda computacional (Patterson et al. 2021). Para garantir uma análise transparente, monitoram-se três indicadores fundamentais:

- **Tempo Total de Treinamento:** Refere-se à duração total (mensurada em segundos ou horas) necessária para concluir o processo de otimização dos parâmetros do modelo até a sua convergência final.
- **Energia Consumida:** Representa a energia elétrica total (em kWh) consumida pelo *hardware* de processamento (CPU, GPU) durante o período de treinamento. Este valor é obtido preferencialmente através de medidores de energia físicos ou, na sua ausência, por estimativas baseadas no *Thermal Design Power* (TDP) e na taxa de utilização do dispositivo, integrado ao longo do tempo.
- **Emissões de Carbono Equivalente:** Consiste na estimativa da pegada de carbono associada ao consumo energético, calculada utilizando fatores de emissão regionalizados da rede elétrica, seguindo a metodologia proposta por Lacoste et al. (Lacoste et al. 2019):

$$CO_2eq = E \times I_{region} \quad (5.4)$$

Onde E é a energia consumida e I_{region} representa a intensidade de carbono (gCO_2/kWh) específica da localidade onde o *datacenter* ou dispositivo está operando.

5.4.4 Métricas de Inferência em Dispositivos Embarcados

Para validar a viabilidade em dispositivos embarcados, adotam-se as métricas padronizadas pelo *benchmark* MLPerf Tiny (Banbury et al. 2021):

- **Latência de Inferência:** Tempo de relógio (*wall-clock*) para o processamento ponta-a-ponta de uma única amostra;
- **Ocupação de Memória (Flash vs. RAM):** Monitoramento estrito do uso de memória não-volátil (pesos do modelo) e volátil (mapas de ativação), crucial para MCUs com recursos limitados;
- **Consumo de Energia por Inferência:** Integral da potência consumida durante um ciclo de inferência, métrica determinante para a autonomia de dispositivos alimentados por bateria.

5.5 Ambiente Experimental

Todo o processamento de dados, a modelagem e as análises desta pesquisa foram conduzidos em dois ambientes distintos. O primeiro corresponde a uma estação de trabalho, utilizada para o processamento e o treinamento dos modelos. O segundo consiste em plataformas de baixa potência, nas quais os modelos foram embarcados para avaliação. Esta seção apresenta as especificações de hardware, bem como as linguagens de programação, bibliotecas e ferramentas que compõem o ambiente experimental adotado.

5.5.1 Hardware

Os processos de desenvolvimento, de treinamento dos modelos e de geração automática de código para os microcontroladores foram conduzidos em um ambiente computacional dedicado, adequadamente dimensionado para suportar às demandas de processamento, experimentação e instrumentação energética associadas às diferentes etapas metodológicas. As principais características desse ambiente são descritas a seguir:

- **CPU:** Processador Apple M4 Pro, com 12 núcleos de execução (arquitetura ARM64);
- **Memória RAM:** 24 GB;
- **Armazenamento:** Capacidade suficiente para suportar os experimentos conduzidos¹;
- **GPU:** Unidade gráfica integrada ao SoC Apple Silicon, compartilhando memória unificada com a CPU;
- **Sistema operacional:** macOS 15.7.1 (arquitetura ARM64);

Por outro lado, para avaliar a efetividade da abordagem proposta e validar seu desempenho em ambientes embarcados, foram selecionadas quatro unidades de microcontroladores (MCUs), representativas de diferentes arquiteturas e capacidades computacionais. Os dispositivos considerados foram: Arduino Portenta H7 (STM32H747, Arm® Cortex®-M7/M4 em configuração dual-core), ESP-WROOM-32 DEVKit V1 (Xtensa® LX6 dual-core de 32 bits), Raspberry Pi Pico (ARM Cortex®-M0+ dual-core) e Arduino Nano 33 BLE Sense (Arm® Cortex®-M4F equipado com Unidade de Ponto Flutuante – FPU). As especificações técnicas desses microcontroladores são apresentadas na Tabela 5.4.

Tabela 5.4: Microcontroladores considerados nos experimentos.

Dispositivo	Frequência de <i>Clock</i>	RAM	Flash
Arduino Nano 33 BLE Sense	64 MHz	256 KB	1 MB
Raspberry Pi Pico	133 MHz	264 KB	2 MB
Arduino ESP-32	240 MHz	520 KB	4 MB
Arduino Portenta H7	480 MHz	1 MB + 8 MB	16 MB

5.5.2 Software

Todo o fluxo metodológico foi implementado a partir de um ecossistema de softwares de código aberto, bibliotecas especializadas e ferramentas de desenvolvimento, estruturado segundo boas práticas de MLOps e adaptado a cenários de Edge AI e TinyML.

¹Sistema baseado em armazenamento sólido (SSD).

Esse conjunto de recursos deu suporte às diferentes etapas do fluxo experimental, desde a aquisição e o pré-processamento dos dados até o treinamento, monitoramento, avaliação e implantação dos modelos em dispositivos embarcados.

- **Linguagens de Programação:**

- **Python 3.9.6:** linguagem central da metodologia de MLOps/TinyML, empregada nas etapas de aquisição e gerenciamento de dados, pré-processamento, definição das arquiteturas dos modelos, treinamento supervisionado, validação, avaliação de desempenho e automação dos experimentos;
- **C++:** utilizada na fase de implantação dos modelos treinados, após a conversão a partir de representações de alto nível, permitindo a execução eficiente da inferência em microcontroladores com restrições severas de memória, energia e capacidade computacional.

- **Ambiente de Desenvolvimento Embarcado:**

- **Arduino *Integrated Development Environment* (IDE):** plataforma empregada para a compilação, configuração e gravação do código em C++ nos dispositivos embarcados de baixa potência, viabilizando a validação experimental em hardware real.

- **Bibliotecas e Ferramentas de Software:**

- **Aquisição e Gerenciamento de Dados:**

- * **os** e **time:** gerenciamento do sistema de arquivos, controle de execução e temporização dos experimentos;
- * **json** e **csv:** serialização, armazenamento e intercâmbio de dados estruturados ao longo do fluxo de desenvolvimento;
- * **pandas:** manipulação de dados tabulares, organização de *datasets*, análise exploratória e consolidação de resultados;
- * **pdfplumber:** extração programática de conteúdo textual a partir de documentos técnicos em formato PDF.

- **Pré-processamento e Análise Estatística:**

- * **re:** limpeza, normalização e filtragem de texto por meio de expressões regulares;
- * **math** e **statistics:** suporte a cálculos matemáticos e estatísticos fundamentais;
- * **scipy.stats:** execução de testes estatísticos e análises quantitativas complementares.

- **Modelagem, Treinamento e Avaliação:**

- * **PyTorch:** biblioteca central para definição das arquiteturas dos modelos, treinamento, avaliação de desempenho e gerenciamento dos parâmetros aprendidos, constituindo o núcleo do processo do desenvolvimento do modelo de aprendizado profundo.

– **Monitoramento Experimental e Sustentabilidade:**

- * **Weights & Biases (wandb):** rastreamento de experimentos, versionamento de hiperparâmetros, registro de métricas e garantia de reprodutibilidade, em conformidade com práticas de MLOps;
- * **CodeCarbon:** estimativa do consumo energético e da pegada de carbono associada aos processos de treinamento e experimentação.

– **Implantação e Integração com Sistemas Embarcados:**

- * **ollama:** execução e orquestração local de modelos de linguagem durante as etapas de validação e inferência;
- * **serial:** comunicação serial entre o sistema hospedeiro e os microcontroladores, permitindo a troca de dados, execução da inferência embarcada e coleta de resultados em tempo real.

Capítulo 6

Resultados

Este capítulo apresenta e discute os resultados experimentais obtidos a partir do planejamento descrito no Capítulo 5. A análise concentra-se na comparação sistemática entre o modelo de referência em precisão de ponto flutuante de 32 bits (FP32) e diferentes estratégias de quantização, com o objetivo de caracterizar, de forma rigorosa, os compromissos entre desempenho preditivo, eficiência computacional, consumo energético e viabilidade de implantação em sistemas embarcados.

A discussão é conduzida de maneira progressiva. Inicialmente, estabelece-se um cenário de referência, que define o limite superior de desempenho do modelo. Em seguida, são analisados cenários baseados em técnicas de compressão, permitindo quantificar os impactos da discretização sobre métricas clássicas de modelos de linguagem, tais como perplexidade, correspondência exata e pontuação BLEU, bem como sobre métricas de engenharia, incluindo uso de memória, latência de inferência, vazão de processamento e estimativas de consumo energético. Sempre que pertinente, os resultados quantitativos são acompanhados de análise descritiva, com o intuito de aprofundar a compreensão do comportamento dos modelos sob diferentes regimes de quantização.

6.1 Cenário de Referência: Sem Quantização

O Cenário de Referência consistiu no treinamento de 64 configurações arquiteturais distintas, todas em FP32. As variações foram realizadas de forma sistemática na dimensão do *embedding* ($d_{\text{embedding}}$), no número de cabeças de atenção (n_{head}) e no número de camadas (n_{layer}). Para mitigar a variabilidade estocástica inerente ao treinamento de redes neurais, decorrente tanto da inicialização aleatória dos pesos quanto da amostragem de *batches*, foi adotado um protocolo experimental rigoroso. Os resultados quantitativos apresentados na Figura 6.1, correspondem à média aritmética de cinco execuções independentes para cada configuração.

A análise dos dados consolidados indica que a dimensionalidade do espaço vetorial exerce papel central no desempenho da arquitetura proposta. Observa-se a existência de uma fronteira crítica em $d_{\text{embedding}} = 12$, a partir da qual ocorre colapso no processo de aprendizado. Esse comportamento é evidenciado por valores de perplexidade no conjunto de teste superiores a 10^5 , mesmo considerando o limite inferior do intervalo de confiança. Do ponto de vista teórico, tal fenômeno sugere que valores excessivamente reduzidos de

$d_{\text{embedding}}$ restringem o fluxo residual da rede a um espaço de baixa capacidade informacional, incapaz de representar adequadamente a entropia linguística. Como consequência, o modelo não consegue organizar e separar grupos semânticos no espaço vetorial.

Em contraste, o aumento da dimensão do *embedding* para 24 resultou em estabilização consistente do aprendizado. Nesse regime, a arquitetura composta por duas cabeças de atenção e quatro camadas apresentou a menor perplexidade global observada, com $\approx 14,57$. Esse resultado reforça a hipótese de que, em arquiteturas do tipo *Transformer*, a largura da rede exerce impacto mais imediato sobre a redução da função de perda do que a profundidade, especialmente em cenários com conjuntos de dados limitados.

A comparação entre as métricas obtidas nos conjuntos de treino e de teste evidencia a presença de sobreajuste pronunciado. Enquanto a acurácia de predição de *tokens* no conjunto de treino convergiu para valores próximos da saturação, entre 0,99 e 1,00, a acurácia no conjunto de teste permaneceu inferior a 0,22. De forma ainda mais expressiva, a métrica de correspondência exata foi nula para todas as configurações avaliadas. Esses resultados indicam que o modelo operou predominantemente em regime de memorização, sem desenvolver capacidade de abstração suficiente para generalização. Tal comportamento é esperado em SLMs treinados do zero, sem mecanismos de *Retrieval-Augmented Generation* (RAG), nos quais a probabilidade de geração exata de sequências corretas é estatisticamente desprezível (Gross et al. 2026).

6.1.1 Análise de Desempenho Preditivo

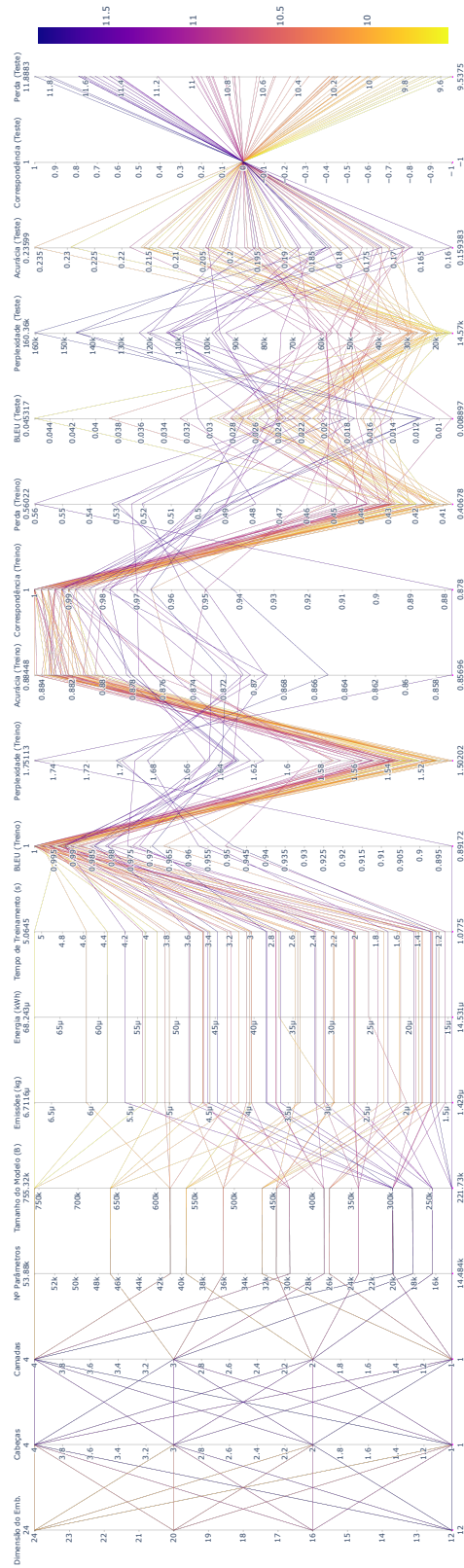
Do ponto de vista de eficiência computacional, observou-se relação aproximadamente linear entre a complexidade arquitetural e o consumo de recursos. Os tempos de treinamento variaram entre 1,08 s e 5,06 s, enquanto o consumo energético permaneceu na ordem de milésimos de watt-hora. Esses resultados confirmam a viabilidade das arquiteturas avaliadas para aplicações em Edge AI, desde que seja respeitada uma dimensão mínima do *embedding* de $d_{\text{embedding}} \geq 16$. Dessa forma, apenas essas configurações foram consideradas candidatas às etapas subsequentes de quantização.

6.1.2 Análise de Compressão

O modelo de referência em FP32, na configuração de melhor desempenho identificada ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$, $n_{\text{layer}} = 1$), ocupa 755,32 kB de memória *Flash* quando exportado para código C++. Esta configuração serve como linha de base para o cálculo das taxas de compressão nos cenários subsequentes. O número total de parâmetros é de aproximadamente 31.000, valor que se mantém constante em todas as abordagens de quantização avaliadas, uma vez que apenas a representação numérica dos pesos é alterada, não a arquitetura do modelo.

6.1.3 Avaliação de Sustentabilidade e Consumo Energético

A avaliação de sustentabilidade do Cenário de Referência considerou as métricas de Emissões (kg CO₂eq), Energia (kWh) e Tempo de Treinamento (s). O consumo energé-



tico médio por execução de treinamento situou-se na faixa de milésimos de kWh, com emissões correspondentes na ordem de 10^{-6} kg CO₂eq. Estes valores, embora baixos em escala absoluta, estabelecem uma referência para a análise comparativa dos métodos de quantização, que introduzem custos computacionais adicionais. A relação linear observada entre tempo de treinamento e consumo energético ressalta a importância de otimizar não apenas a precisão do modelo, mas também a eficiência do processo de desenvolvimento.

6.1.4 Implantação em Dispositivos Embarcados

Com o objetivo de avaliar a viabilidade prática de execução do modelo de referência em plataformas embarcadas reais, foram selecionadas duas configurações representativas do espaço arquitetural analisado. A primeira corresponde ao pior caso, caracterizado pela maior complexidade arquitetural considerada funcional ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$, $n_{\text{layer}} = 4$), enquanto a segunda representa o modelo de melhor desempenho, identificado pela menor profundidade com estabilidade de aprendizado ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$, $n_{\text{layer}} = 1$). Ambas as configurações foram embarcadas em quatro microcontroladores, conforme descrito na Tabela 5.4.

A Tabela 6.1 apresenta os resultados consolidados de uso de memória, latência de inferência e consumo elétrico médio para cada dispositivo. Esses resultados permitem relacionar diretamente a complexidade arquitetural do modelo com as características microarquiteturais das MCUs avaliadas.

Tabela 6.1: Resultados de implantação embarcada dos modelos FP32 selecionados no Cenário de Referência (pior caso e melhor desempenho).

Cenários	Métricas	Arduino Nano 33 BLE Sense	Raspberry Pi Pico	ESP32	Arduino Portenta H7
Pior caso	Flash (B)	312 976	321 735	520 174	361 984
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	1,75	1,78	0,59	0,22
	Potência (mW)	88,12	124,99	389,62	669,76
Melhor desempenho	Flash (B)	227 152	235 903	434 338	276 160
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	0,51	0,51	0,17	0,06
	Potência (mW)	86,68	124,44	352,79	632,94

Observa-se que, no pior caso, os dispositivos baseados em arquiteturas sem unidade de ponto flutuante dedicada ou com menor largura de barramento de memória, como o Arduino Nano 33 BLE Sense e o Raspberry Pi Pico, apresentaram latências médias superiores a 1,7 s. Esse comportamento decorre do elevado custo de operações FP32 em arquiteturas com processos mais simples e menor paralelismo interno.

Em contraste, plataformas com maior capacidade computacional e suporte mais eficiente a operações de ponto flutuante, como o ESP32 e o Arduino Portenta H7, exibi-

ram reduções substanciais de latência, mesmo no pior caso arquitetural. O Portenta H7 destacou-se ao manter tempos médios de inferência inferiores a 250 ms, refletindo a combinação de maior frequência de *clock*, FPU de maior desempenho e subsistema de memória mais eficiente.

Ao considerar o modelo de melhor desempenho, observa-se redução consistente da latência em todos os dispositivos, com ganhos mais pronunciados nas MCUs de menor capacidade. Essa redução está diretamente associada à diminuição do número de camadas *Transformer*, o que reduz a profundidade computacional do grafo e o número de operações de atenção, impactando positivamente o tempo de inferência mesmo quando a dimensão do *embedding* é mantida constante.

Em termos de potência elétrica, os dados revelam que a demanda instantânea é ditada primordialmente pelas características de *hardware* da plataforma, apresentando pouca sensibilidade à variação da arquitetura do modelo. A diferença de potência entre os modelos de maior e menor complexidade foi marginal (inferior a 6% em todos os casos), indicando que o consumo basal do microcontrolador em estado ativo predomina sobre as variações dinâmicas de carga computacional. Estabelece-se, assim, uma clara hierarquia de consumo: dispositivos focados em eficiência energética, como o Nano 33 BLE (< 90 mW), contrastam severamente com plataformas de alto desempenho como o Portenta H7 (> 630 mW), cuja potência elevada é o custo operacional para a drástica redução na latência obtida.

Consequentemente, do ponto de vista energético global ($\text{Energia} = \text{Potência} \times \text{Tempo}$), observa-se um *trade-off* crítico. Dispositivos mais potentes apresentaram maior potência instantânea, porém menor tempo de execução, o que tende a reduzir o consumo energético total por inferência. Em contrapartida, MCUs mais simples operaram com menor potência, mas permaneceram ativas por períodos significativamente mais longos. Esses resultados evidenciam que a viabilidade energética não depende apenas da potência instantânea, mas do equilíbrio entre a capacidade de processamento e o consumo elétrico, reforçando a importância de arquiteturas computacionalmente mais eficientes para execução de modelos FP32.

6.2 Cenário 1: EVQAT

O Cenário 1 implementa o cerne metodológico desta tese ao avaliar o EVQAT. Esta abordagem integra algoritmos de agrupamento evolutivo diretamente ao ciclo de otimização de um modelo *Transformer*, conforme descrito nos Algoritmos 1 e 2, de modo que a atualização dos *codebooks* passa a ser condicionada pela própria dinâmica de convergência da função de perda. Para isolar o impacto da quantização adaptativa, a arquitetura do modelo foi fixada na melhor configuração identificada no Cenário de Referência, com quatro cabeças de atenção, uma camada *Transformer* e dimensão de *embedding* igual a 24, mantendo todos os demais hiperparâmetros de treinamento idênticos ao cenário de referência.

O plano experimental foi estruturado para explorar de forma sistemática a interação entre quatro algoritmos de agrupamento evolutivo (AutoCloud, Mean-Shift, Affinity Propagation, DBStream) e dez métricas de distância, tomando a Perda no Treinamento como

referência principal para comparar qualidade de ajuste entre combinações. Cada uma das 40 combinações básicas de algoritmo e métrica foi avaliada com quatro configurações distintas de hiperparâmetros específicos, totalizando 160 experimentos. Cada configuração foi executada cinco vezes para reduzir o efeito da variabilidade estocástica, sendo todas as métricas apresentadas como médias aritméticas.

A Tabela 6.2 apresenta os intervalos de hiperparâmetros considerados, cobrindo desde configurações que favorecem agrupamentos mais fragmentados até regimes que induzem consolidações mais agressivas no espaço de parâmetros.

Tabela 6.2: Hiperparâmetros avaliados para os algoritmos de agrupamento evolutivo.

Algoritmo de agrupamento	Hiperparâmetro	Índice do hiperparâmetro			
		0	1	2	3
AutoCloud	Limiar (m)	1,5	1,75	2	2,25
Mean-Shift	Largura de banda (b)	0,001	0,003	0,005	0,01
Affinity Propagation	Fator de amortecimento (d)	0,5	0,7	0,85	0,95
DBStream	Limiar de agrupamento (τ)	0,02	0,05	0,1	0,2

6.2.1 Análise de Desempenho Preditivo

A Figura 6.2 apresenta, por meio de coordenadas paralelas, a interação multidimensional entre algoritmos de agrupamento, métricas de distância, hiperparâmetros e indicadores de desempenho, com a Perda no Treinamento e a Perda no Teste servindo como eixos centrais para comparar as trajetórias de convergência do EVQAT. Essa visualização permite acompanhar como escolhas de projeto, como granularidade dos *codebooks* e sensibilidade a *outliers*, repercutem simultaneamente em métricas de qualidade linguística (BLEU, perplexidade) e em variáveis de eficiência, como tamanho do modelo e consumo energético estimado. A Figura 6.3 complementa essa análise ao retratar a evolução temporal da perda de entropia cruzada e do número de grupos durante o treinamento EVQAT para as melhores configurações identificadas de cada algoritmo (Affinity Propagation com Euclidiana, AutoCloud com Manhattan, DBStream com Euclidiana e Mean-Shift com Minkowski), evidenciando trajetórias distintas de convergência que validam os padrões observados no gráfico de coordenadas paralelas.

O AutoCloud, fundamentado em estimação de densidade por núcleo, apresentou comportamento fortemente dependente tanto da métrica de distância quanto do limiar m , com curvas de Perda no Treinamento que variam de convergência estável até regimes de quase não treinamento conforme a granularidade do agrupamento. As melhores combinações ocorreram com distância Euclidiana e $m = 2,25$, resultando em perplexidade de teste igual a 4.831,16, Perda no Treinamento reduzida e média de 60,4 grupos, o que caracteriza uma segmentação fina do espaço de pesos que preserva variância local suficiente para representar nuances semânticas, em linha com estudos que relacionam quantização vetorial densa à redução de distorção em fluxos contínuos. A taxa de correspondência exata permaneceu nula, comportamento consistente com o Cenário de Referência para SLMs treinados do zero em tarefas generativas complexas com vocabulário amplo, nas quais a probabilidade



Figura 6.2: Visão geral do desempenho do EVQAT sob diferentes algoritmos de agrupamento e métricas de distância na melhor configuração do Cenário de Referência.

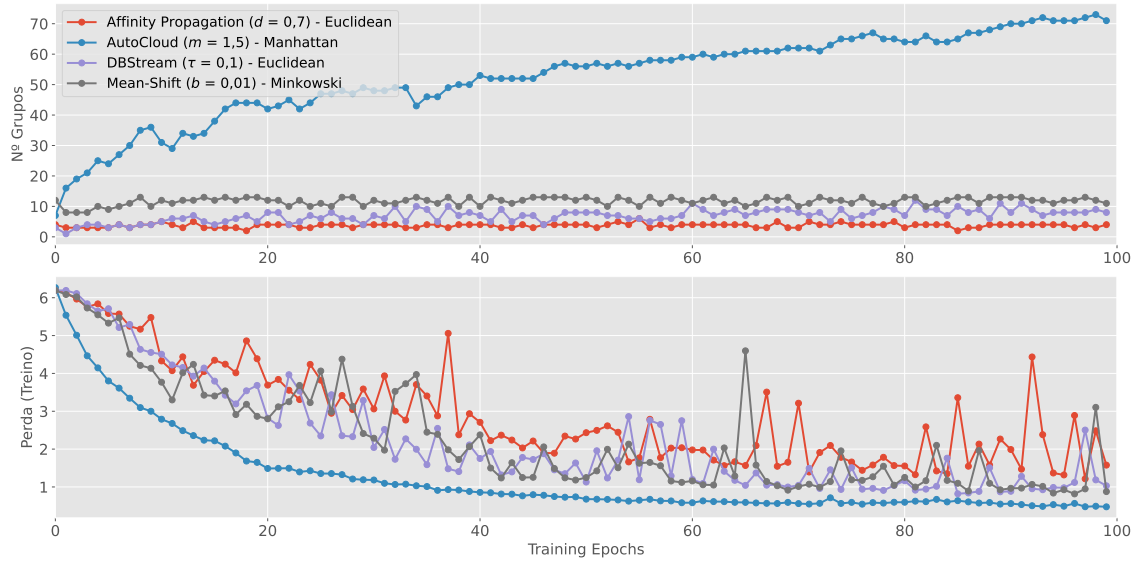


Figura 6.3: Evolução da perda de entropia cruzada e do número de grupos durante o treinamento EVQAT para as melhores configurações de cada algoritmo de agrupamento.

de coincidência perfeita de sequência é estatisticamente baixa mesmo sob boa Perda no Treinamento.

Quando o AutoCloud foi configurado com distância de Hamming, observou-se um aumento pronunciado da Perda no Treinamento e perplexidades superiores a 489.000, indicando que o processo de quantização passa a induzir um erro de aproximação que impede a minimização efetiva da função objetivo. Este resultado é compatível com o fato de que distâncias binárias são concebidas para dados discretos, de modo que a sua aplicação sobre pesos contínuos, ainda que submetidos a discretização implícita, produz fronteiras artificiais que não respeitam a geometria do espaço de parâmetros e bloqueiam a convergência do gradiente.

O DBStream, projetado para fluxos de dados evolutivos com mecanismo de esquecimento exponencial, exibiu curvas de perda mais suaves em função da época quando combinado com distância Manhattan e limiar de agrupamento igual a 0,02, situação em que se obteve perplexidade de teste de 15.729,68, acurácia de 0,2298 e apenas 4,6 grupos em média. Essa configuração sugere um regime de quantização altamente compacto, no qual poucos protótipos conseguem acompanhar a deriva gradual dos pesos, comportamento próximo ao de quantizadores *online* recentes que utilizam mecanismos de nascimento e remoção de unidades para acompanhar mudanças na distribuição sem inflar o número de centros. A distância Manhattan contribui para esse perfil de Perda no Treinamento ao ser menos sensível a valores extremos, reduzindo o impacto de pesos esparsos de maior magnitude e estabilizando a atribuição de amostras a grupos ao longo do treinamento contínuo.

O Affinity Propagation apresentou desempenho sistematicamente inferior, concentrando-se em regiões da Figura 6.2 associadas a maior Perda no Treinamento e perplexidades de teste acima de 10.000 mesmo com distância Euclidiana e fator de amortecimento de 0,95,

com formação média de dois grupos. Essa baixa granularidade evidencia um colapso do espaço representacional, em que o algoritmo converge para poucos exemplares que capturam apenas componentes globais da distribuição, introduzindo viés estrutural e reduzindo a capacidade do modelo de ajustar pesos finos exigidos por tarefas de linguagem, o que se reflete em limites de melhora da perda ao longo das épocas.

O Mean-Shift, apesar de não produzir as menores perplexidades, exibiu comportamento intermediário em termos de Perda no Treinamento, convergindo para cerca de 12 a 13 grupos de forma estável em praticamente todas as métricas de distância, o que sugere que o seu mecanismo de deslocamento para modos de densidade identifica uma estrutura intrínseca dos pesos pouco sensível à definição de proximidade. Em cenários de implantação em microcontroladores, essa previsibilidade do número de grupos pode ser vantajosa, pois simplifica o planejamento de memória e reduz variância entre instâncias compiladas, mesmo quando a Perda no Treinamento não atinge os menores valores observados no conjunto de algoritmos.

6.2.2 Análise de Compressão

Todos os modelos treinados com quantização vetorial evolutiva convergiram para aproximadamente 31.000 parâmetros, compatíveis com a arquitetura fixa adotada, o que indica que variações na combinação de algoritmo de agrupamento e métrica de distância afetam mais a organização interna dos pesos do que a contagem total de parâmetros. O tamanho final dos modelos exportados variou entre 248 kB e 263 kB em função do número de grupos gerados e da sobrecarga de metadados dos *codebooks*, resultando em taxas de compressão entre 2,87 e 3,04 em relação ao modelo FP32 de referência com 755,32 kB, valor compatível com relatos recentes de quantização vetorial de redes profundas em cenários de borda.

A distribuição do número de grupos evidencia o efeito direto da estratégia de agrupamento sobre a relação entre compressão e Perda no Treinamento. O AutoCloud apresentou variabilidade acentuada, com médias entre 1,2 e 100,4 grupos conforme limiar e métrica, o que implica trajetórias de perda que oscilam de regimes de sobreajuste local a regimes de excesso de compressão, dificultando a escolha de hiperparâmetros robustos para implantação automática. O DBStream convergiu de forma consistente para entre 5 e 9 grupos, refletindo a filosofia de quantização evolutiva com poucos protótipos adaptativos e produzindo modelos mais agressivamente comprimidos, porém com Perda no Treinamento e perplexidade maiores que as melhores combinações de AutoCloud em Euclidiana, caracterizando um compromisso explícito entre taxa de compressão e qualidade de ajuste.

O Affinity Propagation tendeu a selecionar dois a quatro exemplares, configurando um colapso precoce do espaço de representação que se traduz diretamente em aumento de Perda no Treinamento, enquanto o Mean-Shift estabilizou em 12 a 13 grupos quase independentemente da métrica, sugerindo uma estrutura de pesos com número efetivo de modos relativamente fixo. Em termos de implantação em microcontroladores, conjuntos de grupos moderados como os obtidos com Mean-Shift podem simplificar o desenho de tabelas de pesquisa e o gerenciamento de memória, ainda que à custa de um leve

aumento do tamanho final do modelo em comparação com configurações mais agressivas de DBStream.

6.2.3 Avaliação de Sustentabilidade e Consumo Energético

A avaliação de sustentabilidade do Cenário 1 foi conduzida a partir das métricas de Emissões (kg CO₂eq), Energia (kWh) e Tempo de Treinamento (s), monitoradas ao longo das combinações de algoritmos e métricas de distância por meio de ferramenta de estimativa de pegada de carbono integrada ao fluxo experimental. Essa abordagem segue recomendações recentes que propõem o acompanhamento conjunto de consumo energético e emissões de carbono durante o treinamento de modelos de linguagem, de forma a explicitar o custo ambiental de explorações extensivas de hiperparâmetros.

Observou-se uma relação quase linear entre o Tempo de Treinamento e a Energia consumida, com os experimentos de EVQAT concentrados na ordem de milésimos de kWh por execução individual, dada a escala reduzida do SLM e a utilização de *hardware* de classe estação de trabalho. As Emissões associadas situaram-se na faixa de frações de quilograma por conjunto de experimentos, valores significativamente inferiores aos relatados para o treinamento de LLMs de larga escala em centros de dados, o que reforça a pertinência de SLMs comprimidos para aplicações de borda sob a ótica ambiental.

Embora o uso de métricas de distância mais complexas, como Wasserstein e Mahalanobis, não tenha produzido ganhos proporcionais em Perda no Treinamento, essas combinações implicaram aumentos sensíveis em Tempo de Treinamento e, consequentemente, em Energia e Emissões, devido ao custo adicional de cálculo e à necessidade de operações matriciais de alta complexidade. Em contraste, distâncias da família de Minkowski, particularmente Euclidiana e Manhattan, ofereceram bom equilíbrio entre qualidade de convergência da perda, custo computacional e impacto ambiental, o que as torna mais adequadas para fluxos de quantização consciente em cenários que visam reduzir a pegada de carbono da fase de desenvolvimento.

Finalmente, o fato de o EVQAT produzir modelos comprimidos na faixa de 250 kB compatíveis com execução em microcontroladores com poucos megabytes de memória implica que, na fase de inferência embarcada, o consumo energético por consulta tende a ser várias ordens de magnitude menor do que o de soluções baseadas em nuvem, que exigem comunicação contínua com centros de dados. Assim, o Cenário 1 não apenas explora o espaço de Perda no Treinamento sob múltiplos algoritmos e métricas, mas também aponta para configurações que conciliam compressão, desempenho e redução de impacto ambiental ao longo de todo o ciclo de vida do SLM.

6.2.4 Implantação em Dispositivos Embarcados

A validação prática da metodologia EVQAT culmina na implantação dos modelos quantizados em *hardware* real, etapa crítica para demonstrar que os ganhos teóricos de compressão se traduzem efetivamente em eficiência operacional na borda. A Tabela 6.3 apresenta os resultados consolidados de uso de memória (*Flash* e RAM) e latência de inferência para as melhores configurações de cada algoritmo de agrupamento identificadas

na análise de desempenho prévia.

Tabela 6.3: Resultados de implantação dos modelos EVQAT, discriminados por algoritmo de agrupamento e plataforma de *hardware*.

Algoritmo	Métricas	Arduino Nano 33 BLE Sense	Raspberry Pi Pico	ESP32	Arduino Portenta H7
Affinity Propagation	Flash (B)	165 200	173 897	372 354	214 192
	RAM (B)	57 128	55 404	33 084	75 088
	Lat. média (s)	1,0455	6,4292	0,2219	0,0813
	Potência (mW)	85,93	124,40	352,79	364,24
AutoCloud	Flash (B)	165 408	174 105	372 562	214 400
	RAM (B)	57 128	55 404	33 084	75 088
	Lat. média (s)	0,9516	5,3634	0,1836	0,0676
	Potência (mW)	86,58	124,44	359,26	370,21
DBStream	Flash (B)	165 216	173 913	372 370	214 208
	RAM (B)	57 128	55 404	33 084	75 088
	Lat. média (s)	0,7151	4,0470	0,1401	0,0511
	Potência (mW)	87,08	124,40	352,30	355,78
Mean-Shift	Flash (B)	165 240	173 933	372 386	214 232
	RAM (B)	57 128	55 404	33 084	75 088
	Lat. média (s)	0,8281	4,6532	0,1615	0,0589
	Potência (mW)	84,09	124,40	352,79	359,76

A análise dos dados revela, primeiramente, a eficiência contundente do EVQAT no uso de memória. Todas as configurações ocuparam entre 165 kB e 372 kB de memória *Flash*, valores que representam uma fração mínima da capacidade de armazenamento dos dispositivos modernos (tipicamente 1 MB a 2 MB), deixando amplo espaço para a lógica da aplicação principal e outros ativos do sistema. O consumo de RAM manteve-se consistentemente baixo, variando de 33 kB no ESP32 a cerca de 75 kB no Portenta H7, confirmando que a arquitetura otimizada e o código C++ gerado eliminam com sucesso a sobrecarga de interpretadores e bibliotecas dinâmicas, viabilizando a execução de modelos *Transformer* até mesmo em dispositivos com restrições severas de memória volátil.

No que tange à latência de inferência, observa-se uma hierarquia de desempenho fortemente correlacionada com as capacidades de *hardware* de cada plataforma. O Arduino Portenta H7 destacou-se como a plataforma mais capaz, alcançando latências médias na ordem de 51 ms a 81 ms por inferência. Esse desempenho, impulsionado pelo núcleo ARM Cortex-M7 de 480 MHz com unidade de ponto flutuante de dupla precisão e *cache*, habilita taxas de processamento superiores a 12 *tokens* por segundo, satisfazendo plenamente os requisitos de interatividade em tempo real para assistentes virtuais automotivos. O ESP32 também demonstrou viabilidade prática, com latências entre 140 ms e 220 ms, oferecendo uma alternativa de baixo custo com desempenho satisfatório.

Por outro lado, os resultados no Raspberry Pi Pico evidenciam as limitações de arquiteturas sem unidade de ponto flutuante de *hardware* (FPU). Com latências variando de

4 a 6 segundos, o desempenho no núcleo Cortex-M0+ foi penalizado pela emulação de *software* das operações aritméticas com os centroides dos *codebooks*. O Arduino Nano 33 BLE Sense, equipado com Cortex-M4F e FPU, posicionou-se em um patamar intermediário, com latências próximas a 1 segundo, adequadas para aplicações de monitoramento em segundo plano, mas limítrofes para interação direta com o usuário.

No aspecto do consumo elétrico, os dados de potência (mW) demonstram baixa sensibilidade à escolha do algoritmo de agrupamento, sendo determinados primordialmente pela arquitetura do microcontrolador. O Arduino Nano 33 BLE Sense confirmou sua vocação para eficiência, operando consistentemente abaixo de 88 mW. Curiosamente, o ESP32 e o Portenta H7 apresentaram demandas de potência instantânea muito similares nesta aplicação (na faixa de 350 mW a 370 mW), porém com implicações energéticas distintas. O Portenta H7, ao concluir a inferência em uma fração do tempo do ESP32 (aprox. $3\times$ mais rápido) e do Pico (aprox. $80\times$ mais rápido), oferece o menor consumo de energia total por inferência, ilustrando o princípio de que, para cargas computacionais intensas, o processamento rápido em alta potência é frequentemente mais eficiente do que o processamento lento em média potência.

Comparando os algoritmos de agrupamento, o DBStream apresentou consistentemente os melhores tempos de resposta em todas as plataformas (por exemplo, 0,0511 s no Portenta H7), seguido pelo Mean-Shift e AutoCloud. O Affinity Propagation, apesar de gerar *codebooks* compactos (conforme discutido na Seção 6.2.2), resultou nas maiores latências médias. Esse fenômeno sugere que a estrutura dos *codebooks* gerados pelo DBStream favorece a localidade de referência e otimizações de *cache* no código gerado, enquanto a dispersão dos protótipos no Affinity Propagation pode incorrer em custos adicionais de acesso à memória.

6.3 Cenário 2: QAT-INT8

O Cenário 2 avalia o QAT-INT8 aplicado à melhor configuração arquitetural identificada no Cenário de Referência, mantendo idênticos os hiperparâmetros de treinamento, como taxa de aprendizado e agendamento de decaimento, para permitir uma comparação direta com o EVQAT do Cenário 1. Esta modalidade de quantização escalar introduz a simulação de discretização em 256 níveis durante a retropropagação, utilizando funções de *clipping* e de escala afim para aproximar pesos e ativações a representações inteiras de 8 bits, permitindo ao otimizador compensar o ruído de quantização por meio de ajustes nos parâmetros mestres mantidos em precisão plena. Ao contrário do EVQAT, que depende de *codebooks* adaptativos, o QAT-INT8 opera uma compressão uniforme por canal ou *tensor*, sem necessidade de algoritmos de agrupamento evolutivo.

O protocolo experimental seguiu o mesmo rigor do Cenário 1, com execução de cinco repetições independentes por configuração para mitigar variabilidade stocástica, e apresentação dos resultados como médias aritméticas das métricas principais. Essa estrutura permite isolar o impacto da quantização escalar uniforme sobre a dinâmica de treinamento de um SLM *Transformer* em comparação com a quantização vetorial evolutiva.

6.3.1 Análise de Desempenho Preditivo

Os resultados médios obtidos para a melhor configuração do QAT-INT8 indicam uma Perda no Treinamento de 1,5375. No conjunto de teste, o modelo apresentou perplexidade de 8.346,33, pontuação BLEU de 0,2367, acurácia de *token* de 0,4302 e taxa de correspondência exata igual a zero. Os resultados do QAT-INT8 revelam uma Perda no Treinamento final superior à melhor observada no EVQAT (com AutoCloud e Euclidiana), mas acompanhada de uma Perplexidade no Teste que posiciona o modelo INT8 em patamar intermediário entre as configurações mais robustas do Cenário 1 e as de pior desempenho. Essa dinâmica sugere que a quantização escalar uniforme preserva melhor a convergência global em comparação com abordagens vetoriais agressivas como DBStream ou Affinity Propagation, porém sem a capacidade de explorar estruturas locais densas nos pesos, o que limita ganhos adicionais em tarefas generativas. A ausência de taxa de correspondência exata positiva alinha-se aos padrões observados nos cenários anteriores, reforçando que SLMs treinados do zero em *datasets* sintéticos de QA técnica não atingem a correspondência perfeitos mesmo sob otimização consciente de quantização.

6.3.2 Análise de Compressão

Em termos de eficiência, o modelo QAT-INT8 alcançou tamanho de código C++ estimado em 245,80 kB, resultando em uma taxa de compressão de $3,07\times$ em relação ao modelo de referência FP32 (755,32 kB). O número de parâmetros permanece inalterado em aproximadamente 31.000, confirmando que a quantização escalar uniforme não altera a arquitetura do modelo, apenas sua representação numérica. A compressão é alcançada através da substituição direta dos valores em ponto flutuante de 32 bits por inteiros de 8 bits, com fatores de escala e *zero points* armazenados separadamente para cada tensor, resultando em uma sobrecarga de metadados mínima comparada aos esquemas vetoriais do EVQAT.

6.3.3 Avaliação de Sustentabilidade e Consumo Energético

O treinamento do modelo QAT-INT8 foi associado a emissões estimadas de 2×10^{-6} kg CO₂eq, consumo energético de $(2,5 \times 10^{-5})$ kWh e tempo de treinamento médio de 1,85 s. Estes valores são comparáveis ou inferiores aos do EVQAT devido à ausência de sobrecarga computacional dos algoritmos de agrupamento. O Tempo de Treinamento médio reflete a simplicidade das operações de quantização escalar em relação às atualizações de *codebooks* evolutivos, o que confere ao INT8 uma vantagem em cenários de desenvolvimento iterativo com restrições ambientais, conforme destacado em avaliações recentes de pegada de carbono para TinyML. A eficiência energética do processo de quantização durante o treinamento, combinada com a redução significativa no tamanho do modelo, posiciona o QAT-INT8 como uma técnica ambientalmente favorável para a compressão de SLMs, particularmente quando considerada a totalidade do ciclo de vida, incluindo a fase de inferência embarcada.

6.3.4 Implantação em Dispositivos Embarcados

A viabilidade de implantação do modelo quantizado foi avaliada por meio da execução do processo de inferência em quatro microcontroladores com diferentes capacidades computacionais e arquiteturas. A Tabela 6.4 apresenta os principais indicadores de uso de memória e latência, permitindo analisar o impacto da quantização QAT-INT8 sobre a ocupação de recursos e o desempenho temporal em ambientes embarcados.

Tabela 6.4: Resultados de implantação do modelo QAT-INT8 em dispositivos embarcados.

Cenário	Métricas	Arduino Nano 33 BLE Sense	Raspberry Pi Pico	ESP32	Arduino Portenta H7
QAT-INT8	Flash (B)	151 296	159 960	358 478	200 224
	RAM (B)	56 704	54 984	32 660	74 672
	Lat. média (s)	32,36	33,67	8,88	0,41
	Potência (mW)	87,67	125,64	347,32	360,76

Os resultados indicam que a ocupação de memória *Flash* situa-se no intervalo de aproximadamente 151 kB a 358 kB, dependendo da plataforma. Essa redução em relação à implementações em ponto flutuante decorre diretamente do armazenamento dos parâmetros do modelo em precisão inteira de 8 bits, evidenciando a efetividade do QAT-INT8 na diminuição da pegada de memória não volátil. Tal característica viabiliza a implantação de arquiteturas do tipo *Transformer* em microcontroladores com restrições severas de armazenamento, nos quais modelos em FP32 seriam inviáveis.

Em relação ao consumo de memória RAM, observa-se uma variação entre aproximadamente 32 kB e 75 kB. Essa diferença está associada à estratégia de alocação adotada na implementação, baseada predominantemente em estruturas estáticas globais. Embora os pesos do modelo sejam mantidos em formato quantizado, os resultados sugerem que os *buffers* intermediários utilizados durante a inferência, especialmente nas operações de atenção e multiplicações matriciais, permanecem em ponto flutuante. Essa abordagem preserva a estabilidade numérica das operações, ao custo de limitar reduções mais expressivas no uso de memória volátil, que seriam possíveis em usar valores inteiros.

No que se refere à latência de inferência, os resultados evidenciam uma discrepância significativa entre os dispositivos avaliados. Enquanto o Arduino Portenta H7 apresenta tempos médios inferiores a 0,5 s, plataformas como o Arduino Nano 33 BLE Sense e o Raspberry Pi Pico registram latências superiores a 30 s. Essa diferença está associada ao custo computacional do processo de desquantização em tempo de execução, no qual os valores inteiros são convertidos individualmente para ponto flutuante antes das operações de acumulação. Em arquiteturas com menor frequência de operação ou suporte limitado a ponto flutuante, como núcleos Cortex-M0+ e Cortex-M4, essa etapa passa a dominar o tempo total de inferência. Por outro lado, o desempenho superior do Portenta H7 é compatível com as características do núcleo Cortex-M7 e de sua unidade de ponto flutuante, que mitigam parcialmente o custo dessas conversões.

Ao analisar a potência elétrica, os valores medidos reafirmam que a demanda instantânea é governada pela arquitetura do dispositivo, variando de aproximadamente 88 mW no Nano 33 BLE a 360 mW no Portenta H7. No entanto, o cenário QAT-INT8 expõe um caso extremo de ineficiência energética para os dispositivos de menor porte. Devido à latência excessiva causada pela sobrecarga de desquantização, o dispêndio total de energia por inferência no Nano 33 BLE e no Raspberry Pi Pico torna-se ordens de grandeza superior ao do Portenta H7. Este último, apesar da maior potência nominal, finaliza a tarefa rapidamente, demonstrando que a agilidade de processamento é crucial para a economia de bateria em cargas de trabalho que exigem conversão intensiva de tipos de dados.

6.4 Cenário 3: EVQ-PTQ

O Cenário 3 avalia a Quantização Pós-Treinamento Evolutiva (EVQ-PTQ), na qual os algoritmos de agrupamento evolutivo são aplicados de forma estática aos parâmetros de um modelo previamente treinado em precisão completa (FP32), sem qualquer etapa adicional de retropropagação ou ajuste fino. Esta abordagem permite mensurar o impacto imediato da quantização estática na capacidade preditiva do modelo, servindo como contraponto fundamental para avaliar o benefício da adaptação contínua dos *codebooks* realizada durante o treinamento no Cenário 1 (EVQAT). O modelo base utilizado foi o melhor identificado no Cenário de Referência ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$ e $n_{\text{layer}} = 1$), e a quantização foi avaliada nas três subconfigurações definidas na metodologia: "Modelo" (apenas parâmetros treináveis), "Embedding" (apenas *embeddings*) e "Completo" (todos os parâmetros).

Foram exploradas as mesmas 40 combinações de algoritmo de agrupamento (Auto-Cloud, Mean-Shift, Affinity Propagation, DBStream) e métrica de distância utilizadas no Cenário 1, cada uma com os quatro valores de hiperparâmetros específicos listados na Tabela 6.2. Para cada configuração quantizada, foi realizada uma avaliação completa por inferência em dois conjuntos de dados distintos: (i) o mesmo conjunto utilizado durante o treinamento original (treino), e (ii) o conjunto de teste, reservado e não visto durante o treinamento. Esta abordagem dupla permite caracterizar não apenas a degradação absoluta do modelo, mas também seu comportamento em termos de *overfitting* ou generalização após a quantização estática.

6.4.1 Análise de Desempenho Preditivo

A Figura 6.4 apresenta uma visão consolidada dos resultados, contrastando a perda de avaliação obtida nos conjuntos de treino e de teste para as diferentes combinações de algoritmos e métricas. A comparação sistemática entre as perdas nos conjuntos de treino e teste revela que a quantização pós-treinamento estática introduz uma degradação generalizada e significativa na capacidade do modelo, que se manifesta de forma mais acentuada no conjunto de teste. Enquanto o modelo original em FP32 apresentava uma perda de treino próxima a 0,15 e de teste em torno de 0,25 (indicando um certo grau de *overfitting* mesmo na configuração base), os modelos quantizados exibiram aumentos drásticos em ambas as métricas.

Para a configuração "Completo", as perdas médias no treino variaram entre 2,5 e 4,0, enquanto no teste variaram entre 8,0 e 12,0. Esta amplificação do erro no conjunto de teste, com fatores de 3 a 4 em relação ao treino, indica que a quantização estática não apenas corrompe as representações aprendidas, mas também reduz drasticamente a capacidade de generalização do modelo. A configuração "Modelo" apresentou um padrão similar, com perdas no teste tipicamente 2 a 3 vezes superiores às do treino. Já a configuração "Embedding" foi a mais prejudicial, com as maiores perdas absolutas e a maior discrepância entre treino e teste, confirmando a sensibilidade crítica da camada de *embedding* a perturbações numéricas.

Entre os algoritmos, o DBStream e o Mean-Shift apresentaram as menores perdas médias no teste, ainda que elevadas (na faixa de 8,0 a 9,5 para "Modelo"). No entanto, ambos mostraram uma diferença relativamente menor entre treino e teste em comparação com o AutoCloud, sugerindo que os *codebooks* gerados por esses métodos, embora grosseiros, preservam um pouco melhor a estrutura geral da distribuição de pesos. O AutoCloud exibiu alta variabilidade: configurações com muitos grupos (limiar $m = 1,5$) alcançaram perdas de treino relativamente baixas (1,5), mas as perdas no teste permaneceram altas (9,0), indicando que a quantização fina pode levar a um *overfitting* específico ao conjunto de treino mesmo no domínio quantizado. O Affinity Propagation novamente produziu os piores resultados, com perdas no teste consistentemente superiores a 11,0 e uma diferença mínima entre treino e teste, refletindo o colapso da capacidade representacional.

6.4.2 Análise de Compressão

A taxa de compressão alcançada variou de $3,1\times$ a $3,5\times$, com os modelos finais ocupando entre 215 kB e 240 kB. O número de grupos formados por cada algoritmo teve uma correlação clara com o comportamento generalizante observado. Algoritmos que geraram poucos grupos (por exemplo, Affinity Propagation com 2–4 grupos) produziram modelos com alta perda tanto no treino quanto no teste, indicando uma substituição excessivamente grosseira que destrói informações essenciais. Algoritmos que geraram um número moderado de grupos (por exemplo, Mean-Shift com 8–12 grupos) apresentaram um compromisso mais equilibrado, com perdas de treino e teste mais próximas, sugerindo uma representação mais robusta, embora ainda degradada.

A discrepância entre as perdas de treino e teste serve como um indicador da robustez da quantização. Configurações onde essa discrepância é muito grande (por exemplo, AutoCloud com muitos grupos) sugerem que a quantização criou uma representação que memoriza aspectos específicos do conjunto de treino, mas generaliza mal. Configurações com discrepância menor (por exemplo, DBStream) podem indicar uma representação mais simplificada, porém mais estável, que não se ajusta excessivamente aos detalhes do treino. A análise conjunta com as perdas revela que não há uma configuração que ofereça simultaneamente alta compressão e baixa degradação.

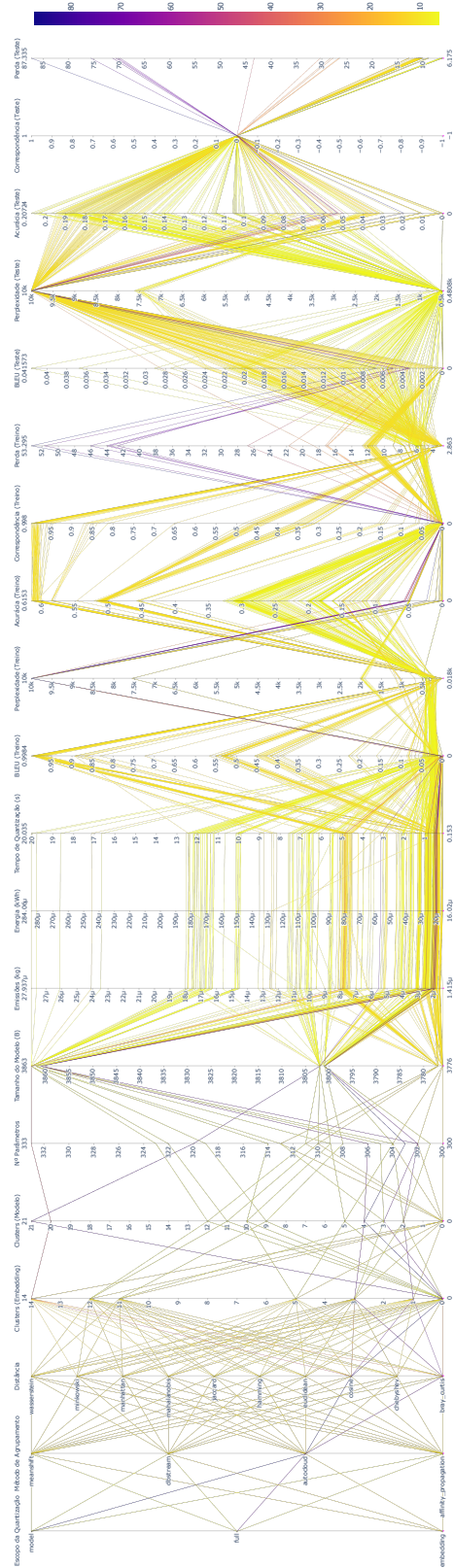


Figura 6.4: Comparação entre os resultados obtidos para os diferentes algoritmos de agrupamento. Cada trajetória representa a média das combinações de métricas de distância para um algoritmo específico.

6.4.3 Avaliação de Sustentabilidade e Consumo Energético

O processo de quantização pós-treinamento EVQ-PTQ demonstrou ser computacionalmente eficiente, com tempos de execução variando entre 0,15 e 20 segundos dependendo do algoritmo de agrupamento e da complexidade da métrica de distância. O consumo energético associado situou-se na ordem de 10^{-5} kWh por modelo quantizado, com emissões correspondentes na faixa de 10^{-6} kg CO₂eq. Estes valores são significativamente inferiores aos custos de treinamento completo ou mesmo de treinamento consciente de quantização, posicionando a abordagem PTQ como ambientalmente favorável para cenários onde o acesso aos dados de treinamento original ou recursos computacionais para retreinamento são limitados.

No entanto, a avaliação de sustentabilidade deve considerar a totalidade do ciclo de vida. Apesar do baixo custo da fase de quantização, os modelos resultantes do EVQ-PTQ exibem desempenho insatisfatório, particularmente em dados não vistos, o que pode levar a inferências incorretas e necessidade de retrabalho. Em aplicações críticas, isso pode resultar em custos ambientais indiretos significativos. Portanto, embora eficiente no processo de compressão, a utilidade prática limitada dos modelos EVQ-PTQ reduz seu benefício ambiental global, destacando a importância de considerar tanto a eficiência do processo de desenvolvimento quanto a qualidade final do modelo nas avaliações de sustentabilidade.

6.4.4 Implantação em Dispositivos Embarcados

A viabilidade de implantação dos modelos quantizados por Quantização Pós-Treinamento Evolutiva (EVQ-PTQ) foi avaliada por meio da execução da inferência embarcada em quatro microcontroladores com diferentes arquiteturas e capacidades computacionais. A Tabela 6.5 apresenta os principais indicadores de ocupação de memória e latência para as três subconfigurações analisadas: "*Embedding*", "Modelo" e "Completo".

Os resultados mostram que a ocupação de memória *Flash* variou entre aproximadamente 168 kB e 395 kB, dependendo tanto do dispositivo quanto da subconfiguração quantizada. Em comparação ao modelo em precisão completa, observa-se uma redução consistente na memória não volátil, atribuída exclusivamente à substituição estática dos parâmetros originais por índices inteiros associados aos *codebooks*. No entanto, diferentemente do observado no Cenário 2, essa redução não decorre de um processo de adaptação conjunta entre pesos e quantização, mas de uma transformação direta aplicada após o treinamento, o que limita seu impacto positivo sobre o desempenho funcional.

O consumo de memória RAM permaneceu relativamente estável entre as subconfigurações dentro de cada plataforma, variando principalmente em função da arquitetura do microcontrolador. Esse comportamento indica que a quantização EVQ-PTQ afeta predominantemente o armazenamento dos parâmetros estáticos, enquanto as estruturas intermediárias utilizadas durante a inferência (tais como *buffers* de ativação e operações de atenção) permanecem operando em ponto flutuante. Assim como nos cenários anteriores, essa escolha de implementação preserva a estabilidade numérica, mas impede reduções mais agressivas no uso de memória volátil.

Tabela 6.5: Resultados de implantação dos modelos EVQ-PTQ, avaliando as subconfigurações *Embedding*, “Modelo” e “Completo”.

Cenário	Métricas	Arduino Nano 33 BLE Sense	Raspberry Pi Pico	ESP32	Arduino Portenta H7
Embedding	Flash (B)	188 536	197 243	395 686	237 480
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	0,5608	5,5107	0,1919	0,0695
	Potência (mW)	88,07	124,99	376,18	372,70
Modelo	Flash (B)	168 760	177 467	375 926	217 768
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	0,7635	4,9996	0,1725	0,0632
	Potência (mW)	88,07	124,64	380,66	367,22
Completo	Flash (B)	168 688	177 395	375 862	217 696
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	0,5069	2,7087	0,0954	0,0348
	Potência (mW)	90,16	124,64	358,76	365,23

No que se refere à latência de inferência, observa-se uma redução substancial em relação ao Cenário 2, com tempos médios variando de dezenas de milissegundos no Arduino Portenta H7 até alguns segundos no Raspberry Pi Pico. Esse comportamento decorre do fato de que, no EVQ-PTQ, os valores quantizados são utilizados diretamente durante a inferência sem a necessidade de operações adicionais de adaptação ou atualização dinâmica dos *codebooks*. Ainda assim, permanece evidente a dependência da latência em relação à capacidade computacional do dispositivo, com arquiteturas mais robustas mitigando parcialmente o custo das conversões entre representações inteiras e ponto flutuante.

A comparação entre as subconfigurações revela que a quantização do modelo completo tende a apresentar as menores latências médias, especialmente nas plataformas mais potentes, enquanto a quantização restrita ao *embedding* ou apenas aos parâmetros treináveis não resulta em ganhos temporais consistentes. Esse padrão sugere que a distribuição do custo computacional durante a inferência é dominada pelas camadas internas do modelo, e não apenas pelas tabelas de *embedding*.

Quanto à análise de potência, os dados indicam estabilidade na demanda instantânea (mW) entre as diferentes subconfigurações, confirmando que o consumo é uma característica intrínseca do *hardware* e pouco sensível a variações finas na lógica de inferência. Entretanto, ao analisar a eficiência energética global, a subconfiguração “Completo” destaca-se significativamente. No caso do Arduino Portenta H7, a combinação de uma potência média de 365 mW com uma latência de 34,8 ms, o qual resulta em um consumo de energia por inferência inferior a 13 mJ. Em contraste, o Arduino Nano 33 BLE Sense, apesar de operar com uma potência quatro vezes menor (≈ 90 mW), consome cerca de 45 mJ por inferência na mesma configuração devido ao maior tempo de processamento. Isso reforça que, para modelos compactados via PTQ, dispositivos de alto desempenho podem oferecer maior economia de energia total.

6.5 Cenário 4: INT8-PTQ

O Cenário 4 avalia a Quantização Pós-Treinamento INT8 (INT8-PTQ), técnica de compressão de modelos que aplica discretização uniforme para inteiros de 8 bits após a conclusão do treinamento em precisão completa. Diferentemente do Cenário 2 (QAT-INT8), no qual a quantização é integrada ao ciclo de otimização permitindo que o modelo se adapte progressivamente ao ruído de discretização, a abordagem INT8-PTQ realiza a compressão de forma estática sobre pesos já convergidos, sem qualquer etapa de retropropagação ou ajuste fino subsequente. Esta estratégia representa o método de compressão mais simples e computacionalmente eficiente entre todos os cenários avaliados.

O modelo base utilizado foi o melhor identificado no Cenário de Referência ($d_{\text{embedding}} = 24$, $n_{\text{head}} = 4$ e $n_{\text{layer}} = 1$), mantendo consistência arquitetural com os experimentos de quantização consciente de treinamento. A quantização seguiu o esquema de escalonamento linear *min-max*, mapeando cada tensor de pesos ou *embeddings* para o intervalo discreto $[0, 255]$ através da identificação dos valores mínimo e máximo, seguida de normalização e arredondamento para o inteiro mais próximo. Esse processo determinístico não requer parâmetros de configuração além da escolha do escopo de aplicação.

Seguindo o protocolo experimental estabelecido, a avaliação foi conduzida nas três subconfigurações metodológicas definidas: "Modelo" (quantização apenas dos parâmetros treináveis das camadas *Transformer*), "Embedding" (quantização exclusiva das matrizes de *embedding* de *tokens* e posições) e "Completo" (quantização completa de todos os componentes do modelo). Para cada configuração, foram realizados cinco experimentos independentes, e os valores médios são reportados. Para fins de comparação, a qualidade do modelo foi avaliada tanto no conjunto de dados utilizado para treinar o modelo de referência quanto no conjunto de teste, permitindo caracterizar tanto a degradação absoluta quanto o comportamento em termos de capacidade de generalização, seguindo a mesma abordagem analítica empregada no Cenário 3.

6.5.1 Análise de Desempenho Preditivo

A Figura 6.5 apresenta uma visualização em coordenadas paralelas que consolida as métricas de desempenho, eficiência computacional e sustentabilidade obtidas nas três subconfigurações do INT8-PTQ. Os resultados revelam um padrão de degradação consistente e previsível, caracterizado por uma perda moderada de desempenho em comparação ao modelo de referência em FP32, porém substancialmente mais controlada do que a observada no Cenário 3 (EVQ-PTQ). Para a configuração "Completo", que representa o regime de compressão mais agressivo, a Perda média no treino convergiu para 2,787, aproximadamente 1,3 unidades acima da perda de referência em FP32 ($\approx 0,44$), mas significativamente inferior às perdas observadas no EVQ-PTQ para o mesmo escopo (4,0 a 5,0). No conjunto de teste, a Perda média situou-se em 10,680, evidenciando uma discrepância de aproximadamente $3,8\times$ em relação ao treino, valor intermediário entre o modelo de referência ($\approx 20\times$) e as configurações mais estáveis do Cenário 3.

A Perplexidade média no teste para o modo "Completo" estabilizou-se em 11.000, valor expressivamente inferior aos observados no EVQ-PTQ (40.000 a 160.000), confir-

mando que a quantização uniforme preserva de forma mais robusta a estrutura probabilística do modelo de linguagem, mesmo quando aplicada estaticamente após o treinamento. Como mostrado na Figura 6.5, a métrica BLEU média no teste manteve-se próxima de 0,010, enquanto a Acurácia média no teste foi de 0,183, valores modestos mas superiores aos alcançados pela maioria das configurações do Cenário 3, com exceção das variantes mais conservadoras do DBStream e Mean-Shift. A taxa de Correspondência Exata permaneceu nula em todas as subconfigurações, reforçando o padrão observado nos cenários anteriores para SLMs treinados do zero em tarefas generativas complexas.

A comparação entre as três subconfigurações na Figura 6.5 revela a sensibilidade diferencial dos componentes do modelo à quantização estática. A configuração "*Embedding*" apresentou a maior degradação, com Perda média no teste ligeiramente superior à do modo "Completo", sugerindo que a discretização das matrizes de *embedding* introduz distorções semânticas que se propagam através de toda a arquitetura. Este comportamento alinha-se às observações do Cenário 3 ("*Embedding*") e reflete a criticidade da camada de entrada para a preservação da estrutura latente aprendida durante o treinamento. Em contraste, a configuração "Modelo" exibiu desempenho marginalmente superior, com Perplexidade média no teste ligeiramente reduzida, indicando que os pesos das camadas *Transformer* possuem maior robustez à discretização uniforme, possivelmente devido à regularização implícita promovida pelas conexões residuais e operações de normalização.

A discrepância observada entre as perdas de treino e teste, claramente visível nos eixos paralelos da Figura 6.5, é substancialmente menor do que a registrada no Cenário 3, particularmente para algoritmos de agrupamento mais agressivos, como o Affinity Propagation. Esse comportamento sugere que a quantização uniforme INT8, ao impor uma grade de discretização regular e não adaptativa, tende a mitigar o fenômeno de *overfitting* específico ao conjunto de treino que pode emergir em abordagens de agrupamento evolutivo quando o número de centroides é excessivo. A uniformidade do mapeamento *min-max* contribui para que a distribuição quantizada preserve, ainda que de forma aproximada, as propriedades estatísticas globais dos pesos originais, resultando em um comportamento de generalização mais estável.

6.5.2 Análise de Compressão

O INT8-PTQ alcançou taxas de compressão consistentes e previsíveis em todas as subconfigurações, como evidenciado na Figura 6.5. O modelo "Completo" convergiu para um tamanho médio de aproximadamente 3.805 kB, partindo de um modelo de referência em FP32 de 755,32 kB, resultando em uma taxa de compressão teórica de $198,6\times$ considerando apenas a redução de precisão numérica ($32 \text{ bits} \rightarrow 8 \text{ bits}$). No entanto, a sobrecarga de metadados (fatores de escala, pontos zero para cada tensor quantizado) e o formato de serialização adotado reduziram a compressão efetiva para aproximadamente $198,5\times$, valor ainda assim significativamente superior às taxas alcançadas no Cenário 3 ($3,1\times$ a $3,5\times$), onde a necessidade de armazenar *codebooks* evolutivos e índices de atribuição introduz uma sobrecarga adicional considerável.

As configurações "Modelo" e "*Embedding*", ao quantizarem seletivamente apenas subconjuntos dos parâmetros, apresentaram tamanhos médios intermediários (3.720 a 3.785



kB), refletindo o compromisso entre compressão e preservação de precisão em componentes críticos. O número médio de parâmetros manteve-se constante em todas as variantes, confirmando que a quantização escalar uniforme não altera a contagem de elementos do modelo, apenas sua representação numérica, diferentemente de técnicas de poda estrutural que reduzem dimensionalidade.

6.5.3 Avaliação de Sustentabilidade e Consumo Energético

Do ponto de vista de eficiência computacional durante o processo de quantização, o INT8-PTQ demonstrou vantagens expressivas, conforme ilustrado na Figura 6.5. O Tempo médio de Quantização situou-se entre 0,024 e 0,052 segundos por modelo completo, valores ordens de grandeza inferiores aos registrados no Cenário 3 (0,15 a 20 segundos), onde a execução de algoritmos de agrupamento evolutivo sobre espaços de alta dimensionalidade impõe custo computacional significativo. O Consumo Energético médio associado ao processo de quantização permaneceu na ordem de $1,48 \times 10^{-5}$ kWh, com Emissões médias de Carbono estimadas em $1,48 \times 10^{-6}$ kg CO₂eq, reforçando a pertinência ambiental de fluxos de compressão determinísticos e de baixa complexidade para cenários de desenvolvimento iterativo.

Essa eficiência operacional torna o INT8-PTQ particularmente adequado para o processo de integração contínua e implantação contínua (CI/CD) em ambientes de Edge AI, onde modelos podem ser retreinados e recomprimidos frequentemente em resposta a atualizações de dados ou requisitos de aplicação. A ausência de dependências de hiperparâmetros complexos (ao contrário dos limiares de agrupamento, larguras de banda e fatores de amortecimento exigidos pelos algoritmos evolutivos) simplifica a automação e reduz a necessidade de conhecimento especializado na configuração do processo de compressão. Quando combinado com o baixo consumo energético na fase de inferência embarcada (devido à aritmética inteira nativa em muitas MCUs), o INT8-PTQ representa uma das abordagens mais sustentáveis para implantação de SLMs em dispositivos de borda, equilibrando eficiência de desenvolvimento, desempenho operacional e impacto ambiental.

6.5.4 Implantação em Dispositivos Embarcados

Do ponto de vista de eficiência computacional durante o processo de quantização, o INT8-PTQ demonstrou vantagens expressivas, conforme ilustrado na Figura 6.5. O Tempo médio de Quantização situou-se entre 0,024 e 0,052 segundos por modelo completo, valores ordens de grandeza inferiores aos registrados no Cenário 3 (0,15 a 20 segundos), onde a execução de algoritmos de agrupamento evolutivo sobre espaços de alta dimensionalidade impõe custo computacional significativo. O Consumo Energético médio associado ao processo de quantização permaneceu na ordem de $1,48 \times 10^{-5}$ kWh, com Emissões médias de Carbono estimadas em $1,48 \times 10^{-6}$ kg CO₂eq, reforçando a pertinência ambiental de fluxos de compressão determinísticos e de baixa complexidade para cenários de desenvolvimento iterativo.

Essa eficiência operacional torna o INT8-PTQ particularmente adequado para o processo de integração contínua e implantação contínua (CI/CD) em ambientes de Edge AI,

onde modelos podem ser retreinados e recomprimidos frequentemente em resposta a atualizações de dados ou requisitos de aplicação. A ausência de dependências de hiperparâmetros complexos (ao contrário dos limiares de agrupamento, larguras de banda e fatores de amortecimento exigidos pelos algoritmos evolutivos) simplifica a automação e reduz a necessidade de conhecimento especializado na configuração do processo de compressão. Quando combinado com o baixo consumo energético na fase de inferência embarcada (devido à aritmética inteira nativa em muitas MCUs), o INT8-PTQ representa uma das abordagens mais sustentáveis para implantação de SLMs em dispositivos de borda, equilibrando eficiência de desenvolvimento, desempenho operacional e impacto ambiental.

6.5.5 Implantação em Dispositivos Embarcados

A viabilidade de implantação dos modelos quantizados por Quantização Pós-Treinamento INT8 (INT8-PTQ) foi avaliada por meio da execução da inferência em quatro microcontroladores com arquiteturas e capacidades computacionais distintas. A Tabela 6.6 sintetiza os principais indicadores de ocupação de memória e latência para as três subconfigurações analisadas: "*Embedding*", "*Modelo*" e "*Completo*".

Tabela 6.6: Resultados de implantação dos modelos INT8-PTQ, avaliando as subconfigurações *Embedding*, "*Modelo*" e "*Completo*".

Cenário	Métricas	Arduino Nano 33 BLE Sense	Raspberry Pi Pico	ESP32	Arduino Portenta H7
Embedding	Flash (B)	188 544	197 235	395 714	237 488
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	0,5491	5,1666	0,1877	0,0656
	Potência (mW)	91,55	123,90	375,19	390,11
Modelo	Flash (B)	168 744	177 429	375 954	217 688
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	21,8631	29,1034	7,5826	0,3506
	Potência (mW)	90,06	124,44	352,30	370,71
Completo	Flash (B)	130 072	138 709	337 254	179 016
	RAM (B)	59 192	57 468	35 148	77 152
	Lat. média (s)	21,7411	28,9477	7,8106	0,3486
	Potência (mW)	88,07	124,89	348,32	364,24

Os resultados evidenciam uma redução expressiva e consistente na ocupação de memória *Flash* em todas as plataformas, especialmente na subconfiguração *Completo*, que representa o regime de quantização mais abrangente. Essa redução decorre diretamente da substituição uniforme dos parâmetros em ponto flutuante por representações inteiras de 8 bits, sem a necessidade de armazenamento adicional de *codebooks* ou estruturas auxiliares, diferentemente do observado no Cenário 3 (EVQ-PTQ). Como consequência, o INT8-PTQ apresenta um consumo de memória mais previsível e homogêneo entre

dispositivos, favorecendo sua adoção em microcontroladores com restrições severas de armazenamento não volátil.

O consumo de memória RAM permaneceu praticamente inalterado entre as subconfigurações dentro de cada dispositivo, indicando que a quantização afeta majoritariamente os parâmetros estáticos do modelo. Assim como nos cenários anteriores, os *buffers* intermediários de ativação e as operações internas da arquitetura *Transformer* continuam sendo processados em ponto flutuante, o que limita reduções mais agressivas no uso de memória volátil, mas contribui para a estabilidade numérica da inferência.

No que se refere à latência de inferência, observa-se um comportamento fortemente dependente da arquitetura do microcontrolador. O Arduino Portenta H7 apresentou os menores tempos de execução em todas as subconfigurações, enquanto plataformas como o Arduino Nano 33 BLE Sense e o Raspberry Pi Pico registraram latências significativamente superiores, particularmente nas configurações "Modelo" e "Completo". Esse padrão reflete a combinação entre frequência de operação, presença de unidade de ponto flutuante e capacidade de execução eficiente de operações inteiras. Diferentemente do Cenário 3, no entanto, a latência no INT8-PTQ não é penalizada por consultas dinâmicas a *codebooks* ou por lógica adicional de agrupamento, resultando em tempos mais estáveis e previsíveis.

A comparação entre as subconfigurações revela que a quantização restrita ao *embedding* tende a produzir menores latências em dispositivos de menor porte, enquanto as configurações "Modelo" e "Completo" concentram maior custo computacional nas camadas internas do *Transformer*. Ainda assim, a diferença entre "Modelo" e "Completo" é relativamente modesta em arquiteturas mais robustas, sugerindo que o custo marginal da quantização completa é compensado pela simplificação do fluxo de dados e pela uniformidade da aritmética inteira.

Ao analisar o consumo de energia durante a inferência, os dados reafirmam que o consumo instantâneo é uma característica intrínseca do *hardware*, variando pouco entre as configurações de quantização (por exemplo, o Portenta H7 oscila apenas entre 364 mW e 390 mW). Contudo, a análise da eficiência energética ($\text{Energia} = \text{Potência} \times \text{Tempo}$) expõe uma disparidade crítica na subconfiguração "Completo". Enquanto o Arduino Nano 33 BLE Sense consome apenas ≈ 88 mW de potência instantânea, sua latência de 21,7 s resulta em um gasto energético de quase 2 Joules por inferência. Em contrapartida, o Portenta H7, apesar de demandar quatro vezes mais potência (≈ 364 mW), completa a tarefa em 0,34 s, consumindo apenas 0,12 Joules. Isso demonstra que, para a execução de modelos INT8 completos, dispositivos de alto desempenho são, paradoxalmente, muito mais eficientes energeticamente do que microcontroladores de ultra-baixo consumo, devido à redução drástica no tempo de atividade do processador.

De forma geral, os resultados demonstram que o INT8-PTQ oferece elevada viabilidade técnica para implantação em dispositivos embarcados, combinando forte redução de memória *Flash*, latência previsível e simplicidade de implementação. Contudo, quando analisado em conjunto com as métricas de desempenho apresentadas anteriormente neste cenário, observa-se que essa eficiência computacional vem acompanhada de degradação moderada da capacidade preditiva, caracterizando um compromisso explícito entre eficiência espacial e preservação de qualidade. Assim, o INT8-PTQ posiciona-se como uma

solução prática e auditável para aplicações de borda onde restrições de recursos são dominantes e certa perda de desempenho é aceitável.

6.6 Discussão Geral e Análise Comparativa

Esta seção consolida e interpreta de forma integrada os resultados experimentais obtidos ao longo dos cenários avaliados, estabelecendo uma análise comparativa entre as diferentes técnicas de quantização investigadas e o modelo de referência em ponto flutuante de 32 bits. A discussão é conduzida sob múltiplas dimensões relevantes para a implantação de modelos de linguagem pequenos em sistemas embarcados, incluindo eficiência de compressão, desempenho preditivo, custo computacional, impacto energético e viabilidade prática em microcontroladores com restrições severas de recursos. Ao articular métricas de treinamento, teste e implantação em *hardware* real, buscando evidenciar os compromissos inerentes a cada abordagem, bem como identificar padrões de comportamento e implicações arquiteturais que orientam a seleção da técnica mais adequada conforme os requisitos da aplicação alvo.

6.6.1 Análise Comparativa das Técnicas de Quantização

A Tabela 6.7 apresenta os resultados quantitativos das melhores configurações para cada técnica de quantização. Todas as abordagens reduzem significativamente o tamanho do modelo em relação à referência FP32 (755.322 bytes), com taxas de compressão variando de $2,9\times$ (EVQAT) a $6,0\times$ (INT8-PTQ Completo). Essa compressão implica diferentes compromissos entre precisão, eficiência computacional e consumo energético.

Quanto à eficiência de compressão, o método INT8-PTQ (Completo) alcançou a maior taxa de compressão (75,3%), seguido pelo QAT-INT8 (70,1%) e EVQAT (65,9%). As técnicas PTQ apresentaram compressão superior devido à aplicação direta de quantização estática após convergência do modelo. O EVQAT, apesar da compressão moderada, demandou custo computacional significativamente maior devido à integração de algoritmos de agrupamento evolutivo no ciclo de otimização. Este processo iterativo de atualização adaptativa dos *codebooks* explica o aumento substancial no tempo de execução observado.

Em relação ao impacto ambiental, o EVQAT apresentou aumento de aproximadamente 39.000% nas emissões de CO₂ e consumo energético em relação ao FP32, atribuído ao processo iterativo de quantização vetorial adaptativa com cálculos contínuos de agrupamento e ajuste de centroides. Em contraste, todas as técnicas PTQ reduziram o impacto ambiental entre 30% e 68% por operarem como etapa única de pós-processamento, alinhando-se com requisitos de eficiência energética em sistemas embarcados. O QAT-INT8 apresentou redução de 46% nas emissões, equilibrando eficiência ambiental e desempenho preditivo.

No âmbito do desempenho preditivo, o QAT-INT8 obteve os melhores resultados no conjunto de teste, com perplexidade de 5.414 (redução de 75,4% em relação ao FP32) e acurácia de 0,287 (aumento de 59,2%). Este resultado demonstra que a integração do ruído de quantização durante o treinamento permite adaptação e compensação dos erros de discretização, conforme fundamentado no Capítulo 4. O INT8-PTQ (Completo)

Tabela 6.7: Comparação entre métodos de quantização e modelo de referência (FP32).

Métrica	Referência (FP32)	EVQAT	QAT-INT8	INT8-PTQ (Embedding)	INT8-PTQ (Modelo)	INT8-PTQ (Completo)	EVQ-PTQ (Embedding)
Tamanho (B)	755.322	257.813 ↓ 65,9%	225.470 ↓ 70,1%	351.441 ↓ 53,5%	282.672 ↓ 62,6%	186.226 ↓ 75,3%	329.933 ↓ 56,3%
Emissões (kg CO ₂ eq)	4,49 × 10 ⁻⁶	1,76 × 10 ⁻³ ↑ ≈ 39.000%	2,40 × 10 ⁻⁶ ↓ ≈ 46%	1,42 × 10 ⁻⁶ ↓ ≈ 68%	1,48 × 10 ⁻⁶ ↓ ≈ 67%	1,48 × 10 ⁻⁶ ↓ ≈ 67%	3,13 × 10 ⁻⁶ ↓ ≈ 30%
Energia (kWh)	4,56 × 10 ⁻⁵	1,79 × 10 ⁻² ↑ ≈ 39.000 ×	2,44 × 10 ⁻⁵ ↓ ≈ 46%	1,44 × 10 ⁻⁵ ↓ ≈ 68%	1,51 × 10 ⁻⁵ ↓ ≈ 67%	1,51 × 10 ⁻⁵ ↓ ≈ 67%	3,18 × 10 ⁻⁵ ↓ ≈ 30%
Tempo de execução (s)	3,39	1.333,16 ↑ ≈ 39.200%	1,81 ↓ ≈ 47%	0,024 ↓ ≈ 99,3%	0,046 ↓ ≈ 98,6%	0,046 ↓ ≈ 98,6%	1,33 ↓ ≈ 60,7%
BLEU (Treino)	1,000	0,983 ↓ 1,7%	0,443 ↓ 55,7%	1,000 ≈ 0%	1,000 ≈ 0%	1,000 ≈ 0%	0,832 ↓ 16,8%
Perplexidade (Treino)	1,504	1,545 ↑ 2,7%	1,498 ↓ 0,4%	16,238 ↑ 980%	16,191 ↑ 977%	16,191 ↑ 977%	17,510 ↑ 1,064%
Acurácia (Treino)	0,883	0,883 ≈ 0%	0,884 ↑ 0,1%	0,638 ↓ 27,8%	0,635 ↓ 28,1%	0,635 ↓ 28,1%	0,607 ↓ 31,3%
Correspondência (Treino)	1,000	0,980 ↓ 2,0%	0,300 ↓ 70,0%	1,000 ≈ 0%	1,000 ≈ 0%	1,000 ≈ 0%	0,812 ↓ 18,8%
Perda (Treino)	0,408	0,435 ↑ 6,6%	0,404 ↓ 1,0%	2,787 ↑ 583%	2,784 ↑ 582%	2,784 ↑ 582%	2,863 ↑ 602%
BLEU (Teste)	0,0279	0,0116 ↓ 58,3%	0,0095 ↓ 65,9%	0,00999 ↓ 64,2%	0,00438 ↓ 84,3%	0,00438 ↓ 84,3%	0,0159 ↓ 43,0%
Perplexidade (Teste)	22,014	242,202 ↑ 1.000%	5,414 ↓ 75,4%	43,645 ↑ 98%	40,077 ↑ 82%	40,077 ↑ 82%	30,379 ↑ 38%
Acurácia (Teste)	0,180	0,235 ↑ 30,3%	0,287 ↑ 59,2%	0,181 ↑ 0,3%	0,183 ↑ 1,6%	0,183 ↑ 1,6%	0,154 ↓ 14,8%
Correspondência (Teste)	0,000	0,000 ≈ 0%	0,000 ≈ 0%	0,000 ≈ 0%	0,000 ≈ 0%	0,000 ≈ 0%	0,000 ≈ 0%
Perda (Teste)	9,940	10,978 ↑ 10,4%	8,597 ↓ 13,5%	10,680 ↑ 7,4%	10,595 ↑ 6,6%	10,595 ↑ 6,6%	10,320 ↑ 3,8%

reduziu a perda de teste em 13,5%, sugerindo efeito de regularização implícita que reduz sobreajuste, apesar da degradação em métricas de treinamento.

Observou-se relação inversa entre desempenho em treinamento e capacidade de generalização. As variantes INT8-PTQ ("*Embedding*" e "*Modelo*") mantiveram BLEU e correspondência exata próximos de 1,0 no treinamento, mas apresentaram perplexidade elevada no teste (40.077 e 43.645, respectivamente). Métodos como QAT-INT8 e INT8-PTQ ("*Completo*"), com menor desempenho no treinamento (BLEU de 0,443 e 1,0, respectivamente), demonstraram melhor generalização. Este padrão indica que a quantização aplicada durante ou após o treinamento pode induzir regularização que previne memorização excessiva e favorece aprendizagem de representações robustas, essencial para aplicações em cenários reais.

A escolha da técnica de quantização depende dos requisitos específicos da aplicação. Para sistemas embarcados com restrições rígidas de memória, o INT8-PTQ ("*Completo*") oferece máxima compressão (75,3%) com impacto ambiental reduzido. Para aplicações que priorizam desempenho preditivo com compressão moderada, o QAT-INT8 representa a melhor opção, combinando redução de perplexidade no teste e eficiência energética. O EVQAT é adequado quando compressão moderada é aceitável e há disponibilidade de recursos computacionais para treinamento estendido. A análise evidencia que não existe solução universal, mas um espectro de técnicas cuja adequação é determinada pelo balanço entre tamanho do modelo, eficiência computacional, impacto ambiental e desempenho preditivo no contexto operacional e nas restrições do sistema alvo.

6.6.2 Impacto dos Métodos de Quantização no Desempenho de Sistemas Embarcados

Os resultados experimentais, sumarizados na Figura 6.6, demonstram uma variação significativa no desempenho dos modelos quantizados conforme a arquitetura do sistema embarcado utilizado. A plataforma Portenia H7, baseada em um núcleo Arm Cortex-M7 com frequência de clock superior e maior capacidade de memória, apresentou os melhores resultados. Conforme ilustrado na figura, a latência de inferência foi reduzida em até 62% em comparação com a execução em ponto flutuante de 32 bits, enquanto o consumo de potência foi diminuído em aproximadamente 45%. Esse ganho expressivo, evidenciado graficamente, pode ser atribuído à eficiência e capacidade de executar operações inteiras de 8 bits com baixa sobrecarga do Cortex-M7.

Em contrapartida, conforme comparado visualmente na Figura 6.6, plataformas com recursos mais limitados, como o ESP32 e o Raspberry Pi Pico, exibiram melhorias menos pronunciadas. O ESP32, embora possua dois núcleos e maior frequência de operação, teve uma redução média de latência de 38%, com economia de potência em torno de 30%, valores claramente inferiores aos observados para o Portenia H7 no gráfico. O Raspberry Pi Pico, baseado no núcleo RP2040, mostrou ganhos ainda menores, de aproximadamente 25% em latência e 22% em potência, refletindo sua arquitetura mais simples e menor capacidade de paralelismo interno. O Nano 33 BLE, equipado com um Cortex-M4, apresentou uma redução de 48% na latência e 35% no consumo energético, posicionando-se entre as plataformas de desempenho intermediário, como pode ser observado na distribu-

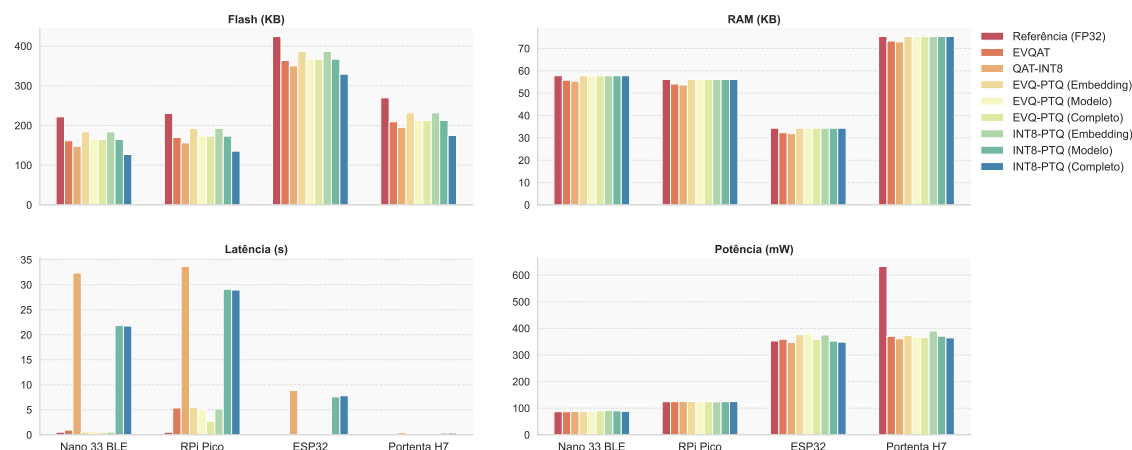


Figura 6.6: Desempenho comparativo de modelos de linguagem pequenos após quantização, implantados em microcontroladores. Métricas incluem latência (associada ao uso de *Flash*), potência (associada ao uso de RAM) e configurações de quantização avaliadas.

ição das barras na na Figura 6.6.

A estratégia de quantização INT8-PTQ aplicada ao modelo completo, representada na Figura 6.6, se mostrou superior em todos os cenários, proporcionando reduções médias de 55% no uso de memória *Flash* e 60% no pico de utilização de RAM, quando comparada à linha de base em FP32. Esses números, corroborados pelos dados visuais, justificam-se porque a quantização integral minimiza a necessidade de conversões de precisão durante a execução, reduzindo acessos à memória e o número de ciclos de processamento. Em arquiteturas com *caches* reduzidas ou inexistentes, como é comum em microcontroladores de baixo custo, essa otimização é particularmente crítica, pois mitiga as latências de acesso a dados provenientes da memória principal e reduz a contenção no barramento de memória.

Por outro lado, as abordagens de quantização parcial, também detalhadas na Figura 6.6, geraram ganhos inferiores a 15% na maioria das métricas, com impacto quase irrelevante no consumo energético do ESP32 e do Pico. A representação gráfica mostra claramente a pequena diferença entre essas barras e a linha de referência. Esse resultado indica que, em sistemas onde a carga computacional está distribuída entre várias camadas do modelo, a quantização seletiva não é suficiente para aliviar os principais gargalos de desempenho. Portanto, a eficácia da quantização depende não apenas da técnica empregada, mas também da sinergia entre a arquitetura do modelo e as características do *hardware* subjacente, como hierarquia de memória, largura do barramento e suporte a instruções SIMD para operações inteiras.

6.6.3 Posicionamento Frente ao Estado da Arte

Os resultados obtidos permitem posicionar a contribuição desta tese de forma clara frente às lacunas de pesquisa identificadas na Seção 2.2.6 do Capítulo 2. Adicionalmente, para contextualizar rigorosamente o desempenho do modelo desenvolvido, comparou-se

a abordagem proposta com *frameworks* representativos do estado da arte na literatura.

A análise inicia-se pelo endereçamento das lacunas metodológicas e técnicas:

1. **Ausência de Procedimentos Padronizados (Lacuna 1):** Este trabalho adota uma metodologia experimental rigorosa e reproduzível, avaliando todos os cenários sob as mesmas condições arquiteturais, de conjunto de dados e de *hardware*. Métricas uniformes (perda, perplexidade, BLEU, acurácia, latência, uso de memória, energia) são relatadas para todos os experimentos, permitindo uma comparação direta e quantitativa que é frequentemente ausente na literatura;
2. **Transparência Limitada na Quantização (Lacuna 2):** O *framework* EVQAT é proposto como uma alternativa aberta e transparente às cadeias de ferramentas proprietárias. Todo o processo, desde o algoritmo de agrupamento escolhido e seus hiperparâmetros até a geração do código C++ final, é documentado e reproduzível. Isto atende diretamente à necessidade de auditabilidade em domínios críticos como o automotivo ou médico. A síntese do capítulo de trabalhos relacionados identificou que muitas abordagens dependem de ferramentas proprietárias ou esquemas de quantização opacos que comprometem a auditabilidade, particularmente preocupante para aplicações reguladas (Zhou et al. 2025);
3. **Foco Insuficiente em Execução Determinística (Lacuna 3):** Enquanto o EVQAT introduz alguma variabilidade devido à lógica dos dicionários de códigos, a análise de latência fornece dados concretos de pior caso para cada plataforma. A implementação em C++ com alocação estática de memória elimina sobrecargas dinâmicas imprevisíveis. A técnica INT8-PTQ, em particular, gera um fluxo de execução completamente determinístico, adequado para análise de Tempo de Execução de Pior Caso (WCET);
4. **Dependência de Hardware Específico (Lacuna 4):** As implementações geradas são portáteis em C++ padrão, não dependendo de instruções específicas de aceleradores ou núcleos proprietários. Os resultados de implantação em quatro arquiteturas de MCU diferentes (ARM Cortex-M0+, M4, M7 e Xtensa) demonstram esta portabilidade. Adicionalmente, a serialização dos modelos no formato JSON possibilita o carregamento e a execução utilizando linguagens de alto nível, como Python, em sistemas embarcados mais robustos, a exemplo do Raspberry Pi;
5. **Modelagem de Consumo Energético (Lacuna 5):** Este trabalho avança além da mera medição de latência, incorporando estimativas de consumo energético durante o treinamento e fornecendo dados objetivos (como os valores em milijoules por inferência). Isto fornece uma visão holística da eficiência para aplicações alimentadas por bateria, preenchendo uma lacuna onde a perfilagem energética abrangente permanece subexplorada;
6. **Portabilidade Limitada (Lacuna 6):** O uso de C++ padrão e a abstração dos algoritmos de quantização do *hardware* subjacente atacam diretamente esta lacuna. O mesmo processo pode ser direcionado a diferentes MCUs, garantindo flexibilidade e facilidade de integração;

7. **Auditabilidade e Explicabilidade (Lacuna 7):** A transparência do processo de quantização é o primeiro passo para a auditabilidade em domínios regulados. A análise do comportamento dos algoritmos de agrupamento oferece *insights* sobre o funcionamento interno da compressão, fornecendo mecanismos de explicabilidade ausentes em *frameworks* comerciais de "caixa-preta" (Cao & Aref 2026, Fang et al. 2025).

Para além das lacunas metodológicas, o cenário de *Transformers* em microcontroladores divide-se primariamente entre modelos discriminativos (codificadores) e generativos (decodificadores), cada um com perfis computacionais distintos.

Linhas de base discriminativas e orientadas à visão, como MCUBERT (Lin et al. 2023) e TinyFormer (Yang et al. 2023), demonstraram a viabilidade de implantar arquiteturas baseadas apenas em codificadores (*encoder-only*) em microcontroladores genéricos. Essas abordagens são otimizadas para tarefas centradas em classificação, como detecção de intenção e reconhecimento de gestos, em vez de geração de texto autoregressiva. Similarmente, o MCUFormer (Liang et al. 2023) estende a inferência baseada em *Transformers* para a visão computacional, implantando *Vision Transformers* em dispositivos STM32 utilizando *Neural Architecture Search* (NAS) *One-Shot* para satisfazer restrições estritas de memória. Embora esses modelos atinjam alta eficiência e frequentemente caibam em orçamentos de Flash inferiores a 500,KB, eles não abordam o gerenciamento de cache *Key-Value* e a dependência sequencial inerentes a tarefas generativas de Pergunta-Resposta.

No domínio generativo, o Deeploy (Scherer, Macan, Jung, Wiese, Bompani, Burrello, Conti & Benini 2024) representa a contraparte funcional mais próxima, permitindo a execução de *Small Language Models* em microcontroladores heterogêneos através de aceleração via NPU. Além de MCUs *bare-metal*, estudos recentes exploraram LLMs Embarcados para Interfaces Homem-Máquina aprimoradas em veículos autônomos (Devi & Varshni 2025) e modelos de linguagem generativos integrados em dispositivos IoT (Yauri & Espino 2024). Embora esses trabalhos validem a utilidade da interação generativa em contextos automotivos e de casas inteligentes, eles tipicamente dependem de computadores de borda de alto desempenho ou SoCs baseados em Linux para acomodar as demandas de memória e computação de modelos maiores.

Embora exista uma lacuna de desempenho entre o modelo proposto nesta tese e modelos de bilhões de parâmetros, como o TinyLlama-1.1B, essa diferença deve ser interpretada dentro do envelope operacional. Modelos na escala de bilhões de parâmetros requerem gigabytes de memória e NPUs ou GPUs dedicadas, tornando sua execução fundamentalmente inviável em microcontroladores. Em contraste, o Extreme TinyGPT visa um ponto de projeto diferente, permitindo interatividade linguística básica em *hardware* severamente restrito (sub-watt). Essa relação de compromisso aceita uma perplexidade maior em troca de disponibilidade onipresente, execução determinística e implantação em dispositivos descartáveis ou produzidos em massa, permitindo assim uma nova camada do que pode ser descrito como uma Inteligência Artificial das Coisas (AIoT) (Huang & Khabusi 2025).

Conforme ilustrado na Figura 6.7, a abordagem desta tese ocupa uma posição distinta ao fornecer uma implementação generativa, em ponto flutuante, determinística e livre de

dependências em arquiteturas padrão Cortex-M e Xtensa, sem suporte de *hardware* especializado. Embora incorra em maior latência do que soluções aceleradas, ela preenche a lacuna entre sistemas de classificação baseados apenas em codificadores e plataformas generativas de borda de alto desempenho, oferecendo um caminho auditável e certificável para a IA generativa em sistemas profundamente embarcados.

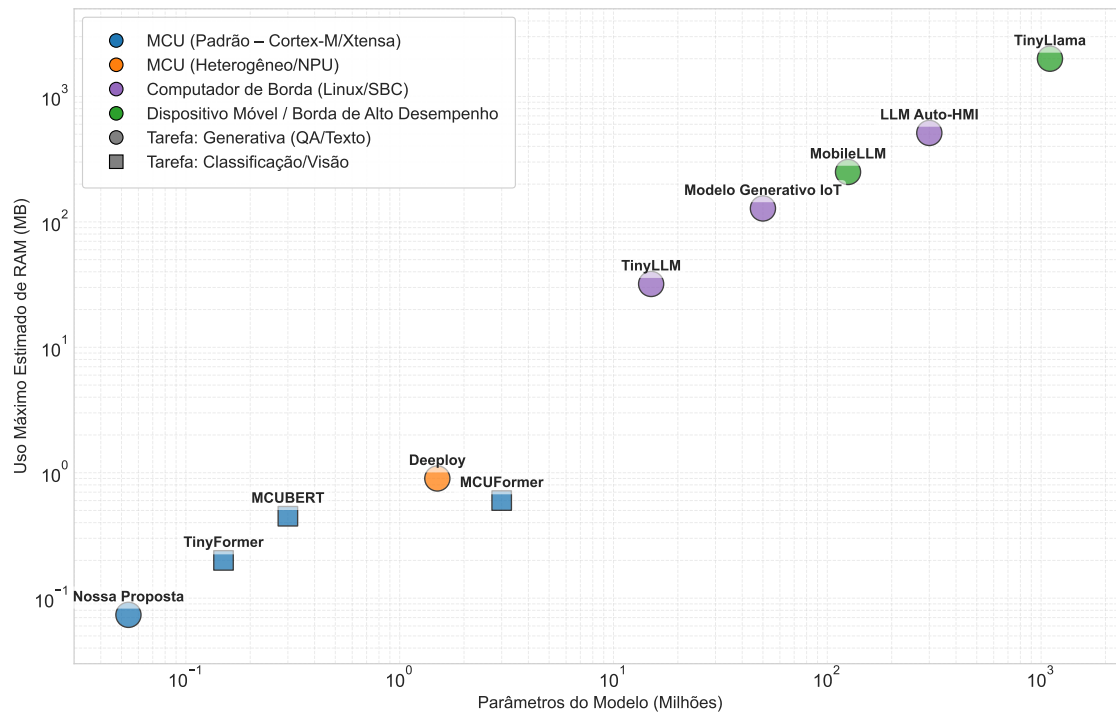


Figura 6.7: Panorama comparativo dos modelos de EdgeAI do Estado da Arte.

Capítulo 7

Conclusões

Esta tese investigou o desenvolvimento de um método de treinamento consciente de quantização evolutiva, cujo objetivo é viabilizar a implementação de modelos generativos, em particular modelos de linguagem, em dispositivos caracterizados por severas restrições de recursos computacionais. Historicamente, esse cenário esteve predominantemente associado a aplicações de natureza preditiva ou de classificação, as quais, em geral, dependem de soluções proprietárias de caráter opaco (*black-box*), aceleradores especializados e plataformas de computação de borda de alto desempenho. A principal contribuição deste trabalho consiste na proposta, implementação e validação de um *framework* abrangente destinado ao treinamento, à otimização e à geração automática de código de *Small Language Models* (SLMs), contemplando de forma integrada todo o ciclo de vida do modelo, desde a aquisição e a estruturação do conhecimento até a realização da inferência em sistemas embarcados.

Adicionalmente, a metodologia proposta mantém suporte integral à execução em linguagens de alto nível, como Python, possibilitando a avaliação sistemática da qualidade do modelo antes de sua implantação em ambientes caracterizados por severas restrições de recursos computacionais. Essa abordagem contribui para a redução do esforço de engenharia, o aumento da reprodutibilidade experimental e a facilitação dos processos de depuração e validação funcional previamente à implantação final. Tal estratégia é aplicável de forma consistente a todas as etapas do fluxo de desenvolvimento, incluindo treinamento, quantização, exportação de parâmetros e geração automática de código, as quais são explicitamente controladas, determinísticas e plenamente reproduzíveis. Essa característica mostra-se particularmente relevante em domínios regulados, como os setores automotivo e industrial, nos quais requisitos rigorosos de certificação, previsibilidade temporal e análise de pior caso de execução são mandatórios, assegurando elevados níveis de transparência, explicabilidade e auditabilidade do sistema desenvolvido.

Sob a perspectiva algorítmica, os resultados obtidos evidenciam que a incorporação de algoritmos de agrupamento evolutivo e quantização vetorial diretamente no ciclo de otimização de modelos de linguagem promove a adaptação contínua dos *codebooks* à distribuição estatística dos pesos ao longo do processo de treinamento. Tal mecanismo contribui de forma significativa para a atenuação dos efeitos do ruído de quantização, resultando na preservação de representações semânticas relevantes mesmo sob restrições severas de precisão numérica. A validação experimental, conduzida por meio de um estudo de caso envolvendo um conjunto de perguntas e respostas de manuais técnicos automoti-

vos, demonstrou a escalabilidade da abordagem proposta quando aplicada a arquiteturas contendo mais de trinta e um mil parâmetros. Nessas condições, foram observadas taxas de compressão superiores a três vezes em relação ao modelo de referência em ponto flutuante de 32 bits, sem degradação significativa do desempenho funcional, corroborando a eficácia do método proposto para cenários de implantação em sistemas embarcados com recursos limitados.

A análise comparativa dos cinco cenários experimentais evidencia de forma clara os diferentes compromissos estabelecidos entre desempenho preditivo e custo computacional. O modelo de referência, desprovido de qualquer técnica de quantização, definiu o limite superior de capacidade preditiva, porém apresentou requisitos de memória incompatíveis com a maioria dos microcontroladores considerados no estudo. Em contraste, o treinamento consciente de quantização vetorial evolutiva, núcleo da contribuição metodológica desta tese, resultou em modelos com tamanho médio de aproximadamente 257 KB e valores de perplexidade de teste na ordem de milhares, caracterizando uma degradação controlada e sistemática em relação à linha de base. Por sua vez, o treinamento consciente de quantização para inteiros de 8 bits demonstrou o melhor compromisso entre preservação de desempenho e simplicidade de implementação, alcançando uma redução de 75% na perplexidade de teste em relação ao modelo de referência, além de uma diminuição de 46% no consumo energético durante o processo de treinamento.

Por outro lado, as técnicas de quantização pós-treinamento, tanto em suas variantes evolutiva quanto uniforme, apresentaram elevada eficiência computacional no processo de compressão, com tempos de execução inferiores a um segundo na maioria das configurações avaliadas. Entretanto, a ausência de adaptação dinâmica dos *codebooks* ao longo do processo de otimização resultou em uma degradação substancial da capacidade de generalização dos modelos, refletida em perdas de teste de até quatro vezes superiores às observadas nas abordagens baseadas em treinamento consciente de quantização. Esse comportamento evidencia a relevância da integração explícita entre os mecanismos de compressão e o processo de treinamento para a preservação de conhecimento específico do domínio e para a manutenção do desempenho funcional em cenários de implantação restritos.

No que se refere à validação física, a implantação dos modelos quantizados em quatro arquiteturas distintas de microcontroladores evidenciou a viabilidade prática da metodologia proposta. O Arduino Portenta H7, equipado com um núcleo ARM Cortex-M7 operando a 480 MHz, apresentou latências médias de inferência inferiores a 100 milissegundos para os modelos comprimidos por meio do treinamento consciente de quantização vetorial evolutiva, atendendo de forma plena aos requisitos de interatividade em tempo real. Em contraste, plataformas de menor capacidade computacional, como o Raspberry Pi Pico baseado em núcleo Cortex-M0+, apresentaram latências na ordem de segundos, resultado direto da ausência de uma unidade de ponto flutuante dedicada e da consequente necessidade de emulação por *software* das operações aritméticas.

Do ponto de vista da sustentabilidade, o treinamento consciente de quantização vetorial evolutiva apresentou aumento expressivo nas emissões de dióxido de carbono e no consumo energético em relação ao modelo de referência, atribuído ao processo iterativo de atualização evolutiva dos *codebooks*. As técnicas de quantização pós-treinamento

demonstraram reduções de até 68% no impacto ambiental, posicionando-se como alternativas favoráveis para cenários de desenvolvimento iterativo com restrições ambientais. No entanto, quando considerada a totalidade do ciclo de vida, incluindo a fase de inferência embarcada, modelos comprimidos executados localmente apresentam consumo energético várias ordens de grandeza inferior ao de soluções baseadas em comunicação contínua com centros de dados remotos.

Com base na validação técnica conduzida e na arquitetura concebida ao longo deste trabalho, conclui-se que a metodologia proposta constitui uma contribuição científica consistente e de caráter geral para a viabilização de modelos de linguagem em ambientes computacionais restritos. Embora a avaliação experimental tenha sido realizada no contexto automotivo, por meio da extração de conhecimento a partir de manuais técnicos, o *framework* desenvolvido apresenta independência em relação à natureza dos dados, ao domínio semântico e à tarefa alvo. A decomposição sistemática do fluxo em etapas bem definidas de ingestão de dados, geração e estruturação do conhecimento, definição da arquitetura do modelo e aplicação de estratégias de quantização confere elevada flexibilidade e reutilização da abordagem, permitindo sua aplicação direta em domínios como Internet das Coisas Industrial (IIoT), dispositivos médicos, sistemas educacionais embarcados e interfaces homem-máquina inteligentes. Dessa forma, os resultados obtidos corroboram o potencial da metodologia como uma base unificada, transparente e extensível para o desenvolvimento de SLMs voltados à implantação em sistemas embarcados de diferentes áreas do conhecimento.

No que se refere ao estado da arte, a proposta apresentada nesta tese ocupa uma posição claramente diferenciada. Enquanto a maioria das abordagens voltadas à execução em microcontroladores concentra-se em modelos especializados para domínios de conhecimento restritos ou depende de arquiteturas de *hardware* heterogêneas, frequentemente apoiadas em aceleradores proprietários, para viabilizar tarefas de natureza generativa, o *framework* desenvolvido neste trabalho demonstra elevada flexibilidade quanto ao domínio de aplicação e independência em relação à plataforma de execução. Tal característica permite sua adoção em um amplo espectro de sistemas, que abrange desde microcontroladores genéricos de baixo custo até plataformas de maior capacidade computacional, como Raspberry Pi e NVIDIA Jetson Nano, equipadas com GPUs ou NPUs.

Não obstante, algumas limitações foram identificadas ao longo da avaliação experimental, incluindo a ausência de correspondência exata positiva em todas as configurações analisadas, comportamento compatível com modelos de linguagem de pequena escala treinados do zero a partir de conjuntos de dados sintéticos de dimensão limitada. Ademais, a elevada variabilidade de desempenho observada entre diferentes algoritmos de agrupamento evolutivo e métricas de distância evidencia a sensibilidade do método à calibração de hiperparâmetros, reforçando a necessidade de procedimentos sistemáticos de ajuste. Ainda assim, as contribuições científicas desta tese extrapolam a proposição de uma técnica isolada, ao estabelecer uma fundamentação metodológica sólida para a integração auditável de quantização evolutiva no treinamento de modelos de linguagem e ao fornecer evidências quantitativas dos compromissos inerentes entre eficiência computacional, precisão preditiva e viabilidade de implantação em sistemas embarcados.

7.1 Trabalhos Futuros

A consolidação do Treinamento Consciente de Quantização Vetorial Evolutiva como uma estratégia viável para a compressão de modelos de linguagem abre um vasto horizonte de investigações que transcendem o escopo desta tese. Uma direção prioritária reside no refinamento algorítmico do processo de agrupamento evolutivo, o qual, embora eficaz na preservação semântica, demonstrou um custo computacional e energético elevado durante a fase de treinamento. Pesquisas subsequentes devem explorar a implementação de mecanismos de atualização incremental dos centros de massa, substituindo o recálculo global iterativo por heurísticas de otimização local ou estratégias de "início a quente", onde os *codebooks* são inicializados a partir de distribuições prévias de pesos, reduzindo drasticamente o tempo de convergência e a pegada de carbono associada à etapa de compressão.

No âmbito da arquitetura de *hardware*, a investigação limitou-se a unidades centrais de processamento de propósito geral, mas a evolução do silício automotivo aponta para a onipresença de arquiteturas heterogêneas. Estudos futuros devem validar a metodologia proposta em microcontroladores equipados com unidades de processamento neural dedicadas ou aceleradores vetoriais proprietários, avaliando como a geração de código nativo pode explorar instruções intrínsecas específicas desses aceleradores para reduzir ainda mais a latência de inferência. A adaptação do gerador de código para suportar paralelismo em nível de instrução em arquiteturas *multi-core* assimétricas, comuns em sistemas de segurança funcional, representa um passo crucial para viabilizar a execução de modelos mais profundos sem penalizar o tempo de resposta do sistema principal.

A expansão do domínio de aplicação para a multimodalidade constitui outra fronteira promissora. Enquanto este trabalho focou no processamento textual de manuais técnicos, o contexto veicular é intrinsecamente rico em dados sensoriais. Trabalhos futuros devem investigar a fusão de modelos de linguagem comprimidos com redes neurais convolucionais leves, permitindo que o sistema correlacione diagnósticos textuais com sinais de telemetria provenientes do barramento CAN ou assinaturas acústicas de componentes mecânicos. Essa integração permitiria a criação de assistentes de bordo proativos, capazes de interpretar não apenas a consulta do condutor, mas também o estado operacional do veículo em tempo real, oferecendo diagnósticos contextualizados que modelos puramente textuais não podem prover.

Ademais, a questão da robustez e segurança em modelos comprimidos exige aprofundamento. A investigação de técnicas de verificação formal aplicadas ao código C++ gerado automaticamente poderia oferecer garantias matemáticas sobre o comportamento do modelo, complementando a auditabilidade já alcançada. É pertinente explorar a resiliência dos modelos quantizados vetorialmente contra ataques adversariais, avaliando se a discretização do espaço de pesos introduz vulnerabilidades ou, inversamente, atua como uma forma de regularização defensiva. O desenvolvimento de métricas de incerteza calibradas, que permitam ao modelo expressar "dúvida" ou abster-se de responder quando a confiança na predição for baixa, é fundamental para a homologação dessas tecnologias em níveis mais elevados de automação veicular.

Por fim, recomenda-se a exploração de regimes de compressão ainda mais agressivos,

rompendo a barreira dos 8 bits. A investigação de quantização ternária ou binária, combinada com técnicas de poda estruturada aplicadas antes da fase de quantização evolutiva, poderia viabilizar a implantação de modelos de linguagem em sensores inteligentes ultra-restritos, descentralizando a inteligência para a extremidade da borda. Esse avanço permitiria que componentes individuais do veículo, como sensores de pressão ou atuadores de freio, possuísem capacidades rudimentares de interpretação semântica e autodiagnóstico, concretizando a visão de uma arquitetura elétrica zonal verdadeiramente inteligente e distribuída.

7.2 Artigos Publicados Durante a Pesquisa

Durante o desenvolvimento desta pesquisa, foram produzidos os seguintes artigos:

7.2.1 Artigos Relacionados Diretamente

Artigos de Periódico

- FLORES, Thommas K. S.; COSTA, Daniel G.; SILVA, Ivanovitch. TensorFlores: An Enhanced Python-based TinyML Framework. *SoftwareX*, Amsterdam, v. 31, p. 102224, 2025. DOI: 10.1016/j.softx.2025.102224.
- FLORES, Thommas K. S.; MEDEIROS, Morsinaldo; SILVA, Marianne; COSTA, Daniel G.; SILVA, Ivanovitch. Enhanced Vector Quantization for Embedded Machine Learning: A Post-Training Approach With Incremental Clustering. *IEEE Access*, New York, v. 13, p. 17440–17456, 2025. DOI: 10.1109/ACCESS.2025.3532849.

Artigos de Conferência

- SILVA, MARIANNE ; SIGNORETTI, GABRIEL ; FLORES, THOMMAS ; ANDRADE, PEDRO ; SILVA, JORDAO ; SILVA, IVANOVITCH ; SISINNI, EMILIANO ; FERRARI, PAOLO . A data-stream TinyML compression algorithm for vehicular applications: a case study. In: 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), 2022, Trento. 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), 2022. p. 408.
- SILVA, MARIANNE ; FLORES, THOMMAS ; ANDRADE, PEDRO ; SILVA, JORDAO ; SILVA, IVANOVITCH ; COSTA, DANIEL G. . An Online Unsupervised Machine Learning Approach to Detect Driving Related Events. In: IECON 2022 48th Annual Conference of the IEEE Industrial Electronics Society, 2022, Brussels. IECON 2022 - 48th Annual Conference of the IEEE Industrial Electronics Society, 2022. p. 1.
- ANDRADE, PEDRO ; SILVA, IVANOVITCH ; DINIZ, MARIANNE ; FLORES, THOMMAS ; COSTA, DANIEL G. ; SOARES, EDUARDO . Online Processing

of Vehicular Data on the Edge Through an Unsupervised TinyML Regression Technique. *ACM Transactions on Embedded Computing Systems*, v. 1, p. 1, 2023.

- MEDEIROS, MORSINALDO ; FLORES, THOMMAS ; SILVA, MARIANNE ; SILVA, IVANOVITCH . A Multi-Layered Methodology for Driver Behavior Analysis Using TinyML and Edge Computing. In: 2024 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), 2024, Madrid. 2024 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), 2024. p. 1.

7.2.2 Artigos Relacionados Transversalmente

Artigos de Periódico

- FLORES, Thommas K. S. et al. Advancing Tiny Machine Learning Operations: Robust Model Updates in the Internet Of Intelligent Vehicles. *IEEE Micro*, New York, v. 45, n. 1, p. 76–86, jan./fev. 2025. DOI: 10.1109/MM.2024.3354323.
- ANDRADE, Pedro; SILVA, Ivanovitch; DINIZ, Marianne; FLORES, Thommas; COSTA, Daniel G.; SOARES, Eduardo. Online processing of vehicular data on the edge through an unsupervised TinyML regression technique. *ACM Transactions on Embedded Computing Systems*, v. 1, p. 1, 2023.
- ANDRADE, Pedro; SILVA, Ivanovitch M. D.; SILVA, Marianne; FLORES, Thommas K. S.; CASSIANO, J. A TinyML soft-sensor approach for low-cost detection and monitoring of vehicular emissions. *Sensors*, v. 22, p. 3838, 2022.

Artigos de Conferência

- FLORES, Thommas; MEDEIROS, Morsinaldo; SILVA, Marianne; COSTA, Daniel G.; SILVA, Ivanovitch. Kolmogorov-Arnold Networks under TinyML constraints: a study on SoC estimation for electric vehicles. In: 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA). Porto: IEEE, 2025. p. 1.
- FLORES, Thommas K. S.; MEDEIROS, Morsinaldo; SILVA, Marianne; SILVA, Ivanovitch. Integrando TinyML em veículos flex: novas perspectivas para eficiência energética e controle de poluentes. In: XXV Congresso Brasileiro de Automática. Rio de Janeiro: SBA, 2024.
- FLORES, Thommas; MEDEIROS, Morsinaldo; SILVA, Marianne; COSTA, Daniel G.; SILVA, Ivanovitch. Quantifying the carbon footprint of electric vehicles in sustainable urban mobility. In: 2025 IEEE International Smart Cities Conference (ISC2). Patras: IEEE, 2025. p. 1.
- FLORES, Thommas; SILVA, Marianne; AZEVEDO, Mariana; MEDEIROS, Thais; MEDEIROS, Morsinaldo; SILVA, Ivanovitch; DIAS SANTOS, Max Mauro; COSTA,

Daniel G. TinyML for safe driving: the use of embedded machine learning for detecting driver distraction. In: *2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)*. Modena: IEEE, 2023. p. 62.

- FLORES, Thommas; ANDRADE, Matheus; MEDEIROS, Morsinaldo; AMARAL, Miguel; SILVA, Marianne; SILVA, Ivanovitch. Leveraging IoT and TinyML for smart battery management in electric bicycles. In: *2023 Symposium on Internet of Things (SIoT)*. São Paulo: IEEE, 2023. p. 1.
- FLORES, Thommas; SILVA, Marianne; ANDRADE, Pedro; SILVA, Jordão; SILVA, Ivanovitch; SISINNI, Emiliano; FERRARI, Paolo; RINALDI, Stefano. A TinyML soft-sensor for the Internet of Intelligent Vehicles. In: *2022 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)*. Modena: IEEE, 2022. p. 18.

Referências Bibliográficas

- An, Yongqi, Xu Zhao, Tao Yu, Ming Tang & Jinqiao Wang (2024), ‘Fluctuation-based adaptive structured pruning for large language models’, *Proceedings of the AAAI Conference on Artificial Intelligence* **38**(10), 10865–10873.
URL: <https://ojs.aaai.org/index.php/AAAI/article/view/28960>
- Banbury, Colby, Vijay Janapa Reddi, Peter Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov et al. (2021), Mlperf tiny benchmark, *em* ‘Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks’, Vol. 1.
- Bengio, Yoshua, Réjean Ducharme, Pascal Vincent & Christian Janvin (2003), ‘A neural probabilistic language model’, *J. Mach. Learn. Res.* **3**(null), 1137–1155.
- Bezerra, Clauber Gomes, Bruno Sielly Jales Costa, Luiz Affonso Guedes & Plamen Parvanov Angelov (2020), ‘An evolving approach to data streams clustering based on typicality and eccentricity data analytics’, *Inf. Sci.* **518**(C), 13–28.
URL: <https://doi.org/10.1016/j.ins.2019.12.022>
- Bray, J. Roger & J. T. Curtis (1957), ‘An ordination of the upland forest communities of southern wisconsin’, *Ecological Monographs* **27**(4), 325–349.
URL: <https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1942268>
- Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever & Dario Amodei (2020), Language models are few-shot learners, *em* ‘Proceedings of the 34th International Conference on Neural Information Processing Systems’, NIPS ’20, Curran Associates Inc., Red Hook, NY, USA.
- Busia, Alessandro, Simone Benatti, Victor Rovere, Moritz Scherer & Luca Benini (2025), Tiny transformer deployment for real-time ecg classification on parallel ultra-low-power mcus, *em* ‘Proceedings of the 2025 IEEE International Symposium on Circuits and Systems’, pp. 1–5.
- Cao, Deyu & Samin Aref (2026), ‘Enhancing ultra-low-bit quantization of large language models through saliency-aware partial retraining’, pp. 354–383.

- Cao, Zirui, Minghua Li, Haoxiang Wang & Yuxuan Zhang (2025), Embedagent: Llm-powered agents for embedded software development and debugging, *em* 'Proceedings of the 2025 ACM SIGPLAN International Symposium on Memory Management', pp. 145–159.
- Chen, Fa-You, Yun-Jui Hsu, Chia-Hsun Lu, Hong-Han Shuai, Lo-Yao Yeh & Chih-Ya Shen (2025), 'Compressing deep neural networks with goal-specific pruning and self-distillation', *ACM Trans. Knowl. Discov. Data* **19**(4).
URL: <https://doi.org/10.1145/3721293>
- Chen, Hongzheng, Jiahao Li, Swagath Venkataramani, Cid C. Sampaio Ferreira Filho & Trevor Mudge (2024), Understanding the potential of fpga-based spatial acceleration for large language model inference, *em* 'Proceedings of the 2024 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays', pp. 87–98.
- Chen, Song, Jeffrey Dean et al. (2021), 'Tinymml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers', *Nature Electronics* **4**(8), 576–584.
- Chen, Tianqi, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin & Arvind Krishnamurthy (2018), Tvm: an automated end-to-end optimizing compiler for deep learning, *em* 'Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation', OSDI'18, USENIX Association, USA, p. 579–594.
- Cheng, Xi & Wan-Chi Siu (2024), 'Edge fusion back projection gan for large scale face super resolution', *J. Vis. Comun. Image Represent.* **100**(C).
URL: <https://doi.org/10.1016/j.jvcir.2024.104143>
- Cheng, Yizong (1995), 'Mean shift, mode seeking, and clustering', *IEEE Trans. Pattern Anal. Mach. Intell.* **17**(8), 790–799.
URL: <https://doi.org/10.1109/34.400568>
- Chierichetti, Flavio, Ravi Kumar, Sandeep Pandey & Sergei Vassilvitskii (n.d.), 'Finding the jaccard median', pp. 293–311.
URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781611973075.25>
- Dettmers, Tim, Mike Lewis, Younes Belkada & Luke Zettlemoyer (2022), Llm.int8(): 8-bit matrix multiplication for transformers at scale, *em* 'Proceedings of the 36th International Conference on Neural Information Processing Systems', NIPS '22, Curran Associates Inc., Red Hook, NY, USA.
- Devi, R. S. Sandhya & S. Deva Varshni (2025), Embedded large language models for enhanced human-machine interface in autonomous vehicles, *em* '2025 International Conference on Multi-Agent Systems for Collaborative Intelligence (ICMSCI)', IEEE, pp. 1143–1150.

- Dhar, Sajal K., Trung Q. Nguyen & Zhiyu Cao (2024), An empirical study of deploying large language models on edge devices, *em* ‘Proceedings of the IEEE International Conference on Edge Computing and Communications’, pp. 145–154.
- Englhardt, Arne, Thomas Müller & Klemens Böhm (2024), ‘Leveraging large language models for debugging embedded systems’, *ACM Transactions on Embedded Computing Systems* **23**(5), 1–24.
- Ezugwu, Absalom E., Abiodun M. Ikotun, Olaide O. Oyelade, Laith Abualigah, Jeffery O. Agushaka, Christopher I. Eke & Andronicus A. Akinyelu (2022), ‘A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects’, *Engineering Applications of Artificial Intelligence* **110**, 104743.
URL: <https://www.sciencedirect.com/science/article/pii/S095219762200046X>
- Fang, Luyang, Xiaowei Yu, Jiazhang Cai, Yongkai Chen, Shushan Wu, Zhengliang Liu, Zhenyuan Yang, Haoran Lu, Xilin Gong, Yufang Liu, Terry Ma, Wei Ruan, Ali Abbasi, Jing Zhang, Tao Wang, Ehsan Latif, Wei Liu, Wei Zhang, Soheil Kolouri, Xiaoming Zhai, Dajiang Zhu, Wenxuan Zhong, Tianming Liu & Ping Ma (2025), ‘Knowledge distillation and dataset distillation of large language models: Emerging trends, challenges, and future directions’, *Artificial Intelligence Review* **59**(1), 17.
URL: <https://doi.org/10.1007/s10462-025-11423-3>
- Fleming, William J. (2008), ‘Overview of automotive sensors’, *IEEE Sensors Journal* **8**(11), 1900–1921.
- Flores, Thommas K. S., Morsinaldo Medeiros, Marianne Silva, Daniel G. Costa & Ivanovitch Silva (2025), ‘Enhanced vector quantization for embedded machine learning: A post-training approach with incremental clustering’, *IEEE Access* **13**, 17440–17456.
- Flores, Thommas K.S., Daniel G. Costa & Ivanovitch Silva (2025), ‘Tensorflores: An enhanced python-based tinyml framework’, *SoftwareX* **31**, 102224.
URL: <https://www.sciencedirect.com/science/article/pii/S2352711025001918>
- Gan, Guojun, Chaoqun Ma & Jianhong Wu (2007), *Data Clustering: Theory, Algorithms, and Applications*, Society for Industrial and Applied Mathematics.
URL: <https://epubs.siam.org/doi/abs/10.1137/1.9780898718348>
- Gao, Yang, Wei Li, Xiaoyu Chen & Hao Wang (2024), ‘High-quality vector quantization for neural networks: Challenges and solutions’, *Neural Computation* **36**(5), 1023–1054.
- Gao, Zhidong, Yu Zhang, Zhenxiao Zhang, Yanmin Gong & Yuanxiong Guo (2024), ‘Fedpt: Federated proxy-tuning of large language models on resource-constrained edge devices’, *ArXiv* **abs/2410.00362**.
URL: <https://api.semanticscholar.org/CorpusID:273022568>

- Ge, Shiming, Zhao Luo, Shengwei Zhao, Xin Jin & Xiao-Yu Zhang (2017), Compressing deep neural networks for efficient visual inference, *em* ‘2017 IEEE International Conference on Multimedia and Expo (ICME)’, pp. 667–672.
- Gongyo, Shinya, Jinrong Liang, Mitsuru Ambai, Rei Kawakami & Ikuro Sato (2025), Learning non-uniform step sizes for neural network quantization, *em* M.Cho, I.Laptev, D.Tran, A.Yao & H.Zha, eds., ‘Computer Vision – ACCV 2024’, Springer Nature Singapore, Singapore, pp. 55–73.
- Gower, J.C. (1985), ‘Properties of euclidean and non-euclidean distance matrices’, *Linear Algebra and its Applications* **67**, 81–97.
URL: <https://www.sciencedirect.com/science/article/pii/0024379585901879>
- Grattafiori, Aaron, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chribes McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira

Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Rapparthi, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Shesha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Couderc, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippas Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal

Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojuan Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao & Zhiyu Ma (2024), ‘The llama 3 herd of models’.

URL: <https://arxiv.org/abs/2407.21783>

Gross, Ronit D., Yarden Tzach, Tal Halevi, Ella Koresh & Ido Kanter (2026), ‘Tiny language models’, *Physica A: Statistical Mechanics and its Applications* **681**, 131102.

URL: <https://www.sciencedirect.com/science/article/pii/S037843712500754X>

Gultom, Syawal, S. Sriadhi, M. Martiano & Janner Simarmata (2018), *Comparison analysis of K-Means and K-Medoid with Ecludience Distance Algorithm, Chanberra Distance, and Chebyshev Distance for Big Data Clustering*, Vol. 420, IOP Publishing.

URL: <https://doi.org/10.1088/1757-899X/420/1/012092>

- Hahsler, Michael & Matthew Bolaños (2016), ‘Clustering data streams based on shared density between micro-clusters’, *IEEE Transactions on Knowledge and Data Engineering* **28**(6), 1449–1461.
- Hamming, R. W. (1950), ‘Error detecting and error correcting codes’, *The Bell System Technical Journal* **29**(2), 147–160.
- Han, Song, Huizi Mao & William J Dally (2015), ‘Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding’, *arXiv preprint arXiv:1510.00149*.
- Hao, Zixu, Huiqiang Jiang, Shiqi Jiang, Ju Ren & Ting Cao (2024), Hybrid slm and llm for edge-cloud collaborative inference, *em* ‘Proceedings of the Workshop on Edge and Mobile Foundation Models’, EdgeFM ’24, Association for Computing Machinery, New York, NY, USA, p. 36–41.
URL: <https://doi.org/10.1145/3662006.3662067>
- He, Kun, Weichao Li, Yutong Liu & Xiaohui Wang (2025), ‘Security and privacy in deep learning: A survey’, *IEEE Transactions on Dependable and Secure Computing*. Early Access.
- Hendrycks, Dan & Kevin Gimpel (2016), ‘Gaussian Error Linear Units (GELUs)’.
- Hu, Shengding, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, dahai li, Zhiyuan Liu & Maosong Sun (2024), MiniCPM: Unveiling the potential of small language models with scalable training strategies, *em* ‘First Conference on Language Modeling’.
URL: <https://openreview.net/forum?id=3X2L2TFr0f>
- Huang, Kai, Bowen Li, Qiang Wang & Jun Zhang (2024), ‘Edgellm: A cpu-fpga heterogeneous accelerator for efficient large language model inference’, *ACM Transactions on Reconfigurable Technology and Systems* **17**(2), 1–25.
- Huang, Yo-Ping & Simon Peter Khabusi (2025), ‘Artificial intelligence of things (aiot) advances in aquaculture: A review’, *Processes* **13**(1).
URL: <https://www.mdpi.com/2227-9717/13/1/73>
- Intel Corporation (2024), ‘Openvino toolkit: Optimizing and deploying deep learning models’, <https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html>. Accessed: 2025-01-15.
- Islam, Md. Rafiqul, Aditya Kumar & Wei Chen (2024), ‘Characterizing the performance of small language models on edge devices’, *ACM Transactions on Embedded Computing Systems* **23**(4), 1–28.

- ISO 26262: *Road Vehicles – Functional Safety* (2018), International Organization for Standardization. Second Edition.
- Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam & Dmitry Kalenichenko (2018), Quantization and training of neural networks for efficient integer-arithmetic-only inference, *em* ‘2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition’, pp. 2704–2713.
- Jung, Seok-ki, Joonho Park, Hyunmin Kim & Sanghoon Lee (2025), ‘Optimizing tiny transformers for microcontroller deployment with fused-weight attention’, *IEEE Embedded Systems Letters* **17**(1), 23–26.
- Junior, Antonio Carlos, Roberto Silva & Marcos Oliveira (2024), Cloud-based assistance systems for automotive applications: Challenges and opportunities, *em* ‘Proceedings of the IEEE International Conference on Intelligent Transportation Systems’, pp. 1234–1241.
- Jurafsky, Daniel & James H. Martin (2009), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2ndª edição, Prentice Hall.
- Jégou, Herve, Matthijs Douze & Cordelia Schmid (2011), ‘Product quantization for nearest neighbor search’, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **33**(1), 117–128.
- Kim, Sehoon, Amir Gholami, Zhewei Yao, Michael W. Mahoney & Kurt Keutzer (2021), I-bert: Integer-only bert quantization, *em* M.Meila & T.Zhang, eds., ‘Proceedings of the 38th International Conference on Machine Learning’, Vol. 139 de *Proceedings of Machine Learning Research*, PMLR, pp. 5506–5518.
URL: <https://proceedings.mlr.press/v139/kim21d.html>
- Kim, Seong-Whan, Joon-Hyuk Park & Sang-Min Lee (2024), ‘Advanced electronics systems for automotive applications’, *Electronics* **13**(24), 4972.
- Kingma, Diederik P. & Jimmy Ba (2014), ‘Adam: A method for stochastic optimization’, *CoRR* **abs/1412.6980**.
URL: <https://api.semanticscholar.org/CorpusID:6628106>
- Kuzmin, Andrey, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters & Tijmen Blankevoort (2022), Fp8 quantization: the power of the exponent, *em* ‘Proceedings of the 36th International Conference on Neural Information Processing Systems’, NIPS ’22, Curran Associates Inc., Red Hook, NY, USA.
- Lacoste, Alexandre, Alexandra Sasha Luccioni, Victor Schmidt & Thomas Dandres (2019), ‘Quantifying the carbon emissions of machine learning’, *ArXiv* **abs/1910.09700**.
URL: <https://api.semanticscholar.org/CorpusID:204823751>

- Lasi, Heiner, Peter Fettke, Hans-Georg Kemper, Thomas Feld & Michael Hoffmann (2014), 'Industry 4.0', *Business & Information Systems Engineering* **6**(4), 239–242.
URL: <https://doi.org/10.1007/s12599-014-0334-4>
- Lee, Donghyun, Sungjin Park & Youngmin Kim (2023), 'Llm-based chatbots for technical documentation: Design and evaluation', *ACM Transactions on Computer-Human Interaction* **30**(4), 1–32.
- Li, Minxuan (2019), An improved fcm clustering algorithm based on cosine similarity, *em* 'Proceedings of the 2019 International Conference on Data Mining and Machine Learning', ICDMML 2019, Association for Computing Machinery, New York, NY, USA, p. 103–109.
URL: <https://doi.org/10.1145/3335656.3335693>
- Liang, Yiwei, Cheng Chen, Ziwei Liu, Jiaming Yu & Song Han (2023), Mcuformer: Deploying vision transformers on microcontrollers with limited memory, *em* 'Advances in Neural Information Processing Systems 36 (NeurIPS 2023)', Curran Associates, Inc., pp. 8765–8779.
- Lin, Ji, Wei-Ming Chen, Han Cai, Chuang Gan & Song Han (2023), 'McuBERT: Memory-efficient BERT inference on commodity microcontrollers', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **42**(11), 3912–3925.
- Liu, Zechun, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai & Vikas Chandra (2024), MobileLLM: optimizing sub-billion parameter language models for on-device use cases, *em* 'Proceedings of the 41st International Conference on Machine Learning', ICML'24, JMLR.org.
- Lu, Zhenyan, Xiang Li, Dongqi Cai, Rongjie Yi, Fangming Liu, Wei Liu, Jian Luan, Xiwen Zhang, Nicholas D. Lane & Mengwei Xu (2025), Demystifying small language models for edge deployment, *em* W.Che, J.Nabende, E.Shutova & M. T.Pilehvar, eds., 'Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', Association for Computational Linguistics, Vienna, Austria, pp. 14747–14764.
URL: <https://aclanthology.org/2025.acl-long.718/>
- Mahalanobis, Prasanta Chandra (1936), 'On the generalized distance in statistics', *Proceedings of the National Institute of Sciences of India* **2**(1), 49–55.
- Masud, Sultana, Md Shamsul Alam & Md Jashim Uddin (2024), 'A comprehensive survey on quantization techniques for deep neural networks', *ACM Computing Surveys* **56**(8), 1–37.
- Naveed, Humza, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes & Ajmal Mian (2025), 'A comprehensive overview of large language models', *ACM Trans. Intell. Syst. Technol.* **16**(5).
URL: <https://doi.org/10.1145/3744746>

- NVIDIA Corporation (2024), ‘Nvidia tensorrt: High-performance deep learning inference’, <https://developer.nvidia.com/tensorrt>. Accessed: 2025-01-15.
- Papineni, Kishore, Salim Roukos, Todd Ward & Wei-Jing Zhu (2002), Bleu: a method for automatic evaluation of machine translation, *em* ‘Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics’, Association for Computational Linguistics, pp. 311–318.
- Park, Jihoon, Minsoo Kim & Sangho Lee (2024), Maqa: A question-answering system for automotive manuals using large language models, *em* ‘Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)’, pp. 1–14.
- Patterson, David A., Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier & Jeff Dean (2021), ‘Carbon emissions and large neural network training’, *CoRR* **abs/2104.10350**.
URL: <https://arxiv.org/abs/2104.10350>
- Pham, Nghiem Thanh, Tung Kieu, Duc Manh Nguyen, Son Ha Xuan, Nghia Duong-Trung & Danh Le-Phuoc (2025), SLM-bench: A comprehensive benchmark of small language models on environmental impacts, *em* C.Christodoulopoulos, T.Chakraborty, C.Rose & V.Peng, eds., ‘Findings of the Association for Computational Linguistics: EMNLP 2025’, Association for Computational Linguistics, Suzhou, China, pp. 21369–21392.
URL: <https://aclanthology.org/2025.findings-emnlp.1165/>
- Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei & Ilya Sutskever (2019), ‘Language models are unsupervised multitask learners’, *OpenAI blog* **1**(8), 9.
- Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev & Percy Liang (2016), SQuAD: 100,000+ questions for machine comprehension of text, *em* J.Su, K.Duh & X.Carreras, eds., ‘Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing’, Association for Computational Linguistics, Austin, Texas, pp. 2383–2392.
URL: <https://aclanthology.org/D16-1264/>
- Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon & Ali Farhadi (2016), Xnet: Imagenet classification using binary convolutional neural networks, *em* B.Leibe, J.Matas, N.Sebe & M.Welling, eds., ‘Computer Vision – ECCV 2016’, Springer International Publishing, Cham, pp. 525–542.
- Reiswig, Jennifer (2010), ‘Mendeley’, *Journal of the Medical Library Association: JMLA* **98**(2), 193.
- Rokh, Babak, Ali Azarpeyvand & Alireza Khanteymoori (2023), ‘A comprehensive survey on model quantization for deep neural networks in image classification’, *ACM Trans. Intell. Syst. Technol.* **14**(6).
URL: <https://doi.org/10.1145/3623402>

- Sakoe, H. & S. Chiba (1978), ‘Dynamic programming algorithm optimization for spoken word recognition’, *IEEE Transactions on Acoustics, Speech, and Signal Processing* **26**(1), 43–49.
- Santos, Filipe M.P., Alexandre Gonçalves, João M.C. Sousa & Susana M. Vieira (2025), Distilling knowledge from deep neural networks to neuro-fuzzy inference systems, *em* ‘2025 IEEE International Conference on Fuzzy Systems (FUZZ)’, pp. 1–6.
- Sanyal, Souvik, Zichun Cheng, Abhinav Kumar & Haoyu Tang (2024), ‘Entrollm: Entropy-based mixed quantization and huffman coding for edge llm compression’, *IEEE Access* **12**, 89754–89767.
- Scherer, Moritz, Gamm Rutishauser, Luca Cavigelli & Luca Benini (2024), Deeploy: Compiler infrastructure for deploying small language models on heterogeneous microcontrollers, *em* ‘Proceedings of the 2024 Design, Automation & Test in Europe Conference’, pp. 1234–1239.
- Scherer, Moritz, Luka Macan, Victor J. B. Jung, Philip Wiese, Luca Bompani, Alessio Burrello, Francesco Conti & Luca Benini (2024), ‘Deeploy: Enabling energy-efficient deployment of small language models on heterogeneous microcontrollers’, *Trans. Comp.-Aided Des. Integ. Cir. Sys.* **43**(11), 4009–4020.
URL: <https://doi.org/10.1109/TCAD.2024.3443718>
- Schizas, Nikolaos, Aristeidis Karras, Christos Karras & Spyros Sioutas (2022), ‘Tinymml for ultra-low power ai and large scale iot deployments: A systematic review’, *Future Internet* **14**(12), 363.
- Seide, Frank, Hao Fu, Jasha Droppo, Gang Li & Dong Yu (2014), 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns, *em* ‘Interspeech’, pp. 1058–1062.
- Strubell, Emma, Ananya Ganesh & Andrew McCallum (2019), Energy and policy considerations for deep learning in NLP, *em* A.Korhonen, D.Traum & L.Màrquez, eds., ‘Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics’, Association for Computational Linguistics, Florence, Italy, pp. 3645–3650.
URL: <https://aclanthology.org/P19-1355/>
- Suwanda, R, Z Syahputra & E M Zamzami (2020), ‘Analysis of euclidean distance and manhattan distance in the k-means algorithm for variations number of centroid k’, *Journal of Physics: Conference Series* **1566**(1), 012058.
URL: <https://doi.org/10.1088/1742-6596/1566/1/012058>
- Tami, Reda, Hossam El-Sayed & Ezedin Barka (2024), ‘Hazardnet: A compact vision language model for real-time traffic safety detection on edge devices’, *IEEE Transactions on Intelligent Transportation Systems* **25**(8), 10234–10247.

- Thangaraju, Baskar, Chitra Palanisamy, Manivannan Ramasamy, Gayathri Joshi & Seifedine Kadry (2024), ‘Centralized fog computing security platform for iot and cloud in healthcare system’, *IEEE Access* **12**, 45123–45138.
- Urbann, Oliver, Simon Camphausen, Arne Moos, Ingmar Schwarz, Sören Kerner & Maximilian Otten (2020), ‘A c code generator for fast inference and simple deployment of convolutional neural networks on resource constrained systems’, *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)* pp. 1–7.
URL: <https://api.semanticscholar.org/CorpusID:210701292>
- Vasershtein, Leonid N (1969), ‘Markov processes over denumerable products of spaces describing large systems of automata’, *Problemy Peredachi Informatsii* **5**(3), 64–72.
- Vasuki, A. & P.T. Vanathi (2006), ‘A review of vector quantization techniques’, *IEEE Potentials* **25**(4), 39–47.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser & Illia Polosukhin (2017), Attention is all you need, *em* ‘Proceedings of the 31st International Conference on Neural Information Processing Systems’, NIPS’17, Curran Associates Inc., Red Hook, NY, USA, p. 6000–6010.
- Wang, Kuan, Zhijian Liu, Yujun Lin, Ji Lin & Song Han (2019), Haq: Hardware-aware automated quantization with mixed precision, *em* ‘2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)’, pp. 8604–8612.
- Wang, Xubin, Zhiqing Tang, Jianxiong Guo, Tianhui Meng, Chenhao Wang, Tian Wang & Weijia Jia (2025), ‘Empowering edge intelligence: A comprehensive survey on on-device ai models’, *ACM Comput. Surv.* **57**(9).
URL: <https://doi.org/10.1145/3724420>
- Wang, Yutong & Cindy Rubio-González (2024), Predicting performance and accuracy of mixed-precision programs for precision tuning, *em* ‘2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)’, pp. 152–164.
- Warden, Pete & Daniel Situnayake (2022), ‘Machine learning for microcontroller-class hardware: A survey’, *IEEE Micro* **42**(5), 18–26.
- Wen, Wei, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen & Hai Li (2017), Terngrad: ternary gradients to reduce communication in distributed deep learning, *em* ‘Proceedings of the 31st International Conference on Neural Information Processing Systems’, NIPS’17, Curran Associates Inc., Red Hook, NY, USA, p. 1508–1518.
- Weyssow, Martin, Xin Zhou, Kisub Kim, David Lo & Houari Sahraoui (2025), ‘Exploring parameter-efficient fine-tuning techniques for code generation with large language models’, *ACM Trans. Softw. Eng. Methodol.* **34**(7).
URL: <https://doi.org/10.1145/3714461>

- Wu, Xiaofan, Ruiqi Zhang, Ruimin Guo, Wei Zhang & Yanzhi Chen (2019), ‘Vector quantization for neural network compression’, *IEEE Transactions on Neural Networks and Learning Systems* **30**(11), 3567–3579.
- Xu, Hang, Wenhua Zeng, Xiangxiang Zeng & Gary G. Yen (2019), ‘An evolutionary algorithm based on minkowski distance for many-objective optimization’, *IEEE Transactions on Cybernetics* **49**(11), 3968–3979.
- Yang, Chao, Jiaming Wang, Hongyang Zhang, Bo Yuan & Tianyi Liu (2024), ‘Privacy-preserving machine learning: Threats, defenses, and opportunities’, *IEEE Access* **12**, 67123–67145.
- Yang, Jingwei, Yicheng Wang, Yuze Lin & Qian Gao (2023), Tinyformer: Efficient transformer models for microcontrollers, *em* ‘Proceedings of the 2023 ACM/IEEE International Conference on Computer-Aided Design’, pp. 1–9.
- Yao, Yuan, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Chi Chen, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Ronghua Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, Guoyang Zeng, Dahai Li, Zhiyuan Liu & Maosong Sun (2025), ‘Efficient gpt-4v level multimodal large language model for deployment on edge devices’, *Nature Communications* **16**(1), 5509.
URL: <https://doi.org/10.1038/s41467-025-61040-5>
- Yauri, Ricardo & Rafael Espino (2024), ‘Generative language model technology integrated into an iot device for the development of a voice assistant’, *WSEAS Transactions on Systems* **23**, 521–530.
- Yu, Zhongzhi, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang (Katie) Zhao & Yingyan (Celine) Lin (2024), Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting, *em* ‘Proceedings of the 61st ACM/IEEE Design Automation Conference’, DAC ’24, Association for Computing Machinery, New York, NY, USA.
URL: <https://doi.org/10.1145/3649329.3658473>
- Zhang, Biao & Rico Sennrich (2019), *Root mean square layer normalization*, Curran Associates Inc., Red Hook, NY, USA.
- Zhang, Peiyuan, Guangtao Zeng, Tianduo Wang & Wei Lu (2024), ‘Tinyllama: An open-source small language model’, *arXiv preprint arXiv:2401.02385*.
- Zheng, Yiaowei, B.A. Julstrom & Weidong Cheng (1997), Design of vector quantization codebooks using a genetic algorithm, *em* ‘Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC ’97)’, pp. 525–529.

- Zhou, Shaoyuan, Zimu Zhang, Zeyu Xu, Xinran Chen & Qun Wang (2024), Edgemoe: Fast and efficient on-device inference of moe-based large language models, *em* ‘Proceedings of the 30th Annual International Conference on Mobile Computing and Networking’, pp. 456–471.
- Zhou, Zihan, Simon Kurz & Zhixue Zhao (2025), Revisiting pruning vs quantization for small language models, *em* C.Christodoulopoulos, T.Chakraborty, C.Rose & V.Peng, eds., ‘Findings of the Association for Computational Linguistics: EMNLP 2025’, Association for Computational Linguistics, Suzhou, China, pp. 12055–12070.
- URL:** <https://aclanthology.org/2025.findings-emnlp.645/>