Optimizing Small Language Models for In-Vehicle Function-Calling

Yahya Sowti Khiabani[1] **Farris Atif**[1] **Chieh Hsu**[1] **Sven Stahlmann**[2]
Tobias Michels[2] **Sebastian Kramer**[2] **Benedikt Heidrich**[2]
M. Saquib Sarfraz[2] **Julian Merten**[1] **Faezeh Tafazzoli**[1]
[1]Mercedes-Benz Research & Development North America
[2]Mercedes-Benz Tech Innovation
{yahya.sowti, farris.atif, chieh.hsu, julian.merten, faezeh.tafazzoli}
{sven.stahlmann, tobias.michels, sebastian.s.kramer}
{benedikt.heidrich, muhammad_saquib.sarfraz}@mercedes-benz.com

Abstract

We propose a holistic approach for deploying Small Language Models (SLMs) as function-calling agents within vehicles as edge devices, offering a more flexible and robust alternative to traditional rule-based systems. By leveraging SLMs, we simplify vehicle control mechanisms and enhance the user experience. Given the in-vehicle hardware constraints, we apply state-of-the-art model compression techniques, including structured pruning, healing, and quantization, ensuring that the model fits within the resource limitations while maintaining acceptable performance. Our work focuses on optimizing a representative SLM, Microsoft's Phi-3 mini, and outlines best practices for enabling embedded models, including compression, task-specific fine-tuning, and vehicle integration. We demonstrate that, despite significant reduction in model size which removes up to 2 billion parameters from the original model, our approach preserves the model's ability to handle complex in-vehicle tasks accurately and efficiently. Furthermore, by executing the model in a lightweight runtime environment, we achieve a generation speed of 11 tokens per second, making real-time, on-device inference feasible without hardware acceleration. Our results demonstrate the potential of SLMs to transform vehicle control systems, enabling more intuitive interactions between users and their vehicles for an enhanced driving experience.

1 Introduction

The rapid evolution of vehicle software has created a complex ecosystem of interconnected Electronic Control Units (ECUs) via a Controller Area Network (CAN) bus. As consumer demand for advanced features like virtual voice assistants, interior functions, and ambient settings grows, seamlessly integrating these features into existing vehicle systems becomes increasingly complex. Traditionally, each new software feature requires extensive development to interface with core vehicle systems. Here, leveraging a SLM as intermediary to streamline communication between disparate systems may facilitate easier integration of new features and adjustments based on driver conditions or external software updates.

SLMs like Gemma (2B), Microsoft Phi-3 mini (3.8B), Mistral (7B), and Llama-3 (8B) have shown strong performance on academic benchmarks, despite being significantly smaller than traditional LLMs [1]. However, due to the constraints of in-vehicle hardware, deploying these models directly may still be impractical. In this paper, we focused on optimizing these SLMs by further reducing their size and fine-tuning them to maintain performance on domain-specific tasks. We employed advanced

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

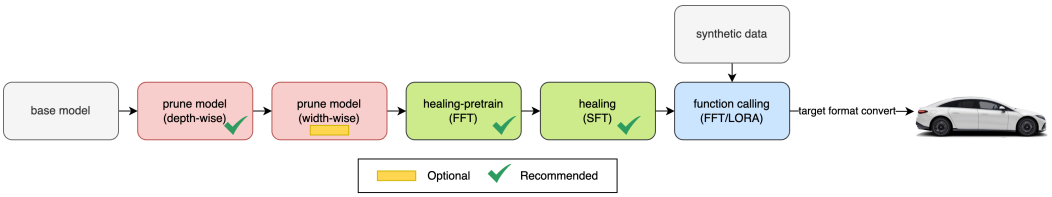

Figure 1: Proposed framework for optimizing and deploying SLM for in-vehicle function-calling. Red represents the pruning stages, green for healing, and blue for function-calling alignment.

model compression techniques such as pruning, quantization, and lightweight runtime execution. Recent studies indicate that each of these techniques results in an acceptable performance loss when used individually. However, there is limited research on the combined use of these techniques while fine-tuning a model for specific tasks, such as function-calling.

Recently, Patil et al. [2] demonstrated the enhancement of function-calling capabilities in Llama-based models [3] using a retrieval paradigm. Building on this, we extended their approach to the automotive domain. Specifically, we optimized a representative SLM to control various in-vehicle functions exposed via gRPC [4], such as seat heating, ambient lighting and more. This enables dynamic control of vehicle settings, reducing manual intervention and allowing seamless software updates.

Similar to function-calling capabilities, Gromov et al. [5] showed that small models like Phi-2 could maintain baseline accuracy on reasoning tasks. Their approach used pruned decoder heads with a healing process using parameter efficient fine-tuning techniques like Low-Rank Adaptation (LoRA) [6]. For in-vehicle deployment, we found that a more robust healing process, including full fine-tuning, is necessary to maintain acceptable performance across benchmarks and real-world tasks. Furthermore, similar to the Octopus v2 approach [7], we introduced special tokens to represent function calls, aligning our pruned and healed model with in-vehicle function-calling tasks. The introduced special tokens map the model's token space to gRPC services, enabling the model to dynamically trigger specific vehicle settings. Figure 1 presents an overview of our proposed framework.

In summary, our contributions are as follows:

1. **Model Pruning and Healing**: We applied similarity-based depth-wise pruning, an optional width-wise pruning and healing techniques to reduce the size of the Phi-3 mini model while maintaining acceptable performance across both general and domain-specific tasks.
2. **Fine-Tuning for In-Vehicle Function-Calling**: We fine-tuned the pruned and healed model using a custom dataset for in-vehicle function-calling, incorporating specialized tokens to map language model outputs to gRPC-based vehicle functions.
3. **Efficient Deployment**: We leveraged llama.cpp for model conversion and quantization, enabling efficient deployment on resource-constrained automotive hardware. This ensures that the language model can operate in real-time environments with limited computational resources.

The rest of the paper is organized as follows: Section 2 presents the related works on pruning, healing, and function-calling. Section 3 details our methodology, including model pruning, healing, and the fine-tuning process for in-vehicle function-calling as well as model conversion and deployment strategies in vehicle. Finally, before concluding in Section 5, Section 4 presents our evaluation results.

2 Related Work

Model Pruning and Healing: Model pruning reduces the size and complexity of machine learning models by removing less important weights, neurons, or entire layers [8]. The goal is to create a model with a reduced memory footprint, lower computational requirements, and faster inference. This is particularly useful for deploying models on resource-constrained devices, such as mobile phones or embedded systems like vehicle head units.

Pruning approaches can be broadly categorized into *structured pruning* and *unstructured pruning*. Structured pruning removes entire structures within the network, such as neurons, filters, or layers. In contrast, unstructured pruning removes individual weights regardless of their position in the network. While unstructured pruning can reduce the number of parameters, it often results in sparse networks, which are difficult to accelerate without custom hardware [9], making it impractical for use in environments like vehicle head units.

In structured pruning, there are two primary approaches: *depth-wise* and *width-wise* pruning. Depth-wise pruning focuses on removing entire layers from the network. For example, ShortGPT [10] calculates a Block Influence metric on a calibration dataset and removes layers with the smallest scores. Another example is the layer collapse method [11], which merges adjacent layers while ensuring that the output on a calibration dataset remains as close as possible to the original. Gromov et al. [5] compute a similarity score between activations before and after a block of several layers and prune the block with the highest similarity.

Once pruning is performed, the model can experience degraded performance due to the abrupt removal of layers. To mitigate this, healing is applied through a small amount of fine-tuning to restore the model's capabilities. As shown in [5], after pruning layers based on activation similarity, LoRA is applied to fine-tune the remaining model on a small subset of the original training data. This step ensures that the pruned model recovers its performance across tasks. For example, Gromov et al. demonstrated that up to 45% of layers, depending on the type of the model, could be pruned with minimal degradation in question-answering tasks when followed by this healing process.

Width-wise pruning aims to reduce the dimensionality of layers. For instance, SliceGPT [12] leverages the computational invariance of weight matrices to apply Principal Component Analysis (PCA) and identify rows and columns of the weight matrices that can be deleted. The MINITRON approach proposed by Muralidharan et al. [13] first records activation magnitudes for all hidden neurons and attention heads on a calibration dataset. Then, the neurons and attention heads with the lowest activation magnitude are pruned across all layers of the model.

Note that it is also possible to combine depth and width pruning strategies. For example, Muralidharan et al. [13] experimented with using their width pruning approach together with a depth pruning strategy similar to the one used by Gromov et al. [5].

Function-Calling: Function-calling is an emergent ability of language models which expands capabilities beyond text generation allowing them to interact with tools, APIs, and the physical world. Toolformer [14] showcased the ability of language models to use external tools through simple API calls, without explicit fine-tuning. Similarly, LangChain [15] provides an interface for chain of thought with various tools and data sources. Moreover, retrieval-augmented generation (RAG) techniques, such as REALM [16], have been shown to further enhance the accuracy and reliability of function-calling by leveraging external knowledge sources during inference.

As a pivotal progress in function-calling by small language models on edge devices, the Octopus v2 paper [7] introduced a novel methodology by incorporating functional tokens directly into the tokenizer, thereby streamlining the function-calling process. This approach inspired our work, where we employed specialized MB tokens, akin to the functional tokens in Octopus v2, to map language model outputs to specific vehicle functions. We also adopted a strategy similar to Octopus v2's negative sample technique, incorporating irrelevant queries into our synthetic dataset to enhance the robustness of our model against unintended function activations.

3 Methodology

We selected Phi3-mini[1], which is a decoder-only transformer language model with $L = 32$ hidden layers, as a representative of an SLM due to its small size of 3.8B parameters, its strong performance across public benchmarks, and its ability to run across various software stacks [17].

[1] `https://huggingface.co/microsoft/Phi-3-mini-4k-instruct/tree/ff07dc01615f8113924aed013115ab2abd32115b`. Note that we used the pre-July-update version of Phi3-mini.

Model	Parameters	# Layers	Hidden dim	MLP dim	Attention heads
Phi3-mini	3.8B	32	3072	8192	32
Phi3-2.8B	2.8B	24	3072	8192	32
Phi3-1.8B	1.8B	24	2688	5120	28

Table 1: Model architecture details for original phi3-mini and its pruned variants.

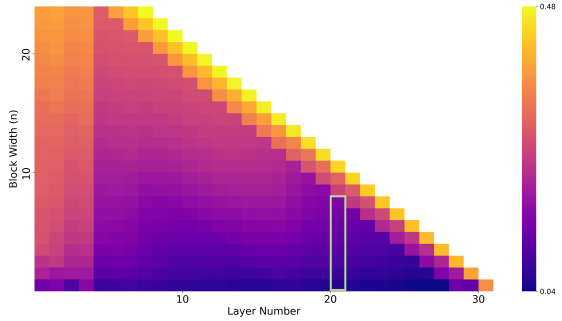

Figure 2: Heatmap of distances for all 32 decoder layers of Phi-3, with varying block sizes $n \in \{1, \ldots, 24\}$, calibrated with the fineweb dataset. Dark purple indicates regions of minimum distance or maximum similarity. Layers 21-29 (highlighted in green) were found to be the optimal block of size $n = 8$ to prune.

3.1 Model Pruning

We used both width and depth pruning in our experiments to produce two different variants of the original Phi3-mini: Phi3-2.8B and Phi3-1.8B. Table 1 contains details regarding the architecture of the two variants and the original Phi3-mini.

For depthwise pruning, we pruned a contiguous block of size $n = 8$ which minimized cumulative layer distance between decoder layers. Here we followed general guidance from Gromov et al. [5], where it was noted that dropping more than 30% of the layers across different model families (Llama, Qwen, Mistral, Phi2, etc) resulted in collapse of the underlying model. Let h_i represent the i-th hidden state of the model and n the chosen block size. Then, for all $i \in \{1, \ldots, L - n\}$, where L is the number of hidden layers in the model, we computed the angular distance between hidden states as $d(h_i, h_{i+n}) := \arccos\left(\frac{\langle h_i, h_{i+n}\rangle}{\|h_i\|\|h_{i+n}\|}\right)$. Figure 2 shows the resulting distances for different block sizes calculated against the fineweb dataset [18]. Layers 21-29 were maximally similar, and thus pruned. The resulting model is denoted as Phi3-2.8B.

Phi3-1.8B was then created by applying the width pruning approach from Muralidharan et al. [13] to Phi3-2.8B by recording activations on each layer (block) in the same manner as the depth-wise approach. Next, for the attention heads, we calculated the L2 norm along the head dimension and computed the mean across the sequence and batch dimensions for all activations. This gives us a score for each hidden neuron, each neuron in the intermediate layer of the multi-layer perceptron (MLP), and each attention head. Finally, we pruned the neurons and attention heads with the lowest score: the hidden dimension from 3072 to 2688, the MLP dimension from 8192 to 5120, and the number of attention heads from 32 to 28.

3.2 Healing and Instruction Tuning

After pruning, the resulting models struggle to generate coherent sentences and lose their alignment. As described in Gromov et al. [5], this can be remedied by applying a healing training afterwards. The authors suggest training for 5000 steps using QLoRA [19] on only the MLP weights with a diverse web-scale dataset, for which we used fineweb dataset. We denote the models produced by this step with h_{short}.

However, we determined that this was not sufficient to fully recuperate the model. While the resulting models were able to form correct and meaningful sentences again, the factual knowledge of the original Phi3-mini was almost entirely lost. Moreover, we observed that the healing loss was still decreasing. As a result, we continued training the pruned models on fineweb-edu [20] for another 45000 steps/ 15B tokens, denoting the result as h_{long}. This roughly matches the approach from Muralidharan et al. [13] who heal their pruned models for 10B tokens.

Afterwards, we instruction tuned the models for one epoch on the OpenHermes-2.5 dataset [21]. The resulting models are marked with *SFT* (Supervised Fine-Tuning). We will further discuss the importance of longer fine-tuning and instruction-tuning (Phi3-2.8B + h_{long} + *SFT*) on getting noticeable improvement on the quality of responses generated by the model in results presented in section 4.

For the Phi3-1.8B model, note that we first applied healing to the pruned Phi3-2.8B model before the width pruning step, i.e., we used Phi3-2.8B + h_{long} as the base model and then applied width pruning and instruction fine-tuning (SFT) on top of that.

3.3 Fine-tuning for In-Vehicle Function-Calling

Fine-tuning language models has become a standard practice, with various approaches being explored. Both full fine-tuning (FFT) and LoRA [22] are widely used methods, each with its own strengths and weaknesses. FFT offers comprehensive model adaptation but can be computationally expensive, while LoRA provides a more parameter-efficient alternative, particularly beneficial when GPU resources are limited. In our research, we leveraged both full model training and LoRA training, allowing us to compare their performance and understand their trade-offs. LoRA's ability to extend model functionalities further highlights its potential for adapting our framework to a broader range of applications. In addition to being more computationally efficient, the modularity of LoRA adapters opens up the possibility of seamlessly switching between different adapters, allowing for dynamic customization and adaptation of the model to various tasks or domains, as explored in works like LoRA-Switch [23]. Building on the methodology of Octopus v2 [7], we fine-tuned the pruned and healed phi3-mini model to enhance its function-calling capabilities for in-vehicle operations.

Synthetic Dataset Generation for Fine-Tuning: We generated a comprehensive synthetic dataset inspired by the Octopus v2 model's technique of integrating functional tokens into the tokenizer. Eight `MB` tokens were defined for specific vehicle functions, such as `set_ambient_light_color_program` mapped to `<MB_1>` and `set_seat_heating_intensity` mapped to `<MB_2>`. To ensure both diversity and naturalness, we utilized a multi-step prompt design for generating positive and negative examples.

Positive Examples: We used a prompt template to generate 25,000 examples evenly distributed across all vehicle functions. For example:

```
Query:
Warm up my seat and set the mood to Malibu Sunset before I get in the car
Response:
<MB_2>(seat_position="FRONT_LEFT", intensity=3);
<MB_1>(color_program="MalibuSunset");
<MB_0>(message="I've warmed up your seat and set the ambient lighting
to Malibu Sunset. Your car will be inviting when you get in.")<MB_end>
```

Negative Examples: To improve model robustness, 500 irrelevant queries were generated using a negative sampling strategy. These queries were plausible but unsolvable by the provided functions (e.g., "Can you teleport the car to Hawaii?"). The assistant responds by politely declining the request.

Quality Control: To ensure the generated dataset reflects real-life spoken user queries, we manually curated a subset of examples derived from common user questions and included them in the prompt to the LLM. We enforced an even distribution of function calls across different functions to avoid imbalance. Specific rules were added to the prompts to ensure high-quality dataset generation, and this, combined with de-duplication and post-processing, maintained a high standard for the final examples.

The dataset thus reflects natural in-vehicle commands, ensuring both accuracy in function-calling and robustness to unsupported queries. Examples of the functions vehicle services are provided in the supplementary materials.

Fine-Tuning Settings: We fine-tuned both the 2.8B and 1.8B pruned models using LoRA fine-tuning and full fine-tuning, with the specific settings outlined in Table 2. Additionally, we fine-tuned the original Phi-3 Mini model using LoRA for comparison purposes. It is important to note that for FFT, we limited the training to just one epoch and used a smaller learning rate along with a weight decay of 0.1 as a form of regularization [24]. This approach aimed to prevent overfitting to the function-calling dataset, which is a common concern with FFT due to its tendency to aggressively adapt to the training data.

In contrast, LoRA fine-tuning required at least two epochs of training without any weight decay and with a larger learning rate to achieve satisfactory results on function-calling tests. This difference in training regimes can be attributed to the nature of LoRA, which introduces a smaller set of trainable parameters compared to FFT, and that is why it may necessitate more training epochs and a higher learning rate to effectively capture the nuances of the function-calling task.

Parameter	LoRA	FFT
Rank	96	-
Alpha	16	-
Batch Size	4	4
Learning Rate	5e-5	5e-6
Number of Epochs	2	1
Weight Decay	0	0.1
Target Modules	embed_tokens, qkv_proj, o_proj, mlp (up/down), lm_head	-

Table 2: Fine-tuning settings for LoRA and FFT

3.4 Model Deployment

There are various lightweight libraries optimized for on-device inference, such as: MLX, tensorflow-lite, ONNX, and ExecuTorch [25][26][27][28]. We chose to leverage the open source on-device runtime, llama.cpp, to host our resulting pruned Phi-3 model.

llama.cpp is a wrapper around the ggml tensor library, which has native support for transformer model operations [29]. The framework makes use of the gguf file format to serialize language models and respective metadata (tokenizer, model type, quantization, etc.) into a single artifact, which is then executed against the ggml tensor library. It is flexible in its implementation and operations can be removed or composed depending on the model graph being executed.

Specifically, we took the following steps to convert our model into the target format: merge LoRA into HF base model (If LoRA is used), convert safetensors artifact to gguf, quantize resulting gguf to 4-bit, test resulting artifact, and quantify distance between gguf and original safetensors implementation.

While gguf artifacts can be quantized from 2-bit to 8-bit, we chose a 4-bit quantization strategy. In doing so, we balanced token throughput and generation with minimal added perplexity. Additionally, in this format a pruned Phi3 model uses less than 2gb of RAM.

4 Experimental Results

4.1 Evaluation Results on the Original Format

To assess the model's general language understanding, after various pruning, healing, and training steps, we used the `lm-evaluation-harness`[2] framework to evaluate models in their original safetensor format. The evaluation covered multiple benchmarks, including standard tasks such as question answering, natural language understanding, and reasoning. Specifically we evaluated our models on Winogrande [30], TruthfulQA [31], MMLU [32], HellaSwag [33], and ARC [34]. The results are outlined in Table 3.

[2] `https://github.com/EleutherAI/lm-evaluation-harness`

Model	Winogrande	TruthfulQA	MMLU	HellaSwag	ARC	Avg
Phi3-3.8B	0.74	0.36	0.70	0.59	0.54	0.59
Phi3-2.8B + h_{short}	0.69	0.34	0.65	0.47	0.42	0.51
Phi3-2.8B + h_{short} + *SFT*	0.71	0.29	0.55	0.49	0.41	0.49
Phi3-2.8B + h_{long} + *SFT*	0.68	0.27	0.56	0.51	0.46	0.50
Phi3-1.8B + h_{long} + *SFT*	0.62	0.27	0.42	0.44	0.36	0.42
Phi3-3.8B + *LoRA* (4-bit)	0.72	0.33	0.66	0.57	0.53	0.56
Phi3-2.8B + h_{long} + *SFT* + *LoRA* (4-bit)	0.67	0.25	0.50	0.48	0.41	0.46
Phi3-2.8B + h_{long} + *SFT* + *FFT* (4-bit)	0.66	0.26	0.51	0.47	0.41	0.46
Phi3-1.8B + h_{long} + *SFT* + *LoRA* (4-bit)	0.60	0.28	0.32	0.41	0.34	0.39
Phi3-1.8B + h_{long} + *SFT* + *FFT* (4-bit)	0.60	0.28	0.35	0.41	0.34	0.40

Table 3: Benchmark results across model variations

Impact of Pruning and Healing: Depth-wise pruning led to a modest decline in model performance - considering that approximately 1B parameters were removed. Moreover it is seen that longer healing yields better scores on MMLU, HellaSwag, and ARC for (Phi3-2.8B + h_{long}) vs (Phi3-2.8B + h_{short}). However, width-wise pruning on top of the 2.8B model caused significant degradation in model capabilities across all benchmarks regardless of the healing and alignment strategy applied to it. As a result, it can be inferred that regardless of the healing strategy applied, the upper limit of parameter removal is roughly 30% of the original model parameters. This falls in-line with the findings from Gromov et al. [5].

Impact of Instruction Tuning: Instruction tuning following healing improves performance in some benchmarks (e.g., Winogrande up to 0.71 for $h_{short} + SFT$), though the overall average score remains close to 0.50 for most configurations. This suggests that instruction tuning can partially recover performance in certain tasks, although the improvements are task-specific and not universal across all benchmarks.

Although the benchmark results in Table 3 indicate that longer training and instruction tuning (h_{long} + *SFT*) lead to only slight improvements in benchmark performance, direct interaction with the model revealed noticeable enhancements compared to shorter training with QLoRA (h_{short} + *SFT*). We observed that, compared to the depth-pruned 2.8B Phi3 Mini model, the combination of long healing together with instruction tuning denoted as h_{long} + *SFT*, helps recover some of the performance lost due to pruning across various datasets. This is why we selected this configuration as the baseline for the next step, which involves further fine-tuning for function-calling.

Impact of Function-Calling Fine-Tuning: We evaluated the fine-tuned models with function-calling dataset on these benchmarks after 4-bit quantization denoted as (4-bit). Notably, there was no significant performance drop when comparing the phi3-3.8B model to its LoRA fine-tuned counterpart, phi3-3.8B + *LoRA*. Similarly, across different benchmarks, the transition from h_{long} + *SFT* to h_{long} + *SFT* + (*FFT* / *LoRA*) did not result in substantial degradation for both the phi3-2.8B and phi3-1.8B models.

4.2 Evaluation on Target Format

Since the models will run with llama.cpp in vehicle, it is imperative to check performance after conversion from safetensor to 4-bit gguf. Unfortunately, the `lm-evaluation-harness` cannot interface with models in this format. So, we used llama.cpp's own evaluation tool for running standard benchmarks on the converted models. Additionally, to evaluate model performance on function-calling tasks, we used an exact match metric which measures accuracy for both the function name and arguments. Table 4 presents the results of evaluations on 4-bit gguf models.

Results on General Language Understanding: The benchmark results across TruthfulQA, MMLU, and HellaSwag reveal a clear correlation between model size and performance on general language understanding tasks. As the model size decreases from 3.8B to 2.8B to 1.8B parameters, there's a consistent decline in performance across all three benchmarks. The data indicates that reducing the model size from 3.8B to 2.8B results in a performance drop of about 4.5 to 5 points on MMLU, while further reducing to 1.8B leads to an additional drop of approximately 3 points.

Model	function-calling	TruthfulQA	MMLU	HellaSwag
Phi3-3.8B + LoRA	0.86	32.44	39.10	74.82
Phi3-2.8B + h_{long} + SFT	-	26.81	34.51	65.86
Phi3-2.8B + h_{long} + SFT + FFT	0.88	26.57	34.30	63.05
Phi3-2.8B + h_{long} + SFT + LoRA	0.88	25.70	34.15	63.93
Phi3-1.8B + h_{long} + SFT	-	26.32	31.18	55.54
Phi3-1.8B + h_{long} + SFT + FFT	0.84	25.70	30.78	54.47
Phi3-1.8B + h_{long} + SFT + LoRA	0.86	27.54	31.03	54.65

Table 4: Benchmark results across model variations (4bit gguf)

However, within models of the same size (e.g., Phi3-2.8B and Phi3-1.8B variants), the performance differences before and after fine-tuning are minimal. For example, Phi3-2.8B + SFT achieves an MMLU score of 34.51, while its fine-tuned variant, Phi3-2.8B + SFT + FFT, scores 34.30, showing little impact from function-calling fine-tuning. This suggests that the fine-tuning process does not negatively affect general language understanding and is applied effectively without diminishing the models' performance on these benchmarks.

Results on Function-Calling Performance: The fine-tuning process leads to considerable improvements in function-calling accuracy across all models. Fine-tuning the Phi3-2.8B model with LoRA achieves a function-calling accuracy of 0.88 similar to FFT score. Even in the case of the smallest model, Phi3-1.8B, function-calling accuracy remains high, with LoRA achieving 0.86 and FFT scoring 0.84. These results highlight the efficiency of fine-tuning in enhancing task-specific performance, particularly for function-calling in vehicular systems. The minor variations in function-calling accuracy among models of different sizes suggests that even smaller models, such as Phi3-1.8B, can effectively handle specialized tasks like function-calling. Considering the current production speech system for function calling, which achieves an accuracy of 0.75 across various vehicle functions, the function-calling accuracy of 0.85 or higher demonstrated by the SLM presents a promising improvement over the existing system.

It should be noted that the message in <MB_0> is not evaluated in function-calling performance since the ability for general language understanding was already evaluated. Although it degraded after fine-tuning for in-vehicle function-calling, the message remains fluent and explains what functions are activated based on human evaluation. The following example is from Phi3-2.8B $h_{long} + SFT + LoraFT$ (4-bit) which clearly depicts how a function activation addresses user needs:

```
Query:
It's stuffy here, can you do something?
Response:
<MB_4>(zone="FRONT_LEFT", fan_value=3);
<MB_0>(message="I'm increasing the fan speed to help circulate
air in your area.")
```

4.3 Token Generation Performance Gains

Table 5 summarizes the token generation results of Phi3-mini across all experiments. It is worth noting that depth-wise pruning yields a 2x increase in token-generation vs width-wise pruning. It can be inferred that removing decoder blocks altogether is more consequential in the model's ability to generate high quality responses, as well as generation speed. Additionally, the 1.8B parameter model achieves a token generation speed of 11 tokens/sec on CPU. For reference, a Llama model running on an NPU achieves the same performance [35]. Figure 3 shows the CPU usage of the 1.8B model on a vehicle head unit. At inference, the pruned model uses 400% CPU (the underlying CPU is an ARM processor with 7 cores). It can be inferred then that even a standard Phi3 which is typically regarded as a *small* language model would use all available cores during inference, further demonstrating the need for a pruning step pre-deployment. While the spike is significant, it is only sustained for the duration of inference and can be further mitigated by dynamically allocating resources to the LLM process before inference. Moreover given the magnitude of the model and applications which it can unlock, the tradeoff is reasonable. As a result we demonstrate the benefit and feasibility of running

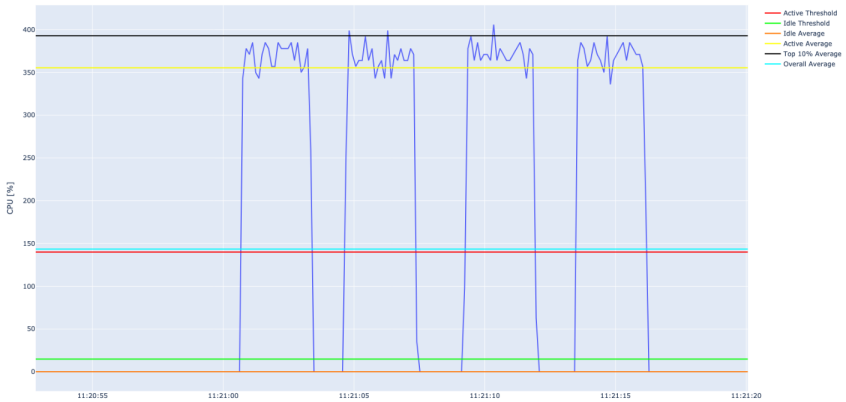

Figure 3: CPU usage of LLM process during inference on the vehicle head unit. The horizontal lines show binned values of the process across time. The *Top 10% average* (black line) shows the top 10% of CPU usage of the process.

Model	**t/s**
Phi3 (base)	6.76
Phi3 (2.8B)	9.44
Phi3 (1.8B)	11.02

Table 5: Benchmark results for 4-bit gguf model (tokens per second)

smaller language models on-device, for an out-of-distribution use case (function-calling) without any hardware acceleration.

5 Conclusion

This work demonstrates the effective optimization of Small Language Models (SLMs) for in-vehicle function-calling, delivering high task accuracy and real-time performance on resource-constrained automotive hardware. Through structured pruning, healing, and fine-tuning, we significantly reduced the size of the Phi-3 mini model while preserving its ability to handle both general language tasks and specific vehicle functions. Our method shows that pruned and quantized models can efficiently perform real-time function execution, generating up to 11 tokens per second without hardware acceleration. This offers a scalable, flexible solution for modern vehicle control systems, enabling more intuitive user interactions. Future work can focus on enhancing general language understanding and further refining these models for specific automotive tasks.

References

[1] Hugging Face. Open llm leaderboard, 2024. URL `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`. Accessed on September 6, 2024.

[2] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL `https://arxiv.org/abs/2305.15334`.

[3] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL `https://arxiv.org/abs/2302.13971`.

[4] Google. grpc: A high-performance, open-source rpc framework. `https://grpc.io/`, 2024. Accessed: [Date].

[5] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2024. URL https://arxiv.org/abs/2403.17887.

[6] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[7] Wei Chen and Zhiyuan Li. Octopus v2: On-device language model for super agent, 2024. URL https://arxiv.org/abs/2404.01744.

[8] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL http://arxiv.org/abs/1608.08710.

[10] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

[11] Yifei Yang, Zouying Cao, and Hai Zhao. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*, 2024.

[12] Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*, 2023.

[13] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation, 2024. URL https://arxiv.org/abs/2407.14679.

[14] Timo Schick, Jane Dwivedi-Yu , Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. Toolformer: Language models can teach themselves to use tools, 2023.

[15] Harrison Chase. Langchain. https://github.com/hwchase17/langchain, 2023.

[16] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1xMH1BtvB.

[17] Marah Abdin et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

[18] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

[19] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=OUIFPHEgJU.

[20] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu, May 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[21] Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023. URL https://huggingface.co/datasets/teknium/OpenHermes-2.5.

[22] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[23] Rui Kong, Qiyang Li, Xinyu Fang, Qingtian Feng, Qingfeng He, Yazhu Dong, Weijun Wang, Yuanchun Li, Linghe Kong, and Yunxin Liu. Lora-switch: Boosting the efficiency of dynamic llm adapters via system-algorithm co-design, 2024. URL https://arxiv.org/abs/2405.17741.

[24] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

[25] Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on apple silicon, 2023. URL https://github.com/ml-explore.

[26] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[27] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Executorch, 2024. URL https://github.com/pytorch/executorch.

[29] Ggerganov. llama.cpp, 2023. URL https://github.com/ggerganov/llama.cpp. Accessed on September 6, 2024.

[30] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

[31] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022. URL https://arxiv.org/abs/2109.07958.

[32] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

[33] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

[34] François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

[35] Qualcomm AI Hub. Llama v2 7b chat quantized model, 2023. URL https://aihub.qualcomm.com/models/llama_v2_7b_chat_quantized.

Supplementary Material

Vehicle function examples:

```
def set_ambient_light_color_program(color_program: str):
    """
    Set ambient light program in the car.

    Parameters:
    - colorProgram (str): Color programs options are
    ["OceanBlue", "MiamiRose", "MalibuSunset", "BurningBlue",
    "VenicePink", "ChromeShine", "RedMoon", "JungleGreen",
    "Ultramarin", "FreshCyan", "RacingYellow", "RacingRed",
    "AmethystHeat", "RoseGoldSparkle"]
    """

def set_seat_heating_intensity(seat_position: str, intensity: int):
    """
    Set seat heating intensity in the car.

    Parameters:
    - seatPosition (str): Seat position options are
    ["FRONT_LEFT", "FRONT_RIGHT", "REAR_LEFT", "REAR_RIGHT"]
    - intensity (str): Intensity options are [0, 1, 2, 3]
    """

def set_temperature(zone: str, temperature: float, unit: str):
    """
    Set temperature in the car.

    Parameters:
    - zone (str): Zone options are
    ["FRONT_LEFT", "FRONT_RIGHT", "REAR_LEFT", "REAR_RIGHT"]
    - temperature (float): Temperature range is
    from 60 to 84 for FAHRENHEIT and 16 to 28 for CELSIUS
    - unit (str): Unit options are ["CELSIUS", "FAHRENHEIT"]
    """

def set_window_position(window_position: str, operation: str):
    """
    Set window position in the car.

    Parameters:
    - windowPosition (str): Window position options are
    ["FRONT_LEFT", "FRONT_RIGHT", "REAR_LEFT", "REAR_RIGHT"]
    - operation (str): Operation options are ["OPEN", "CLOSE"]
    """

def respond_chat(message: str):
    """
    Respond to the user's query, for example to
    provide an answer or ask for more information.

    Parameters:
    - message (str): The message that should be returned to the user.
    """
```