Institute of Mechanism Theory, Machine Dynamics and Robotics

RWTH AACHEN UNIVERSITY

This thesis was submitted to the Institute of Mechanism Theory, Machine Dynamics and Robotics

Optimization of Small Language Models for Embedded Voice Assistance

Master Thesis

by:

Maroof Mohammed Abdul Aziz M.Sc.

Student number: 440542

supervised by:

M.Sc. Sophie Charlotte Keunecke

M.Sc. Matthias Lemberger

Examiner:

Univ.-Prof. Dr.-Ing. Dr. h. c. Burkhard Corves

Prof. Dr.-Ing. Mathias Hüsing

Aachen, 26 May 2025

Master Thesis

by Maroof Mohammed Abdul Aziz M.Sc.
Student number: 440542

Optimization of Small Language Models for Embedded Voice Assistance

---

The emergence of Large Language Models (LLMs) in recent years represents a significant advancement in Natural Language Processing (NLP) and Artificial Intelligence (AI). These models are revolutionizing various industries by powering intelligent assistants that enhance the customer experience in unprecedented ways. Trained on extensive datasets, LLMs possess the ability to comprehend context, provide human-like responses, and perform a variety of language-based tasks. Traditional voice command systems often struggle with understanding complex queries due to use of rule based and statistical methods lacking contextual awareness. LLMs, on the other hand, can interpret and respond to natural language, allowing users to interact as they would with another person. This makes the interaction more fluid and natural, improving overall user satisfaction. However, the scaling of LLMs to enhance their capabilities have led to certain limitations in resource-constrained environments. The large number of parameters and high computational needs necessitate powerful GPUs with with significant memory and processing capabilities. Embedded applications, in particular, often lack the computational power of GPUs and rely solely on CPUs, which limits their ability to run large-scale models effectively. As a result, they depend on cloud or on-premise infrastructure, leading to high costs and needing reliable network connectivity. This has led to a growing focus on Small Language Models for embedded applications, as seen in the rising number of research papers and recent announcements from major companies like Apple, MetaAI, and Google. Future advancements could see these models playing a central role in vehicle assistants and human assistance robots, where understanding and predicting human needs will be essential. The objective of this thesis is to research efficient alternatives to existing state of the art large models for embedded applications with limited hardware. The solution shall be able to run inference with limited compute and memory requirements while providing efficient and accurate in-vehicle assistance functionalities comparable to state of the art models. Topics such as model finetuning, compression and accelerators will be examined and relevant techniques implemented to achieve the optimal solution.

Supervisor: M.Sc. Sophie Charlotte Keunecke

Eidesstattliche Versicherung

Maroof Mohammed Abdul Aziz Matrikel-Nummer: 440542

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Master Thesis mit dem Titel

Optimization of Small Language Models for Embedded Voice Assistance

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, 26 May 2025

Maroof Mohammed Abdul Aziz

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, 26 May 2025

Maroof Mohammed Abdul Aziz

The present translation is for your convenience only.
Only the German version is legally binding.

Statutory Declaration in Lieu of an Oath

Maroof Mohammed Abdul Aziz Student number: 440542

I hereby declare in lieu of an oath that I have completed the present Master Thesis entitled

Optimization of Small Language Models for Embedded Voice Assistance

independently and without illegitimate assistance from third parties. I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, 26 May 2025 Maroof Mohammed Abdul Aziz

Official Notification:

Para. 156 StGB (German Criminal Code): False Statutory Declarations

Whosoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence

(1)If a person commits one of the offences listed in sections 154 to 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly. I have read and understood the above official notification: :

Aachen, 26 May 2025 Maroof Mohammed Abdul Aziz

Contents

Formula symbols and indices

λ	–	learning rate
$\in$	–	is an element of
$\rightarrow$	–	assignment
$\sim$	–	selected from
$\cup$	–	set union
exp	–	exponentiation of a function
t	–	timestep
w	–	token or word
n	–	sequence length
P	–	probability distribution
$\mathcal{L}_{\mathbf{MLM}}$	–	loss for masked language modeling
X	–	input sequence
$d_{\mathbf{model}}$	–	embedding dimension
d_k	–	dimension of key/query vectors
Q	–	query matrix
K	–	key matrix
V	–	value matrix
W^Q	–	projection matrix for queries
W^K	–	projection matrix for keys
W^V	–	projection matrix for values
W^O	–	output projection matrix after multi-head attention
M	–	attention mask matrix

List of abbreviations

AI	Artificial Intelligence
NLP	Natural Language Processing
NLU	Natural Language Understanding
NLG	Natural Language Generation
LLM	Large Language Models
SLM	Small Language Models
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning

1 Introduction

The emergence of Large Language Models (LLMs) in recent years marks a significant breakthrough in the fields of Natural Language Processing (NLP) and Artificial Intelligence (AI). These models are transforming industries by powering intelligent systems capable of understanding context, generating human-like responses, and handling a wide range of language-related tasks.

Traditional voice command systems, which rely on rule-based or statistical methods, often struggle with understanding nuanced or complex queries due to limited contextual awareness. In contrast, LLMs enable more natural and fluid interactions, allowing users to communicate in everyday language and enhancing the overall user experience.

Despite their capabilities, LLMs come with substantial computational and memory requirements. Their performance typically depends on powerful GPUs with high parallelism and large memory footprints. This poses a challenge for embedded applications, which often operate with limited hardware—frequently CPU-only environments—making it difficult to run such models locally. Consequently, many systems rely on cloud-based inference, leading to higher costs, latency, and dependence on stable network connectivity.

To address these challenges, there is growing interest in Small Language Models (SLMs) that are optimized for deployment in resource-constrained environments. Recent efforts by major companies such as Apple, MetaAI, and Google underscore the importance and relevance of this direction. These compact models aim to strike a balance between performance and efficiency, making them suitable for applications like in-vehicle assistants and personal robotics, where real-time, offline language understanding is crucial.

This thesis explores the optimization of Small Language Models for embedded voice assistance. The goal is to investigate efficient alternatives to current large-scale models that can operate with limited computational and memory resources, while still delivering accurate and responsive language capabilities. Topics such as fine-tuning, model compression (including pruning and quantization), and the use of inference-time optimizations will be explored to identify and implement an optimal solution for embedded use cases.

2 Fundamentals

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of computer science and artificial intelligence concerned with enabling computers to understand, interpret, and generate human language in a meaningful way. It combines techniques from linguistics, machine learning, and deep learning to perform tasks such as sentiment analysis, machine translation, question answering, text summarization, and conversational AI.

NLP is commonly divided into two major areas:

- **Natural Language Understanding (NLU):** Focuses on interpreting the meaning behind text, including tasks like intent classification, entity recognition, semantic parsing, and sentiment analysis.
- **Natural Language Generation (NLG):** Focuses on producing coherent and contextually appropriate human-like text from structured or unstructured data, including tasks such as summarization, report generation, and dialogue generation.

2.2 Language Models

A Language Model (LM) estimates the probability distribution over sequences of words, enabling the generation of coherent text and the prediction of missing elements. Traditional models such as n-grams were limited by their reliance on fixed-size context windows and sparse representations.

Modern neural language models capture complex long-range dependencies using dense vector representations and are generally trained using one of two approaches:

- **Autoregressive Modeling (e.g., GPT):** Models that predict the next token given the previous tokens. These models are typically used for generative tasks.

$$P(w_1, w_2, \ldots, w_n) = \prod_{t=1}^{n} P(w_t | w_1, \ldots, w_{t-1}) \tag{2.1}$$

- **Masked Language Modeling (e.g., BERT):** Models that predict masked tokens based on surrounding context, enabling bidirectional understanding. These models

are widely used for tasks such as classification, question answering, and embedding generation.

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \text{masked}} \log P(w_i | \mathbf{w}_{\setminus i}) \tag{2.2}$$

2.3 Voice Assistants and NLP

Voice assistants are one of the most popular applications of NLP, allowing users to interact with devices via natural spoken language. Examples include Apple's Siri, Amazon's Alexa, and Google's Assistant. These systems perform tasks such as answering questions, controlling smart home devices, managing calendars, and providing entertainment.

Earlier generations of voice assistants relied on rule-based dialogue systems and structured, pre-defined flows. Advances in NLP and machine learning have enabled the development of more conversational, flexible, and context-aware voice assistants, increasingly powered by large pre-trained language models.

2.4 Evolution of Voice Assistants

- **Rule-Based Systems:** Early systems such as ELIZA [Wei66] and ALICE [Wal03] used pattern matching and predefined response templates. While innovative at the time, they lacked memory, context awareness, and adaptability.
- **Statistical Approaches:** In the 1990s, statistical learning methods like Bag-of-Words [Joa98] and Support Vector Machines [CV95] enabled greater flexibility and robustness, thanks to the availability of larger datasets and improved computational resources.
- **Neural Networks and Word Embeddings:** The 2000s saw the rise of neural network models such as Recurrent Neural Networks (RNNs) [Elm90] and Long Short-Term Memory (LSTM) networks [HS97] for sequence modeling. Word embeddings like Word2Vec [Mik13] and GloVe [PSM14] provided dense vector representations that captured semantic relationships between words.
- **NLU and Dialogue Management:** Voice assistants evolved to include modules for intent detection, entity recognition [Lam16], and dialogue state tracking. These allowed systems to better manage conversations and support complex, task-oriented interactions.

- **Sequence-to-Sequence Models and Attention Mechanisms:** Encoder-decoder architectures [SVL14] and attention mechanisms [BCB15] improved the flexibility and performance of systems on a wide range of tasks including translation and summarization.

- **Transformers and Pretrained Language Models:** The Transformer architecture [Vas17], introduced in 2017, revolutionized NLP by enabling parallelization and efficient long-range dependency modeling. Models like BERT [Dev18] and GPT [Rad18] were trained on massive corpora and fine-tuned for specific downstream tasks.

- **Large Language Models (LLMs):** Recent LLMs [Gra24; AAA24; YYZ24; Dee23; Ben25] achieve state-of-the-art performance but require significant computational resources. This has led to the parallel development of Small Language Models (SLMs) [Wan24], optimized for resource-constrained environments.

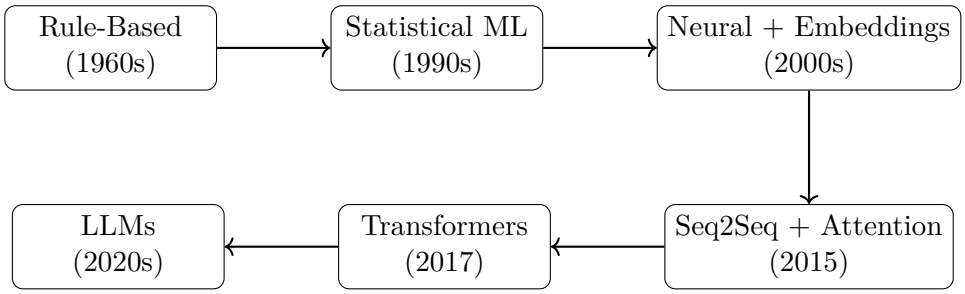

Figure 2.1: Evolution of Voice Assistant Architectures

2.5 The Transformer Architecture

Transformers are the foundation of most modern language models. Unlike earlier architectures that relied on recurrence or convolutions, Transformers use self-attention mechanisms exclusively, allowing for better parallelization and modeling of long-range dependencies.

Core Components

The Transformer architecture is built from several core components:

- **Embedding Layer:** Converts input tokens into dense vector representations, combining token embeddings with positional encodings to retain sequential information:

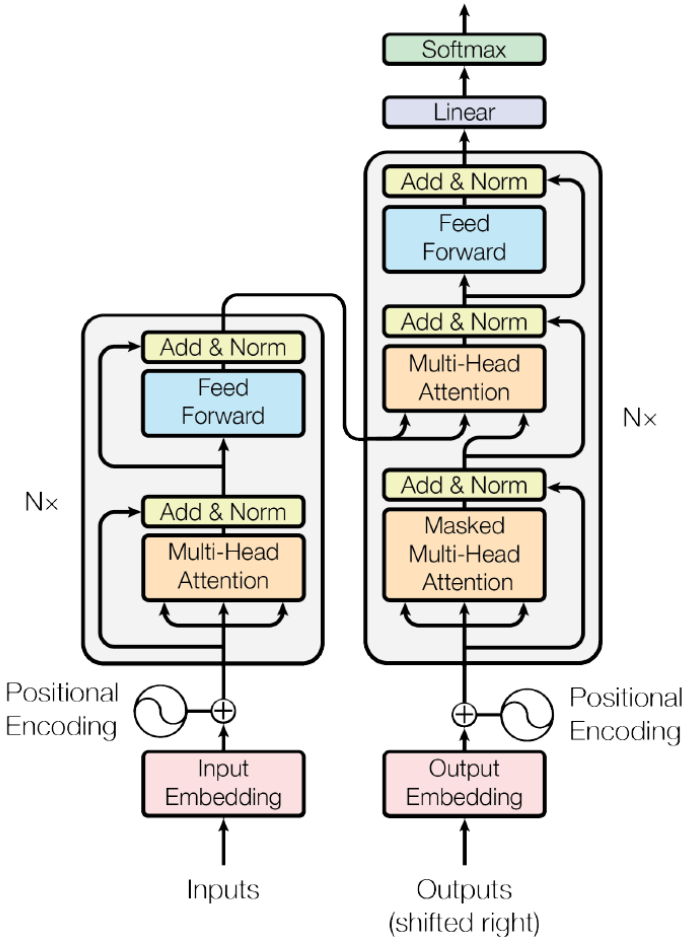

Figure 2.2: Transformer Architecture [Vas17]

$$\text{Input Representation} = \text{Token Embedding} + \text{Positional Encoding} \tag{2.3}$$

- **Multi-Head Attention:** Allows the model to attend to different parts of the input sequence from multiple perspectives. It operates on three fundamental representations derived from the input:
  - **Query (Q):** Represents the request for information.
  - **Key (K):** Represents the information available for matching.
  - **Value (V):** Contains the content to be aggregated based on the attention scores.

  Given an input sequence $X \in \mathbb{R}^{n \times d_{\text{model}}}$, the matrices are computed as:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{2.4}$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learnable projection matrices.

The attention score between a query and a key is computed using the scaled dot product:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.5}$$

where the scaling factor $\sqrt{d_k}$ stabilizes gradients during training. Higher alignment between queries and keys results in larger attention scores, thus assigning greater weight to corresponding values during aggregation.

Multi-head attention extends this by performing multiple parallel attention operations, each with its own set of projections, and then concatenating their outputs:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O \tag{2.6}$$

where each head is defined as:

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.7}$$

and $W^O \in \mathbb{R}^{hd_k \times d_{\text{model}}}$ is a learnable matrix.

- **Feedforward Network (FFN):** Applies two linear transformations with a nonlinear activation (typically ReLU) in between, independently to each token:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.8}$$

where W_1, W_2 are weight matrices and b_1, b_2 are biases.

- **Residual Connections and Layer Normalization:** Applied around each sublayer (attention and FFN) to improve gradient flow and model convergence:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \tag{2.9}$$

- **Masked Attention (in Decoders):** During training, to preserve the autoregressive property, a mask M is added before the softmax to prevent positions from attending to future tokens:

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V \tag{2.10}$$

where masked positions are assigned large negative values to nullify their impact.

Model Variants

- **Encoder-Only Models:** (e.g., BERT) — designed for understanding tasks like classification, entity recognition.
- **Decoder-Only Models:** (e.g., GPT) — designed for text generation and continuation.
- **Encoder-Decoder Models:** (e.g., T5) — ideal for sequence-to-sequence tasks like translation and summarization.

2.6 Knowledge Enhancement techniques

There has been extensive research into enhancing the knowledge and capabilities of language models. Some of the most impactful methods include:

- **Fine-Tuning:** Adapting a pre-trained model to a specific domain or task by continuing training on a smaller, specialized dataset.
- **Retrieval-Augmented Generation (RAG):** Combines LLMs with external knowledge bases by retrieving relevant documents during inference, reducing hallucination and increasing factual accuracy.
- **Function Calling:** Empowers LLMs to interact with external APIs, enabling dynamic capabilities such as booking services or checking live information.

2.7 Need for Small Language Models

Although LLMs deliver impressive capabilities, their computational requirements make them unsuitable for many real-world applications, particularly on edge devices or embedded systems. Furthermore, cloud-based deployments raise concerns around latency, cost, and user privacy.

This has fueled the development of Small Language Models (SLMs), which aim to:

- Operate efficiently on CPU-based or low-memory environments.

- Maintain competitive performance, particularly in domain-specific or specialized tasks.
- Enable offline, private, and low-latency applications, such as on-device voice assistants.

Research and industry adoption in this area is growing rapidly, with many leading organizations now publishing models with fewer than 4 billion parameters to target resource-constrained deployments.

3 State of the Art

3.1 Large Language Models (LLMs)

Large Language Models (LLMs) such as GPT-4 [Ope23], Phi-3 [AAA24], Gemma 2 [Dee23], Llama3 [Gra24] and Qwen 2.5 [YYZ24] have significantly advanced natural language understanding and generation. These models achieve state-of-the-art performance through pretraining on massive datasets, leveraging architectures such as transformer-based dense attention, mixture-of-experts, and parameter-efficient design innovations. However, their deployment in embedded environments remains challenging due to substantial computational and memory demands, often exceeding the capabilities of edge processors [Ali23; Ram24].

3.2 Small Language Models (SLMs)

Small Language Models (SLMs) have emerged as a solution to bridge the gap between capability and efficiency under strict hardware constraints. Major model providers have released compact variants, including *Gemma 2 2B*, *Phi-3.5-mini*, *Qwen 2.5 3B*, and *Llama 3.2 1B*, offering competitive performance with drastically fewer parameters.

Additionally, new SLM architectures such as *SmolLM2*[Ben25], *OpenELM*, and *MobiLlama* showcase further advances in model scaling, optimization, and specialization for low-latency, memory-efficient deployment. Recent surveys [Lu24; Wan24] comprehensively cover the design principles underpinning these models, including width-reduced transformers, layer dropping, and adaptive pretraining strategies.

3.3 Function Calling and Tool Use

Function-calling augments LLMs with the ability to invoke external APIs, perform actions, or retrieve information dynamically—transforming them from passive generators to interactive agents. Recent research emphasizes several key directions for enabling robust function calling:

- **Special Token Insertion:** Defining delimiters and structured outputs that make function-call generation parseable[CL24].

- **Toolformer-style Self-supervision:** Models autonomously annotate prompts with tool-calling opportunities during training.
- **Schema-driven Decoding:** Output structured responses compliant with predefined function call schemas or JSON formats.
- **Dynamic Planning and Execution:** Multi-step task decomposition where models choose, sequence, and verify function calls dynamically [Hua24].

Survey studies such as [Qu24] highlight that tool usage integration improves task complexity handling, agentic capabilities, and reliability in both small and large models.

3.4 Finetuning Strategies

Finetuning allows the adaptation of general-purpose pretrained models to specialized domains or tasks. Major approaches include:

- **Full Finetuning:** Updating all parameters of the model, maximizing adaptation but requiring significant resources.
- **Parameter-Efficient Finetuning (PEFT):** Techniques like LoRA [Hu21] and QLoRA [Det23] inject lightweight trainable adapters, enabling specialization with a small fraction of parameters updated.
- **Alignment and Instruction Tuning:** Methods like Supervised Fine-Tuning (SFT), Reinforcement Learning with Human Feedback (RLHF), and Direct Preference Optimization (DPO) refine outputs for user alignment [Zho23].

Newer frameworks like LLM2LLM [Lee24] further automate dataset generation for specialized finetuning, reducing reliance on costly human annotation.

3.5 Model Compression Techniques

Compression is critical for embedding powerful LLM capabilities into constrained hardware environments. State-of-the-art techniques include:

- **Quantization:** Reducing precision (e.g., INT8, INT4, mixed precision) with minimal degradation in model quality [Ma24; Sax24].
- **Pruning:** Removing redundant neurons, heads, or even entire layers based on sensitivity or importance scoring [Mur24; Sre24].

- **Knowledge Distillation:** Training compact student models to mimic larger teacher models, enabling major reductions in size while preserving capabilities [HVD15; Lu24].

Careful combinations of these strategies are key to achieving high-accuracy, real-time inference in low-power embedded settings.

3.6 On-device Deployment Frameworks

Deployment frameworks play a critical role in bridging model optimization with real-world inference performance on edge devices:

- **llama.cpp:** Highly optimized C++ backend for CPU-based LLM inference, supporting 4-bit GGUF models and quantized computation [Geo24].

- **Ollama:** Simple-to-use runtime for GPU-accelerated inference of quantized models, supporting batching, streaming, and multi-threaded optimization [Oll24].

- **vLLM:** High-throughput, memory-efficient serving framework, particularly effective for batch inference on server-grade GPUs[vLL24].

- **ExecuTorch:** Lightweight PyTorch runtime designed for mobile/embedded deployment, integrating quantization and operator fusion natively [PyT24].

- **Google Edge TPU / AI Edge:** Specialized hardware accelerators and SDKs (e.g., TensorFlow Lite for Edge) to deploy quantized transformer models efficiently on mobile and IoT devices [Goo24].

Xu et al. [Xu24] further highlight critical runtime optimizations such as caching strategies, kernel fusion, and multi-threaded scheduling essential for efficient on-device execution.

The convergence of these frameworks with advances in model design and compression strategies has made the deployment of capable small LLMs on mobile and embedded platforms increasingly practical.

3.7 Related Work

A growing body of research focuses on enabling language model deployment under stringent memory, compute, and latency constraints. Key contributions include:

TinyAgent [Erd24] presents a lightweight dual-stage fine-tuning framework for function-calling SLMs at the edge. By combining synthetic data generation and teacher-student distillation, it achieves robust tool usage while minimizing model footprint and latency.

Octopus v2 [CL24] proposes a multi-agent modular design optimized for on-device execution. The model coordinates multiple specialized agents for tool-augmented reasoning, leveraging a tool-augmented dataset for supervised fine-tuning.

Manduzio et al. [Man24] investigate methods to enhance small model reliability in function calling via reward-based tuning and synthetic reasoning corpora, demonstrating that sub-2B parameter models can achieve competitive structured reasoning capabilities.

LLM in a Flash [Ali23] introduces memory optimization techniques like sliding window attention and memory paging to enable long-context inference even on low-VRAM devices.

MobileLLM [Liu24] targets the co-design of sub-billion parameter architectures and quantization-aware training pipelines, specifically for deployment on mobile-class CPUs and NPUs.

Khiabani et al. [Khi25] presents domain-specific prompt engineering, instruction tuning, and evaluation frameworks tailored to automotive assistant deployment, highlighting safety and latency as critical constraints.

4 Approach

This chapter presents the overall methodology adopted for optimizing Small Language Models (SLMs) for resource-constrained, embedded deployment. The thesis workflow consists of dataset generation, model benchmarking, pruning and finetuning, quantization, and on-device inference acceleration.

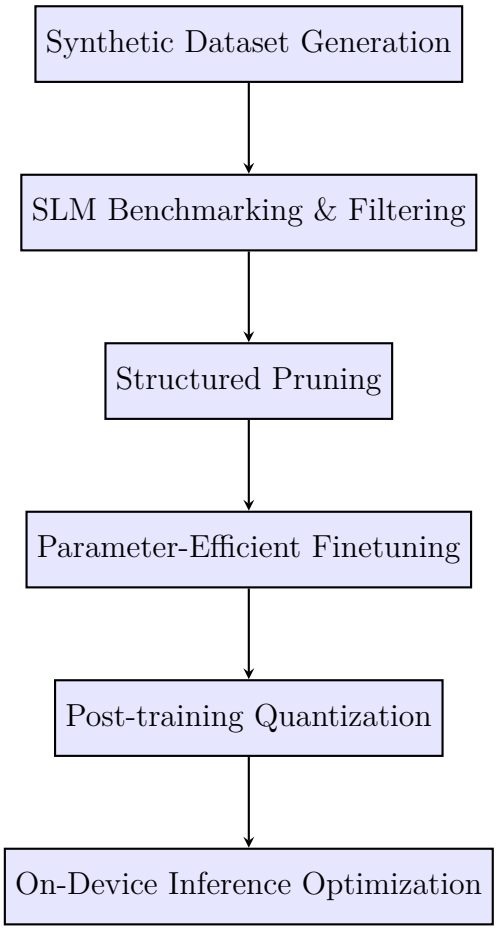

Figure 4.1: Thesis Workflow for Efficient Small Language Model Deployment

4.1 Tools and Frameworks

Only open-source models and tools were utilized throughout this thesis to ensure transparency, reproducibility, and suitability for deployment in embedded environments. The core frameworks and libraries included:

- **Python 3.10** – Development environment.
- **PyTorch** – Deep learning framework for model training and optimization.

- **Hugging Face Transformers and Datasets** – Access to pre-trained models and data utilities.
- **Unsloth** – High-performance LoRA-based finetuning library.
- **MLflow** – Experiment tracking, metric logging, and model versioning.
- **Azure Machine Learning Studio** – Multi-GPU training and distributed experiment management.
- **llama.cpp** – Lightweight C++ runtime for CPU-based quantized model inference.
- **Ollama** – GPU-accelerated inference engine for quantized models.
- **LM Evaluation Harness** – Standardized evaluation suite for benchmarking LLMs.

Several newer models released shortly before thesis submission could not be evaluated due to library compatibility issues and time constraints.

4.2 Datasets and Evaluation Tasks

To simulate in-vehicle assistant behavior, a synthetic dataset with 4 tool-calling tasks (Phone call, Navigation, Music and Vehicle control) was developed. Additionally, to evaluate general language understanding and reasoning, standard benchmarks were used:

- **Winogrande** – Commonsense reasoning through ability to dismbiguate pronouns and other references based on context.
- **HellaSwag** – Commonsense reasoning through Sentence completion given a context.

These benchmarks evaluation ensures both domain-specific functionality and general reasoning capability are assessed.

4.3 Hardware and Deployment Environment

Target Deployment Hardware

The intended final deployment target is a low-power, embedded SoC with the following specifications:

- **CPU:** Quad-core ARM Cortex-A57 MPCore processor

- **GPU:** NVIDIA Maxwell architecture with 128 CUDA cores
- **Memory:** 4 GB LPDDR4 @ 25.6 GB/s
- **Storage:** 16 GB eMMC 5.1

Direct testing on this platform was not feasible due to hardware unavailability. Hence final evaluation was performed on below device to emulate closest possible hardware specifications.

Thesis Evaluation Hardware

Experiments were conducted on the following system:

- **CPU:** Intel(R) Core(TM) i7-8850H (6 cores, 12 threads) @ 2.60GHz
- **GPU:** NVIDIA Quadro P2000 Mobile (4 GB GDDR5 VRAM)
- **RAM:** 62 GB DDR4
- **OS:** Ubuntu 22.04.4 LTS

This environment supports both CPU inference (via llama.cpp) and GPU inference (via Ollama).

4.4 Evaluation Metrics

GPT-4o-mini was used as the performance baseline for benchmarking and evaluation. Model performance was assessed using the following quantitative metrics:

- **Tool Accuracy:** Correct invocation of functions with proper parameters.
- **Syntax Accuracy:** Correct special token formatting in outputs.
- **Latency:** Average inference time per sample.
- **Memory Usage:** Peak RAM utilization during inference.
- **Tokens per Second (Throughput):** Inference speed.
- **Model Size:** Compressed model storage footprint.

4.5 Limitations

The scope of this thesis was defined by the following factors:

- Only finetuning, pruning, and quantization of open-source, pre-trained models were performed; no models were trained from scratch.
- Proof-of-concept tool-calling evaluations were conducted on a small synthetic dataset limited to 4 tool call definitions.
- HellaSwag benchmark was restricted to a 250-sample subset due to compute and time limitations.
- Real-time deployment with full voice assistant integration on target device was not implemented.

5 Implementation

5.1 Dataset generation

A task-specific dataset was constructed to finetune and evaluate Small Language Models (SLMs) in the context of embedded automotive voice assistants. The dataset was designed to simulate realistic user interactions involving tool invocation and general knowledge queries, with a focus on functional coverage, linguistic diversity, and domain relevance.

5.1.1 Dataset generation approach

To ensure prompt diversity and natural language variation, the initial queries were generated using multiple large language models, including GPT-4, GPT-4o, and GPT-4o-mini. This multi-model prompt synthesis approach introduces stylistic and lexical variety that better simulates real-world user input.

For response consistency, all corresponding answers were generated using GPT-4o-mini. This model served as the ground truth oracle for both fine-tuning and evaluation purposes. Responses were structured to reflect the expected output behavior of the assistant, including direct answers, tool-calling syntax, or context-aware follow-ups.

This approach aligns with the principles proposed in LIMA: Less is More for Alignment [Zho23], which demonstrated that high-quality instruction-following models can be trained using relatively small but well-curated datasets. Inspired by LIMA, this thesis emphasizes dataset quality over scale, focusing on clarity, intent expression, and actionable outputs.

Further refinement of prompts and responses was guided by concepts from LLM2LLM [Lee24], which advocates using stronger teacher models to iteratively enhance training data for smaller models. In this work, multiple iterations of prompt rephrasing and response filtering were conducted to ensure that the dataset models optimal task behavior—especially in the context of tool invocation and vehicle-specific functions.

5.1.2 Dataset Structure and Categories

The dataset is categorized into six primary functionality domains commonly encountered in in-vehicle voice assistant systems:

- **General Knowledge:** Fact-based queries answerable without external APIs (e.g., "Who wrote 1984?").
- **Online API Calls:** Queries requiring real-time information retrieval (e.g., "What's the weather in Berlin today?").
- **Offline In-Vehicle Functions:**
  - **Music Control:** Commands for media playback (e.g., play, pause, skip).
  - **Navigation:** Route planning or destination guidance.
  - **Vehicle Control:** Adjustment of car settings like air conditioning or seat heating.
  - **Phone Call Operations:** Voice-controlled call initiation.

A total of 1,200 prompt-response pairs were generated across these categories.

5.1.3 System Prompts for Synthetic Data Generation

High-quality prompts were generated using structured system instructions to guide large language models (LLMs) during dataset synthesis. The following templates were used to ensure consistent query generation and tool-calling behavior.

Online Queries Prompt

You are a synthetic dataset generator for an in-vehicle assistant.
Generate 400 diverse and realistic user queries that require online API calls.
The queries could be regarding weather in any city, stock price info, sports results, flight status, etc.

Guidelines:

- Include realistic natural language variations.
- Assume vehicle position in any European or US city.
- Cover diverse use cases and sentence structures.
- Avoid duplicate queries.
- Output strictly as a Python list.

Offline Tool Queries Prompt

You are a synthetic dataset generator for an in-vehicle assistant.
Generate 400 diverse and realistic user queries that align with the capabilities of the tools listed below:
`<tools> tools </tools>`

Guidelines:

- Each query must map exactly to one provided tool.
- Use realistic variations assuming a car-based assistant.
- Avoid duplicate queries.
- Output strictly as a Python dictionary with tools as keys and queries as values.

Offline Tool Response Instructions

You are an intelligent, concise in-vehicle AI voice assistant.

Task:

- Extract and validate required parameters.
- If unclear, infer the best plausible value without asking back.
- Only use provided tools.
- Strictly follow the output format below.

```
<tool_call>[{"name": "func_name", "arguments": {"argument1":
"value1"}}]</tool_call>
```

5.1.4 Example Dataset Samples

A few representative samples from the final dataset are illustrated below:

- **Phone Call:**

```
User: "Call John Doe."
Assistant: <tool_call>[{"name": "make_call",
"arguments": {"contact": "John Doe"}}]</tool_call>
```

- **Music Control:**

```
User: "Play next song."
Assistant: <tool_call>[{"name": "music_control",
"arguments": {"action": "Play", "track": "Next"}}]</tool_call>
```

- **Navigation:**

```
User: "Navigate to Times Square."
Assistant: <tool_call>[{"name": "navigation",
"arguments": {"destination": "Times Square"}}]</tool_call>
```

- **Vehicle Control:**

```
User: "Turn on the headlights."
Assistant: <tool_call>[{"name": "vehicle_control",
"arguments": {"setting": "headlights", "target_state": 1}}]
</tool_call>
```

5.1.5 Dataset Processing and Storage

The synthetic responses were parsed into structured JSON format and subsequently saved into CSV files for training and evaluation.

Each record included:

- `prompt` – The user input.
- `response` – The model's tool-calling or answering output.

The final dataset was split into three parts:

- 80% for training
- 10% for validation
- 10% for final evaluation

Only the 80%-10% (training-validation) subset was used during fine-tuning, while the reserved 10% evaluation set was strictly held out for final model assessment.

5.1.6 Summary

The custom dataset provides a well-balanced foundation for domain-specific adaptation of SLMs. By ensuring prompt diversity, realistic context assumptions, tool-call structure enforcement, and high-quality supervision, the dataset enables effective fine-tuning of models for in-vehicle embedded assistant use-cases.

5.2 SLM Benchmarking

To identify suitable Small Language Models (SLMs) for optimization and deployment in embedded voice assistant applications, an initial set of models was selected based on recent survey studies [Wan24; Lu24; Xu24]. These studies provided a comprehensive overview and comparison of available SLMs, including model architecture, parameter size, tool integration capabilities, and deployment viability.

5.2.1 Model Selection Criteria

The following criteria were considered while shortlisting the initial SLM candidates:

- Open-source availability
- Model size below 4B parameters
- Compatibility with CPU-only inference frameworks
- Native or supported tool calling capability

5.2.2 Evaluation Methodology

To establish a benchmark, the selected models were evaluated on a custom classification task involving three key aspects:

- **Domain Understanding**: Ability to distinguish between general knowledge, online and in-vehicle functions related queries.

- **Tool Calling**: Ability to correctly recognise the parameters required to invoke functions or APIs.

- **Syntax Validity**: Response formatting and structural correctness.

Accuracy scores were computed by comparing each model's raw output to expected responses generated using GPT-4o-mini. The classification results are shown in Table 5.1.

5.2.3 Benchmark Results

Table 5.1: SLM Benchmarking Results Based on Accuracy across Task Dimensions

Provider	Model	Parameters	Tool Calling	Domain	Tool	Syntax
<1B Parameters						
Hugging Face	SmolLM2	360M	Yes	0.36	0.00	0.00
Alibaba	Qwen2.5	500M	Yes	0.42	0.60	0.00
1B – 3B Parameters						
Meta	LLaMA-3.2-1B-Instruct	1B	Yes	0.33	0.90	0.00
TinyLLaMA	TinyLLaMA-1.1B-Chat	1.1B	–	0.46	0.00	0.00
Alibaba	Qwen2.5	1.5B	Yes	0.33	0.91	0.00
Hugging Face	**SmolLM2**	**1.7B**	**Yes**	**0.58**	**0.83**	**1.00**
Google	**Gemma2**	**2B**	–	**0.70**	**1.00**	**>0**
3B - 4B Parameters						
Alibaba	**Qwen2.5**	**3B**	**Yes**	**0.62**	**1.00**	**0.73**
Meta	**LLaMA-3.2-3B-Instruct**	**3B**	**Yes**	**0.46**	**1.00**	**0.00**
Microsoft	Phi-3-mini-instruct	3.8B	–	0.70	1.00	0.13
Microsoft	**Phi-3.5-mini-instruct**	**3.8B**	–	**0.58**	**1.00**	**0.57**

5.2.4 Observations

- Models like **Gemma2 (2B)** and **Qwen2.5 (3B)** show strong tool-calling and domain understanding capabilities with relatively high syntax accuracy.

- **SmolLM2 (1.7B)** is the smallest model with a strong balance of domain and syntax performance, making it a promising candidate for further optimization.
- Models below 1B parameters show limited functionality, especially in syntax correctness and tool usage.
- Tool-calling support appears to correlate with improved accuracy across multiple dimensions, indicating its importance in real-world assistant scenarios.

Based on above observations, the models SmolLM2(1.7B), Gemma2(2B), Llama3.2(3B), Qwen2.5(3B) and Phi3.5-mini(3.8B) were selected for the finetuning stage.

5.3 Finetuning

Finetuning plays a crucial role in adapting pre-trained language models to domain-specific tasks such as embedded automotive voice assistance. By exposing models to task-relevant behaviors, finetuning enables more accurate, efficient, and structured responses—particularly when original pretraining lacks functional or domain-specific alignment.

5.3.1 Finetuning Methodology

Memory-Efficient Adaptation using QLoRA

Given the tight memory and compute budgets of embedded platforms, this thesis employs Quantized Low-Rank Adaptation (QLoRA) for finetuning. QLoRA achieves significant memory savings by combining 4-bit quantization with low-rank matrix updates, applied to a frozen pre-trained model.

The weight update formulation is:

$$\hat{W} = W + AB$$

where W is the frozen base weight, and $A \in \mathbb{R}^{d\times r}$, $B \in \mathbb{R}^{r\times k}$ are trainable low-rank matrices with $r \ll \min(d, k)$. This drastically reduces the trainable parameters and training memory footprint.

5.3.2 Tool Invocation through Special Tokens

To facilitate explicit tool calling, special tokens were introduced into the tokenizer and model vocabulary. These tokens map each tool (and special cases) to a unique, easily decodable token sequence.

```
<audi_1> – make_call
<audi_2> – music_control
<audi_3> – navigation
<audi_4> – vehicle_control
<audi_5> – general_knowledge
<audi_6> – online
<audi_end> – function call end
```

The model architecture was updated via:

```
model.resize_token_embeddings(len(tokenizer))
```

to accommodate these additional tokens in the output vocabulary.

Prompt Construction

Training samples were structured using a supervised conversational format consisting of a system message defining the assistant's behavior, a user query, and the assistant's structured response. A sample finetuning prompt is shown below:

"prompt": "<bos><start_of_turn>user
You are an intelligent, concise but helpful in-vehicle AI voice assistant designed to help users answer their query or perform the command. You are given a user prompt and a set of possible tools to use ONLY if required.
You have access to the following tools: [{function list}]
Follow these steps:
1. Analyze if online API call is needed → Output <audi_6>online<audi_end>
2. If internal knowledge suffices → Output <audi_5>your_answer<audi_end>
3. Otherwise, select the correct tool and output:
<audi_i>["name": "tool", "arguments": {...}]<audi_end>
Use ONLY provided tools, no made-up functions.
Example:
User: "Can you play the latest hits?"
Assistant: <audi_2>["name": "music_control", "arguments": {"action": "Play", "track": "latest hits"}]<audi_end>

This structured formatting improves the model's ability to precisely select and output the correct tool invocation or provide a direct answer when appropriate.

Supervised Finetuning Objective

The finetuning process is framed as a supervised learning task, where the model is adapted to maximize the conditional probability of task-specific outputs given corresponding inputs.

Given a dataset $\mathcal{C} = \{(x_i, y_i)\}_{i=1}^{N}$ of input-output pairs, the traditional supervised objective seeks to maximize the log-likelihood:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{C}} \sum_{t=1}^{|y|} \log P_{\Phi}(y_t \mid x, y_{<t})$$

where Φ denotes the full set of model parameters, and $P_{\Phi}(y_t \mid x, y_{<t})$ is the probability of generating the token y_t conditioned on the input x and the previously generated tokens.

However, directly updating all parameters Φ can be computationally expensive and storage-intensive, particularly for large pre-trained models. Therefore, this thesis adopts a parameter-efficient finetuning approach based on Quantized Low-Rank Adaptation (QLoRA).

In the QLoRA framework, the base model parameters Φ_0 are kept frozen, and only a small, low-rank update $\Delta\Phi(\Theta)$ is learned, where Θ denotes the newly introduced trainable parameters controlling the low-rank matrices. The optimization objective thus becomes:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{C}} \sum_{t=1}^{|y|} \log P_{\Phi_0+\Delta\Phi(\Theta)}(y_t \mid x, y_{<t})$$

This formulation drastically reduces the number of trainable parameters compared to full-model finetuning, while maintaining strong task adaptation performance.

For the models finetuned in this work, the loss function simplifies to the standard cross-entropy loss without any auxiliary tasks:

$$L(\mathcal{C}) = -\sum_{(x,y)\in\mathcal{C}} \sum_{t=1}^{|y|} \log P(y_t \mid x, y_{<t})$$

where the probabilities are computed using the updated model $P_{\Phi_0+\Delta\Phi(\Theta)}$.

This supervised objective aligns with the standard training regimes used in instruction-tuning and task specialization for small language models.

LoRA Hyperparameters and Training Configuration

Training was performed with QLoRA under the following hyperparameters:

- LoRA rank (r) = 16
- LoRA alpha = 32
- LoRA dropout = 0.1
- Training batch size = 1 per device; Gradient accumulation = 8
- Learning rate = 5×10^{-5} with linear decay
- Warmup steps = 10; Warmup ratio = 0.03
- Max steps = 150

Only a small subset of modules were finetuned, resulting in under 40% trainable parameter footprint compared to full-model training.

5.3.3 Azure Finetuning Environment

All finetuning runs were conducted on an Azure cloud VM with the following configuration:

CPU:

- AMD EPYC 7V12, 64-core, x86_64 architecture
- 64 threads, 2 sockets, 4 NUMA nodes
- 2445 MHz clock speed

GPU:

- 4 x NVIDIA Tesla T4 (16 GB each)
- CUDA 12.2, NVIDIA Driver 535.216.03

5.3.4 Finetuning Results

Performance was assessed along three axes: Tool Accuracy, Syntax Accuracy, and Parameter Extraction Accuracy.

Table 5.2: Finetuning Performance Comparison Across SLMs

Model	Params	Level	Tool	Syntax	Params
SmolLM2	1.7B	Base	0.66	0.99	0.33
		Finetuned	**0.72**	**1.00**	**0.41**
		Finetuned – Special tokens	0.72	0.75	0.35
Gemma2	**2B**	Base	0.68	0.36	0.00
		Finetuned	0.88	0.88	0.54
		Finetuned – Special tokens	**0.96**	**0.99**	**0.77**
Phi3-mini	**3B**	Base	0.53	0.44	0.00
		Finetuned	0.97	0.79	0.45
		Finetuned – Special tokens	**0.99**	**0.95**	**0.69**
Llama3.2	3B	Base	0.38	0.00	0.00
		Finetuned	0.98	0.24	0.19
		Finetuned – Special tokens	**0.96**	**0.51**	**0.38**
Qwen2.5	3B	Base	0.13	1.00	0.012
		Finetuned	0.88	0.94	0.62
		Finetuned – Special tokens	**0.96**	**0.99**	**0.65**

5.3.5 Training Curves

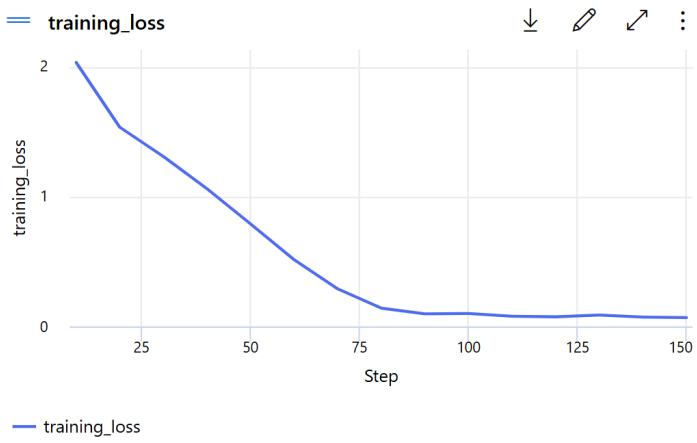

Figure 5.1: Training Loss Curve During Finetuning

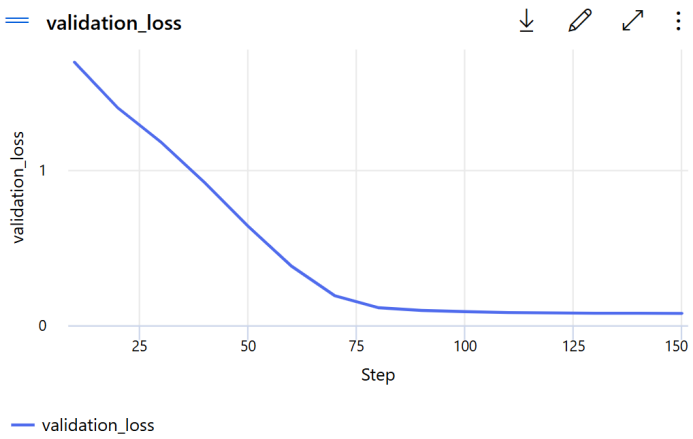

Figure 5.2: Validation Loss Curve During Finetuning

Observations:

- Rapid loss convergence after 80 steps.
- Minimal gap between training and validation loss indicating low overfitting.
- Stable optimization even with limited training steps and memory footprint.

5.3.6 Conclusion

QLoRA combined with black-box knowledge distillation significantly improves tool identification, parameter extraction, and syntax capabilities of all models. Finetuning with Special tokens shows even improved performance. These techniques allow models to achieve high accuracy in structured tool-calling tasks while maintaining a lightweight computational footprint—making them highly suitable for automotive and edge environments. Gemma2, Phi3.5-mini and Qwen2.5 moldels were selected for the next stage as they show parameter extraction higher than 85 percent.

5.4 Model Compression

Deploying large language models in embedded systems requires aggressive model compression without compromising functionality. This chapter presents two core strategies—**structured pruning** and **quantization**—used to compress models efficiently for real-time, resource-constrained deployment.

5.4.1 Structured Pruning

Structured pruning selectively removes entire architectural components—such as attention heads, MLP neurons, embedding channels, or layers—to reduce model size and latency. This strategy follows the activation-based importance scoring and joint pruning methodology proposed in Minitron [Mur24].

A forward pass over a calibration dataset collects intermediate activations using hooks. These activations are then used to compute component-wise importance scores.

Pruning Dimensions and Scoring

1. Attention Head Importance Let $\mathcal{A}_q, \mathcal{A}_k, \mathcal{A}_v$ represent the activations from the query, key, and value projections, reshaped by attention heads. The L2-norm across batch and sequence dimensions is computed per head:

$$\text{Score}_{\text{head}} = \|\mathcal{A}_q\|_2 + \|\mathcal{A}_k\|_2 + \|\mathcal{A}_v\|_2$$

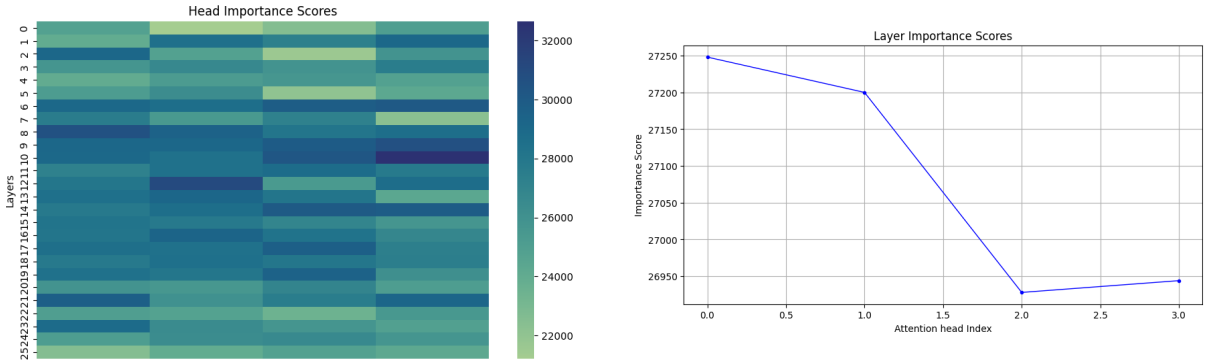

Figure 5.3: Attention Head Importance Scores (left) and Layer-wise Attention Head Importance Scores (right).

2. MLP Neuron Importance Gate projection outputs from the MLP are aggregated and normalized:

$$\text{Score}_{\text{neuron}} = \frac{1}{T} \sum_{t=1}^{T} \left\| \mathcal{A}_{\text{MLP}}^{(t)} \right\|_2$$

where T is the sequence length.

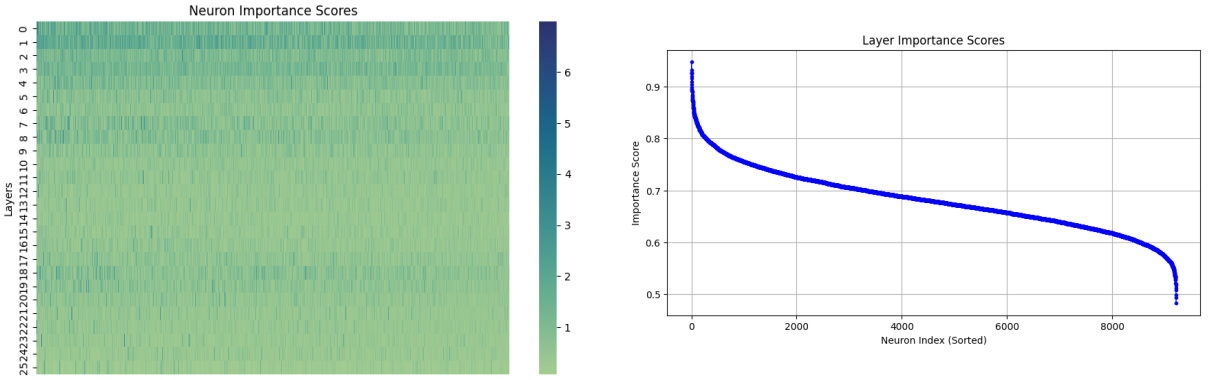

Figure 5.4: MLP Neuron Importance Scores (left) and Layer-wise MLP Neuron Importance Scores (right).

3. Embedding Dimension Importance From the LayerNorm output, per-channel importance is computed as:

$$\text{Score}_{\text{embed}} = \frac{1}{T} \sum_{t=1}^{T} \left\| \mathcal{A}_{\text{LN}}^{(t)} \right\|_2$$

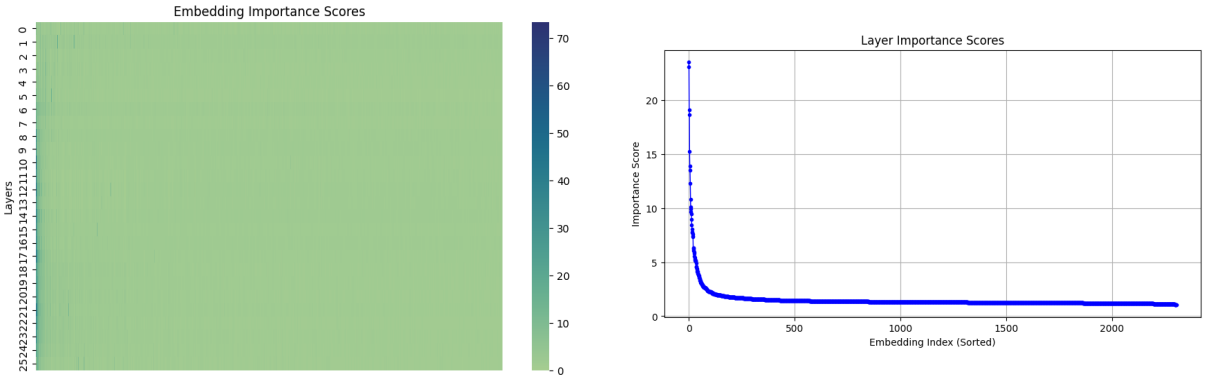

Figure 5.5: Embedding Importance Scores (left) and Layer-wise Embedding Importance Scores (right).

4. Layer Importance (Block Importance) For layer L_i, given its input X_i and output X_{i+1}, the block importance (BI) is defined as:

$$\text{BI}(L_i) = 1 - \cos(\theta(X_i, X_{i+1}))$$

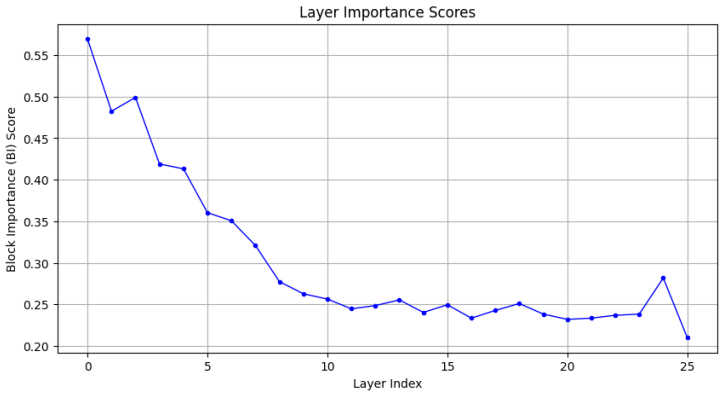

Figure 5.6: Layer Importance Scores

Pruning Challenges

While pruning offers significant compression benefits, several challenges exist:

- **Architecture Dependency:** Pruning pipelines are heavily dependent on model architecture and cannot be easily standardized.
- **Library Incompatibility:** Pruned models may break compatibility with downstream frameworks like Hugging Face Transformers, requiring manual patching.
- **Attention Mechanism Variations:** Different attention architectures (MHA, MQA, GQA) impose varying constraints, requiring specialized pruning approaches.
- **Embedding Dimension Constraints:** In many models, head dimension scaling is tied to embedding dimension, limiting how much pruning is feasible along embedding axes.

These challenges must be carefully handled when applying structured pruning in practice.

Pruning Results

The pruning outcomes for Gemma2 are summarized in Table 5.3.

Table 5.3: Pruning Configurations and Tool Call Accuracy Before and After Finetuning

MLP	MHA	Embed	Layers	Parameters (B)	Time (s)	Tokens	Tokens/s	Tool	Syntax	Params	Similarity (0.8)
-	-	-	-	2.6	3.4	50	14.7	0.68 **0.96**	0.35 **0.95**	0.0 **0.52**	0.0 **0.87**
0.1	-	-	-	2.45 (-5%)	3.3	55	16.7	0.4 **0.98**	0.6 **0.99**	0.0 **0.4**	0.0 **0.9**
0.2	-	-	-	2.28 (-12%)	3.29	55	16.7	0.4 **0.96**	0.56 0.82	0.0 0.33	0.0 0.71
0.3	-	-	-	2.12 (-18%)	3.24	55	16.9	0.2 **0.98**	- 0.77	- 0.28	- 0.7
-	0.25	-	-	2.52 (-3%)	0	0	0	- -	- -	- -	- -
-	-	0.125	-	2.29 (-12%)	2.2	36	15.0	0.38 **0.98**	0.67 **0.99**	0.0 **0.38**	0.0 **0.86**
-	-	0.25	-	1.96 (-24%)	2.0	31	15.5	0.0 0.47	- **1.0**	- 0.09	- 0.87
-	-	-	2/26	2.46 (-5%)	2.3	34	14.8	0.38 **0.94**	0.41 **1.0**	0.0 **0.38**	0.0 **0.88**
-	-	-	4/26	2.3 (-11%)	2.3	36	15.6	0.43 **0.96**	0.47 0.78	0.0 0.33	0.0 0.6
0.1	0.125	-	2/26	2.02 (-22%)	2.2	38	17.2	0.21 **0.94**	0.77 **0.99**	0.0 0.27	0.0 **0.78**

Structured pruning resulted in parameter reductions up to 22–25% with minimal loss in tool call and syntax accuracy.

5.4.2 Post-Training Quantization

Quantization compresses model weights to lower-precision formats (e.g., 8-bit, 4-bit), enabling faster inference and reduced memory consumption. Two approaches were explored:

GPTQ Quantization

GPTQ (Grouped Post-Training Quantization) minimizes reconstruction error across grouped weights:

$$\min_{\hat{W}} \sum_{g=1}^{G} \|W_g - \hat{W}_g\|_2^2$$

where W_g are original weights and $\hat{W}_g$ are quantized approximations.

Key features:

- Dynamic quantization based on calibration data.
- Supports 4-bit and 8-bit quantization.
- Optimized for CUDA-based inference.

Implementation: GPTQ quantization was implemented using the `gptqmodel` library.

GGUF Quantization for llama.cpp

GGUF is a quantized binary format designed for CPU-efficient inference through `llama.cpp`. It applies static quantization without calibration datasets.

Theoretical Background: Quantization maps continuous floating-point weights into discrete levels:

$$\hat{w} = \text{round}\left(\frac{w - \mu}{\Delta}\right) \times \Delta + \mu$$

where Δ is the quantization step size and μ is the center (zero point).

Different GGUF formats like Q4_0, Q5_1, and Q8_0 trade off between compression ratio and accuracy.

Quantization Pipeline:

1. Convert Hugging Face model to f16 GGUF datatype:

```
cd llama.cpp
python3 convert_hf_to_gguf.py ./input_dir/google/gemma-2-2b-it
--outfile ./quantized/google/gemma-2-2b-it_f16.gguf --outtype f16
```

2. Apply quantization on f16 file using `llama-quantize`:

```
./build/bin/llama-quantize ./quantize/google/gemma-2-2b-it_f16.gguf
./quantized/google/gemma-2-2b-it_q5.gguf Q5_0
```

Key features:

- Static quantization; no calibration needed.
- Highly efficient on CPUs and embedded devices.
- Supports diverse quantization granularities.

5.4.3 GPTQ vs GGUF: Comparison

Table 5.4: Comparison of GPTQ vs GGUF

Aspect	GPTQ	GGUF (llama.cpp)
Quantization Method	Dynamic (post-training, group-wise)	Static
Calibration Data	Required	Not Required
Precision	4-bit, 8-bit	Q4_0, Q5_1, Q8_0, etc.
Target Platform	GPU (CUDA)	CPU / Embedded
Integration	PyTorch, Hugging Face	llama.cpp / GGML
Optimization Goal	Minimal MSE in weight reconstruction	Efficient inference on low-end devices

Quantization Results

Table 5.5: GPTQ Quantization Results for Finetuned Gemma2 2B Model

Quantization	Parameters (B)	Memory (GB)	Time (s)	Tokens	Tokens/s	Tool	Syntax	Params	Similarity (0.8)
Base	2.6	4.65	2.47	31	**12.5**	0.97	0.92	0.50	0.86
8 bit	2.6	3.03	2.15	30	**14**	0.98	0.97	0.53	0.86
4 bit	2.6	2.08	2.09	30	**14.35**	0.98	0.97	0.53	0.87

Table 5.6: Combined Pruning, Finetuning, and Quantization Results

MLP	Embed	Layers	Params (B)	Memory (GB)	Time (s)	Tokens	Tokens/s	Tool	Syntax	Params	Similarity (0.8)
				4.57	3.3	55	16.7	0.4	0.6	0.0	0.0
0.1	-	-	2.45 (-5%)	-	-	-	-	**0.98**	**0.99**	**0.4**	**0.9**
				2.88	2.3	36	15.6	**0.6 (eos)**	**1.0**	**0.25**	**0.89**
				4.26	2.4	36	15.0	0.38	0.67	0.0	0.0
-	0.125	-	2.29 (-12%)	-	-	-	-	**0.98**	**0.99**	**0.38**	**0.86**
				2.65	2.4	36	15.0	**0.75 (eos)**	**0.99**	**0.38**	**0.91**
				4.58	3.7	66	17.8	0.38	0.41	0.0	0.0
-	-	2/26	2.46 (-5%)	-	-	-	-	**0.94**	**1.0**	**0.38**	**0.88**
				2.88	-	-	-	**0 (eos)**	**1.0**	**0.38**	**0.88**

Table 5.7: GGUF Quantization Results using `llama.cpp`

Quantization	Parameters (B)	Memory (GB)	Time (s)	Tokens	Tokens/s	Tool	Syntax	Params	Similarity (0.8)
Base	2.6	5.00	2.47	31	12.5	0.97	0.92	0.50	0.86
8 bit	2.6	2.59	0.93	32	34	0.94	0.95	0.49	0.83
5 bit	2.6	1.75	**0.82**	32	**39**	0.94	0.93	0.51	0.87
4 bit	2.6	1.52	**0.82**	**33**	**40**	**0.96**	**0.99**	0.50	**0.86**

5.4.4 Summary

- Structured pruning achieved up to 25% reduction in parameters while maintaining key functional accuracies such as tool-calling and syntax adherence.

- Post-training quantization techniques, specifically GPTQ and GGUF, offered substantial improvements in memory efficiency and inference speed.
- GPTQ provides flexible, calibration-based quantization suited for GPU inference, while GGUF is optimized for lightweight, CPU-based, on-device deployments.
- Although a combined pruning and quantization pipeline was explored, it was concluded that **structured pruning is not practical for inclusion in a standardized compression pipeline** intended for broad applicability across open-source models.
- Pruning remains highly model-specific, architecture-dependent, and may introduce compatibility issues for downstream deployment frameworks such as `llama.cpp` or Hugging Face Transformers.
- Therefore, for building a robust, generalized compression pipeline applicable to various models, **quantization-only approaches are preferred in the initial stages**. Pruning can be selectively considered for specific models where the performance gains significantly outweigh the complexity and risks introduced.

5.5 Inference On-device

To deploy language models effectively in embedded systems and edge devices such as automotive infotainment units, real-time inference performance on low-resource hardware is critical. In this study, the inference latency and throughput of finetuned and quantized models on a linux laptop is assessed, without inference engines, and using optimized runtimes such as `llama.cpp` and `ollama`.

5.5.1 Experimental Setup

On-device inference was evaluated on a local Linux notebook with the following specifications:

CPU:

- Intel Core i7-8850H, 6-core, 12-thread, 4.3 GHz max
- L1/L2/L3 cache: 192 KiB / 1.5 MiB / 9 MiB

GPU:

- NVIDIA Quadro P2000 (4 GB), CUDA 12.7

- Used for `ollama` GPU inference

5.5.2 Inference Acceleration Toolchains

llama.cpp

`llama.cpp` is a highly optimized C++ implementation for inference of transformer models on CPUs and small devices. It achieves acceleration through:

- **Quantized Kernels:** Supports GGML and GGUF formats with integer quantization (e.g., Q4_0, Q5_1).
- **SIMD Vectorization:** Uses AVX, AVX2, and AVX512 instructions for efficient matrix operations even without GPU.
- **Thread Affinity:** Allows CPU core binding (e.g., `taskset`) to optimize multi-core usage.
- **Memory Mapping:** Maps model weights directly to memory to avoid overheads of loading and unpacking.
- **No GPU Dependency:** Enables high-throughput CPU-only inference for real-time or offline usage.

Ollama

`Ollama` is a containerized runtime built for interactive LLM inference on personal machines, especially with GPUs. It integrates:

- **GPU Execution:** Leverages CUDA acceleration for transformer blocks using `llama.cpp` backend.
- **Built-in Scheduler:** Automatically selects optimal quantization and device strategy.
- **Model Registry:** Allows loading quantized models (e.g., 4-bit, 5-bit) and serving them with minimal latency.
- **Developer Friendly:** CLI and API access for rapid local inference during development.

5.5.3 Inference Results

Baseline GPU Inference Without Acceleration

Table 5.8 shows the inference time of the finetuned model when run using GPU without acceleration tools like TensorRT.

Table 5.8: Base Finetuned Model Inference (GPU without acceleration)

Quantization	Time (s)	Tokens	Tokens/s
Base Finetuned	28.25	38.25	1.35

Quantized Inference Using Ollama on GPU

Table 5.9 compares quantized model inference time on GPU using the `ollama` runtime.

Table 5.9: Quantized Inference Using `ollama` (GPU)

Quantization	Time (s)
8 bit	2.66
5 bit	**1.63**
4 bit	**1.63**

Quantized Inference Using Llama.cpp on CPU

Table 5.10 presents the results of model inference using `llama.cpp` on CPU with varying core counts.

Table 5.10: Quantized Inference Using `llama.cpp` (CPU only)

Quantization	6 CPU Cores			4 CPU Cores		
	Time (s)	Tokens	Tokens/s	Time (s)	Tokens	Tokens/s
8 bit	4.6	32.5	7.0	5.13	33.2	6.5
5 bit	3.47	33.35	9.6	4.77	33	6.9
4 bit	**2.98**	**34.3**	**11.5**	**4.3**	**33**	**7.7**

Note: CPU inference was optimized using `taskset -c 0,2,4,6` *to assign threads to non-adjacent physical cores.*

5.5.4 Discussion

- On a baseline GPU without inference engines, performance is limited (1.35 tokens/s), underscoring the importance of quantization.
- Using `ollama` on a local GPU significantly boosts throughput (up to 17x speedup) with minimal accuracy degradation.
- Quantized 4-bit models on CPU achieve up to 11.5 tokens/s using 6 threads, demonstrating feasibility for offline or low-power environments.
- 5-bit and 4-bit quantization provide a good balance of compression and runtime performance across both GPU and CPU settings.

5.5.5 Conclusion

On-device inference performance can be significantly improved using post-training quantization and efficient inference runtimes. These results validate that small language models can achieve practical speeds suitable for embedded scenarios through techniques like GGUF quantization and CPU core affinity optimization.

6 Conclusion

6.1 Summary of Contributions

This thesis explored the optimization and deployment of Small Language Models (SLMs) for embedded use-cases, particularly in the context of automotive voice assistants. It focused on three main strategies to enable real-time, resource-efficient inference: structured pruning, quantization, and parameter-efficient finetuning.

Key contributions include:

- Systematic benchmarking across model configurations, analyzing the trade-offs between compression, latency, and accuracy.
- Development of a tool-oriented finetuning pipeline using QLoRA with special token insertion and instruction-formatted data to support voice assistant behaviors.
- Implementation of component-wise structured pruning based on activation sensitivity across MLP neurons, attention heads, embedding dimensions, and transformer layers.
- Integration of two post-training quantization pipelines—GPTQ and GGUF—with detailed analysis on inference-time acceleration and memory reduction.
- Deployment of models using optimized runtimes like `llama.cpp` (CPU) and `Ollama` (GPU), enabling practical on-device usage.

6.2 Key Findings

6.2.1 Finetuning

Parameter-efficient finetuning using QLoRA showed strong improvements in tool usage accuracy and syntax correctness with minimal memory overhead. Introducing special tool tokens and updating the model head significantly enhanced structured response generation. Finetuned Gemma2-2B with special tokens achieved 0.96 tool accuracy and 0.99 syntax accuracy while updating only a fraction of the model parameters.

6.2.2 Pruning

Structured pruning of MLP layers, embeddings, and transformer depth led to significant reductions in model size and inference time. For instance, a 2.02B parameter version of Gemma2 with pruning (MLP 0.1, Emb 0.125, Layers 2) achieved a speedup of over 30% (17.2 tokens/s vs 12.5 tokens/s) while maintaining a high tool call accuracy of 0.78.

6.2.3 Quantization

Quantization was effective in reducing both memory usage and inference latency. Using GGUF 4-bit quantization, memory usage dropped from 5 GB to 2.1 GB with no drop in tool usage accuracy. With llama.cpp on CPU, 4-bit quantized Gemma2 reached 11.5 tokens/s on 6 threads. On GPU, `Ollama` inference brought inference time down to 1.63s—over 20× faster than baseline.

6.2.4 Combined Impact

A combination of special-token finetuning, and 4-bit GGUF quantization on Gemma2-2B achieved 0.86 accuracy at just 2.1 GB memory usage, while maintaining fast inference speeds:

- **CPU (Llama.cpp, 6 threads):** 11.5 tokens/sec, 2.98s
- **GPU (Ollama):** 1.63s latency
- **CPU (Llama.cpp, 4 threads):** 7.7 tokens/sec, 4.3s

Table 6.1: Final Results for Gemma2-2B-it across Compression Techniques

Pruning	Finetuning	Quantization	Inference Engine	Params (B)	Memory (GB)	Accuracy	Tokens/s	Time (s)
-	-	-	-	2.6	5.0	0.00	12.5	3.4
-	FT	-	-	2.6	5.0	0.64	12.5	3.4
-	ST	-	-	2.6	5.0	0.87	12.5	3.4
MLP 0.1, Embed 0.125, Layers 2	ST	-	-	2.02	3.9	0.78	17.2	2.2
Embed 0.125	ST	GPTQ 8-bit	-	2.29	4.4	0.68	15.0	2.4
-	ST	GGUF 4-bit	-	2.6	2.1	0.86	14.35	0.82
-	ST	GGUF 4-bit	Ollama	2.6	2.1	0.86	-	1.63
-	ST	GGUF 4-bit	Llamacpp (6 cores)	2.6	2.1	0.86	11.5	2.98
-	ST	GGUF 4-bit	Llamacpp (4 cores)	2.6	2.1	0.86	7.7	4.3

6.2.5 General Language Understanding Benchmarks

To validate the impact of quantization on general reasoning performance, we also evaluated the models on two standard benchmarks: **Winogrande** and **Hellaswag (limit 250)**. Results are shown in Table 6.2. Notably, 4-bit quantization introduced a small drop (0.68 vs. 0.7) in Winogrande performance but retained competitive levels on Hellaswag.

Table 6.2: Gemma2-2B: Benchmark Accuracy on General Language Understanding Tasks

Model	Quantization	Winogrande	Hellaswag (limit 250)
Gemma2 2B	Base	0.70	0.54
	Finetuned	0.70	0.54
	8 bit	0.69	0.50
	4 bit	0.68	0.50

6.3 Conclusion and Future Work

The experiments and analyses presented in this thesis confirm that with intelligent compression and adaptation strategies, large language models can be effectively tailored for low-resource embedded environments. Compression methods such as pruning and quantization, when paired with domain-specific finetuning, enable real-time and memory-efficient deployment of high-performing models like Gemma2.

Future directions for research and development include:

- Evaluation of latest released SLMs which boasts even better general tool calling ability.
- Deployment and evaluation of optimizes SLMs on automotive-grade SoCs.
- Continued research on pruning strategies compatible with downstream tasks and broader models.
- Extending deployment to speech-to-text and multimodal agents by combining ASR and vision inputs.

This work lays the foundation for future research in building practical, intelligent, and efficient LLM-powered agents on edge platforms.

Bibliography

[AAA24] Abdin, M.; Aneja, J.; Awadalla, H., et al.
Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone
In: arXiv preprint arXiv:2404.14219 (, 2024).

[Ali23] Alizadeh, K.; Mirzadeh, I.; Belenko, D.; Khatamifard, K.; Cho, M.; Del Mundo, C. C.; Rastegari, M.; Farajtabar, M.
LLM in a Flash: Efficient Large Language Model Inference with Limited Memory
In: arXiv preprint arXiv:2312.11514 (, 2023), DOI 10.48550/arXiv.2312.11514, https://arxiv.org/abs/2312.11514.

[BCB15] Bahdanau, D.; Cho, K.; Bengio, Y.
Neural Machine Translation by Jointly Learning to Align and Translate
In:

[Ben25] Ben Allal, L.; Lozhkov, A.; Bakouch, E.; Martín Blázquez, G.; Penedo, G.; Tunstall, L.; Marafioti, A.; Kydlíček, H.; Piqueres Lajarín, A.; Srivastav, V., et al.
SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model
In: arXiv preprint arXiv:2502.02737 (, 2025), DOI 10.48550/arXiv.2502.02737, https://arxiv.org/abs/2502.02737.

[CL24] Chen, W.; Li, Z.
Octopus v2: On-device language model for super agent
In: arXiv preprint arXiv:2404.01744 (, 2024), https://arxiv.org/abs/2404.01744.

[CV95] Cortes, C.; Vapnik, V.
Support-vector networks
In: Machine Learning, 20 (1995) 3, DOI 10.1007/BF00994018, https://link.springer.com/article/10.1007/BF00994018, pp. 273–297.

[Dee23] DeepMind, G.
Gemma: Open models for language understanding
In: arXiv preprint arXiv:2312.11805 (, 2023), https://arxiv.org/abs/2312.11805.

[Det23] Dettmers, T.; Pagnoni, A.; Holtzman, A.; Zettlemoyer, L.
QLoRA: Efficient Finetuning of Quantized LLMs
In: arXiv preprint arXiv:2305.14314 (, 2023), https://arxiv.org/abs/2305.14314.

[Dev18] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K.
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
In: arXiv preprint arXiv:1810.04805 (, 2018), https://arxiv.org/abs/1810.04805.

[Elm90] Elman, J. L.
Finding structure in time
In: Cognitive Science, 14 (1990) 2, DOI 10.1207/s15516709cog1402_1, https://doi.org/10.1207/s15516709cog1402_1, pp. 179–211.

[Erd24] Erdogan, L. E.; Lee, N.; Jha, S.; Kim, S.; Tabrizi, R.; Moon, S.; Hooper, C.; Anumanchipalli, G.; Keutzer, K.; Gholami, A.
TinyAgent: Function Calling at the Edge
In: arXiv preprint arXiv:2409.00608 (, 2024), DOI 10.48550/arXiv.2409.00608, https://arxiv.org/abs/2409.00608.

[Geo24] Georgi Gerganov and Contributors
llama.cpp: LLM Inference in C/C++

[Goo24] Google AI
Google AI Edge

[Gra24] Grattafiori, A. et al.
The Llama 3 Herd of Models
In: arXiv preprint arXiv:2407.21783 (, 2024), DOI 10.48550/arXiv.2407.21783, https://arxiv.org/abs/2407.21783.

[HS97] Hochreiter, S.; Schmidhuber, J.
Long short-term memory
In: Neural Computation, 9 (1997) 8, DOI 10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735, pp. 1735–1780.

[Hu21] Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.
LoRA: Low-Rank Adaptation of Large Language Models
In: arXiv preprint arXiv:2106.09685 (, 2021), https://arxiv.org/abs/2106.09685.

[Hua24] Huang, S.; Zhong, W.; Lu, J.; Zhu, Q.; Gao, J.; Liu, W.; Hou, Y.; Zeng, X.; Wang, Y.; Shang, L.; Jiang, X.; Xu, R.; Liu, Q.
Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios
In: arXiv preprint arXiv:2401.17167 (, 2024), https://arxiv.org/abs/2401.17167.

[HVD15] Hinton, G.; Vinyals, O.; Dean, J.
Distilling the knowledge in a neural network
In: arXiv preprint arXiv:1503.02531 (, 2015).

[Joa98] Joachims, T.
Text categorization with Support Vector Machines: Learning with many relevant features
In: Machine Learning: ECML-98 (, 1998), DOI 10.1007/BFb0026683, https://link.springer.com/chapter/10.1007/BFb0026683, pp. 137–142.

[Khi25] Khiabani, Y. S.; Atif, F.; Hsu, C.; Stahlmann, S.; Michels, T.; Kramer, S.; Heidrich, B.; Sarfraz, M. S.; Merten, J.; Tafazzoli, F.
Optimizing Small Language Models for In-Vehicle Function-Calling
In: arXiv preprint arXiv:2501.02342 (, 2025), https://arxiv.org/abs/2501.02342.

[Lam16] Lample, G.; Ballesteros, M.; Subramanian, S.; Kawakami, K.; Dyer, C.
Neural Architectures for Named Entity Recognition
In: pp. 260–270.

[Lee24] Lee, N.; Wattanawong, T.; Kim, S.; Mangalam, K.; Shen, S.; Anumanchipalli, G.; Mahoney, M. W.; Keutzer, K.; Gholami, A.
LLM2LLM: Boosting LLMs with Novel Iterative Data Enhancement
In: arXiv preprint arXiv:2403.15042 (, 2024), https://arxiv.org/abs/2403.15042.

[Liu24] Liu, Z.; Zhao, C.; Iandola, F.; Lai, C.; Tian, Y.; Fedorov, I.; Xiong, Y.; Chang, E.; Shi, Y.; Krishnamoorthi, R.; Lai, L.; Chandra, V.
MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases
In: arXiv preprint arXiv:2402.14905 (, 2024), https://arxiv.org/abs/2402.14905.

[Lu24] Lu, Z.; Li, X.; Cai, D.; Yi, R.; Liu, F.; Zhang, X.; Lane, N. D.; Xu, M.
Small Language Models: Survey, Measurements, and Insights
In: arXiv preprint arXiv:2409.15790 (, 2024), https://arxiv.org/abs/2409.15790.

[Ma24] Ma, S.; Wang, H.; Ma, L.; Wang, L.; Wang, W.; Huang, S.; Dong, L.; Wang, R.; Xue, J.; Wei, F.
The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits
In: arXiv preprint arXiv:2402.17764 (, 2024), https://arxiv.org/abs/2402.17764.

[Man24] Manduzio, G. A.; Galatolo, F. A.; Cimino, M. G. C. A.; Scilingo, E. P.; Cominelli, L.
Improving Small-Scale Large Language Models Function Calling for Reasoning Tasks
In: arXiv preprint arXiv:2410.18890 (, 2024), https://arxiv.org/abs/2410.18890.

[Mik13] Mikolov, T.; Chen, K.; Corrado, G.; Dean, J.
Efficient Estimation of Word Representations in Vector Space
In: arXiv preprint arXiv:1301.3781 (, 2013), https://arxiv.org/abs/1301.3781.

[Mur24] Muralidharan, S.; Sreenivas, S. T.; Joshi, R.; Chochowski, M.; Patwary, M.; Shoeybi, M.; Catanzaro, B.; Kautz, J.; Molchanov, P.
Compact Language Models via Pruning and Knowledge Distillation
In: arXiv preprint arXiv:2407.14679 (, 2024), https://arxiv.org/abs/2407.14679.

[Oll24] Ollama Contributors
Ollama: Run Large Language Models Locally

[Ope23] OpenAI
GPT-4 Technical Report
In: arXiv preprint arXiv:2303.08774 (, 2023), https://arxiv.org/abs/2303.08774.

[PSM14] Pennington, J.; Socher, R.; Manning, C.
GloVe: Global Vectors for Word Representation
In: pp. 1532–1543.

[PyT24] PyTorch Team
ExecuTorch Alpha: Taking LLMs and AI to the Edge with Our Community and Partners

[Qu24] Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; Wen, J.-R.
Tool Learning with Large Language Models: A Survey

In: arXiv preprint arXiv:2405.17935 (, 2024), https://arxiv.org/abs/2405.17935.

[Rad18] Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.
Improving Language Understanding by Generative Pre-Training
In: OpenAI (, 2018), https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[Ram24] Ramapuram, J.; Danieli, F.; Dhekane, E.; Weers, F.; Busbridge, D.; Ablin, P.; Likhomanenko, T.; Digani, J.; Gu, Z.; Shidani, A.; Webb, R.
Theory, Analysis, and Best Practices for Sigmoid Self-Attention
In: arXiv preprint arXiv:2409.04431 (, 2024), https://arxiv.org/abs/2409.04431.

[Sax24] Saxena, U.; Sharify, S.; Roy, K.; Wang, X.
ResQ: Mixed-Precision Quantization of Large Language Models with Low-Rank Residuals
In: arXiv preprint arXiv:2412.14363 (, 2024), https://arxiv.org/abs/2412.14363.

[Sre24] Sreenivas, S. T.; Muralidharan, S.; Joshi, R.; Chochowski, M.; Mahabaleshwarkar, A. S.; Shen, G.; Zeng, J.; Chen, Z.; Suhara, Y.; Diao, S.; Yu, C.; Chen, W.-C.; Ross, H.; Olabiyi, O.; Aithal, A.; Kuchaiev, O.; Korzekwa, D.; Molchanov, P.; Patwary, M.; Shoeybi, M.; Kautz, J.; Catanzaro, B.
LLM Pruning and Distillation in Practice: The Minitron Approach
In: arXiv preprint arXiv:2408.11796 (, 2024), https://arxiv.org/abs/2408.11796.

[SVL14] Sutskever, I.; Vinyals, O.; Le, Q. V.
Sequence to Sequence Learning with Neural Networks
In: pp. 3104–3112.

[Vas17] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I.
Attention Is All You Need
In: arXiv preprint arXiv:1706.03762 (, 2017), https://arxiv.org/abs/1706.03762.

[vLL24] vLLM Project Contributors
vLLM: A High-Throughput and Memory-Efficient LLM Inference and Serving Library

[Wal03] Wallace, R. S.
The Elements of AIML Style

[Wan24] Wang, F.; Zhang, Z.; Zhang, X.; Wu, Z.; Mo, T.; Lu, Q.; Wang, W.; Li, R.; Xu, J.; Tang, X.; He, Q.; Ma, Y.; Huang, M.; Wang, S.
A Comprehensive Survey of Small Language Models in the Era of Large Language Models: Techniques, Enhancements, Applications, Collaboration with LLMs, and Trustworthiness
In: arXiv preprint arXiv:2411.03350 (, 2024), https://arxiv.org/abs/2411.03350.

[Wei66] Weizenbaum, J.
ELIZA—a computer program for the study of natural language communication between man and machine
In: Communications of the ACM, 9 (1966) 1, DOI 10.1145/365153.365168, https://doi.org/10.1145/365153.365168, pp. 36–45.

[Xu24] Xu, J.; Li, Z.; Chen, W.; Wang, Q.; Gao, X.; Cai, Q.; Ling, Z.
On-Device Language Models: A Comprehensive Review
In: arXiv preprint arXiv:2409.00088 (, 2024), https://arxiv.org/abs/2409.00088.

[YYZ24] Yang, A.; Yang, B.; Zhang, B., et al.
Qwen2.5 Technical Report
In: arXiv preprint arXiv:2412.15115 (, 2024).

[Zho23] Zhou, C.; Liu, P.; Xu, P.; Iyer, S.; Sun, J.; Mao, Y.; Ma, X.; Efrat, A.; Yu, P.; Yu, L.; Zhang, S.; Ghosh, G.; Lewis, M.; Zettlemoyer, L.; Levy, O.
LIMA: Less Is More for Alignment
In: arXiv preprint arXiv:2305.11206 (, 2023), https://arxiv.org/abs/2305.11206.

List of Tables

List of Figures