Tiny Voice: A Library For Fine-tuning Context-Specific ASR Models on Edge CPU

https://github.com/derpysquid10/tiny_voice

Gabriel Wu
University of Toronto
yihuigabriel.wu@mail.utoronto.ca

Ronaldo Luo
University of Toronto
ronaldo.luo@mail.utoronto.ca

Gordon Tan
University of Toronto
gordon.tan@mail.utoronto.ca

Abstract

Fine-tuning automatic speech recognition (ASR) models enables domain adaptation and personalization, but remains computationally expensive, especially on resource-constrained hardware. Existing solutions either rely on cloud-based fine-tuning–introducing privacy and usability concerns–or require powerful GPUs for full model updates. In this report, we present `Tiny Voice`, a lightweight library for efficient CPU-based fine-tuning of Whisper models. The library supports three parameter-efficient fine-tuning methods (partial fine-tuning, LoRA, and IA3) and integrates system-level optimizations such as gradient checkpointing. We conduct a comprehensive evaluation across various dataset sizes, learning rate configurations, and hardware settings. Our experiments show that LoRA achieves competitive transcription accuracy with only 0.2% of the trainable parameters, while also demonstrating robustness to hyper-parameters and memory efficiency. We also outline next steps toward building high-performance, hardware-native ASR training pipelines for edge devices.

1 Introduction

Modern automatic speech recognition (ASR) models perform well in general speech recognition tasks but often struggle to adapt to specific contexts, such as user-specific accents, expression styles, or domain-specific idioms and acronyms. Customizing these models through fine-tuning can significantly improve accuracy and enhance the user experience. However, most individual users and small businesses have limited computational resources. While deploying large models locally is feasible, fine-tuning them remains a major challenge due to its high computational cost (1). Fine-tuning also requires substantial memory: for full fine-tuning of large transformer models in half precision, approximately 16 GB of GPU memory is needed per billion parameters (2). This poses a significant bottleneck, given the memory constraints of consumer-grade laptops.

As shown in Figure 1, the most common current solutions involve either relying on the model provider to update models with newly collected user data or using cloud services for fine-tuning. The first approach raises privacy concerns and limits user-specific customization. The second requires users to manage cloud-based fine-tuning themselves, which demands technical expertise, complicates automation, and also introduces privacy risks. Therefore, a method for performing local fine-tuning automatically would address this problem, enabling individuals and small businesses to benefit from model customization while preserving data privacy.

Preprint. Under review.

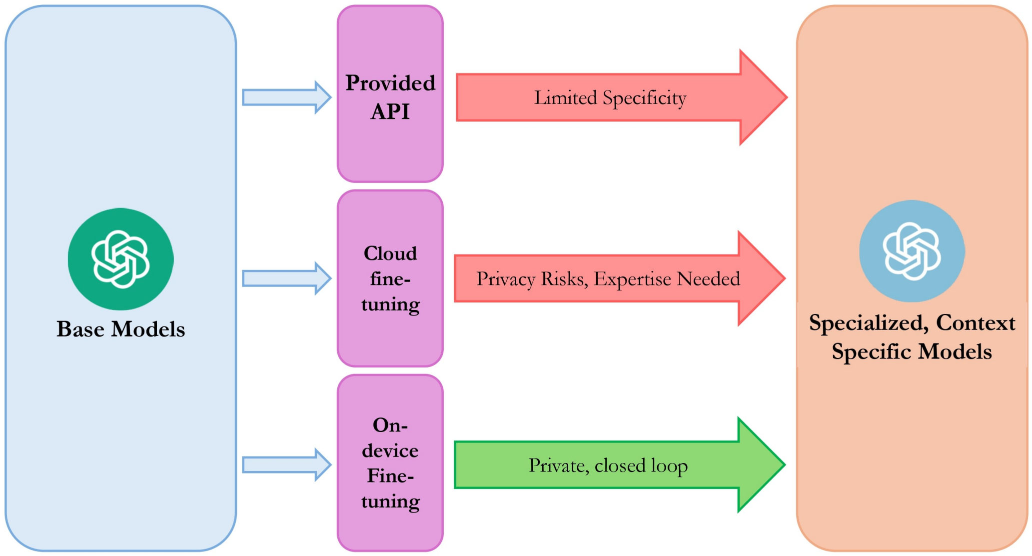


Figure 1: Current fine-tuning methods with their challenges and concerns.

In this project, we present the following contributions. First, we developed `Tiny Voice`, an easy-to-use software library that integrates three efficient fine-tuning techniques for faster training on edge CPUs: partial fine-tuning, low-rank adapters (LoRA), and infused adapters (IA3). The library also includes built-in system optimizations, such as gradient checkpointing, which further reduces memory usage. Second, we demonstrate the effectiveness of this library by using it to fine-tune Whisper Base(3)—a widely used ASR model by OpenAI—for speech-to-text transcription on the Afrispeech-200 dataset(4). This dataset contains 200 hours of everyday and medically related conversations from speakers with diverse English accents across Africa. Finally, we conducted comprehensive experiments to analyze the training dynamics of these three efficient fine-tuning methods, evaluating system-level performance, generalization across different dataset sizes, and robustness to various hyperparameters.

The report is structured as follows. Section 2 reviews related work on different fine-tuning methods, with a focus on parameter efficiency, memory efficiency, and hardware-specific optimizations. Section 3 outlines the architecture of our `Tiny Voice` library and its user-level APIs. Section 4 provides an overview of the Whisper Base model and the Afrispeech-200 dataset. Section 5 defines the evaluation metrics used to assess fine-tuning performance. Section 6 presents the fine-tuning results, along with extended experiments to analyze the training dynamics of the fine-tuning methods. Finally, Section 7 summarizes our findings and discusses future directions.

2 Related Works

Model fine-tuning efficiency can be optimized along three key dimensions: 1) parameter-efficient fine-tuning, which reduces the number of trainable parameters while maintaining model performance; 2) memory-efficient fine-tuning, which lowers memory usage by reducing precision or trading memory storage for recomputation; and 3) hardware-specific kernel optimizations, which leverage specialized implementations to accelerate fine-tuning on specific hardware platforms.

2.1 Parameter-efficient Fine-tuning

When fine-tuning large transformers, updating all parameters can be computationally expensive and memory-intensive. Parameter-efficient fine-tuning (PEFT) methods aim to adapt the model with minimal parameter updates while retaining its expressiveness. There are several notable techniques.

Partial Fine-tuning (5): Instead of fine-tuning the entire model, only the last layer (or a small subset of layers) is updated. The transformer model is treated as a fixed feature extractor. This technique assumes that the higher layers of a pre-trained transformer encode task-relevant features and that minimal adaptation is required. Only the task-specific classifier (e.g., the final linear layer) is fine-tuned.

Low-rank Adaptations (LoRA) (6): Low-Rank Adaptation is a method that freezes the original model weights and injects trainable low-rank matrices into the attention layers. For a pre-trained weight matrix, $W \in \mathbb{R}^{d\times k}$, W is updated by the weight update matrix $\Delta W = AB$, where $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$ are the low-rank matrices of LoRA. Here, the intrinsic rank, $r \ll d, k$ to satisfy the low-rank property. During training, only A and B are fine-tuned, while the weights in W are frozen. In high dimensional models, this allows for a significant reduction in trainable parameters. During inference, W and ΔW are combined, incurring no additional inference overhead.

Infused Adapter by Inhibiting and Amplifying Inner Activations (IA3) (7): IA3 is a fine-tuning mechanism that learned the additional rescaling vectors of the keys, values and intermediate activation of feed-forward networks for every transformer block in the transformer model. Only those additional vectors are fine-tuned and others are frozen, which significantly reduced the number of trainable parameters while enabling adaptation to context.

2.2 Memory-efficient Fine-tuning

To reduce memory usage during fine-tuning, two common strategies are employed: quantization, which reduces the precision of computations and storage, and gradient checkpointing, which trades memory consumption for training speed by recomputing intermediate activations on the fly.

Quantization-Aware Training (QAT)(8): Quantization is a technique that maps floating point weights to low-bitwidth integers (INT8 or INT4) through a linear mapping. Quantization-Aware Training simulates lower-bit integer precision during training while keeping computations in floating-point during backpropagation. This allows the model to learn robust quantized weights, leading to better performance after deployment. Unlike Post-Training Quantization (PTQ), which applies quantization after training, QAT fine-tunes the model with quantization effects in mind.

Gradient checkpointing (9): This technique trades computation for memory efficiency. In standard training, intermediate activations from all layers are stored to enable gradient computation during the backward pass. In contrast, gradient checkpointing selectively stores activations from only $O(\sqrt{n})$ layers in an n-layer network. During backpropagation, the remaining activations are recomputed as needed. This reduces the memory requirement from $O(n)$ to $O(\sqrt{n})$, at the cost of performing additional forward passes during training (9).

2.3 Hardware-Specific Kernel Optimization

Fine-tuning large transformers on edge devices requires specialized kernel optimizations to maximize efficiency. Since edge devices have constrained power and memory, optimizing computations at the kernel level is crucial.

Intel IPEX (Extension for PyTorch)(10): This library provides Intel-optimized PyTorch kernels for both inference and fine-tuning. It includes weight prepacking for efficient memory access and supports BF16, INT8, and FP32 auto-optimizations. It also uses graph-level optimizations such as operation fusion.

3 The Tiny Voice Library

As illustrated in Figure 2, the `Tiny Voice` library provides a modular and user-friendly interface for fine-tuning ASR models. The library is optimized for CPU-based training workflows and integrates gradient checkpointing.

Tiny Voice centers around four core functions that together compose a complete fine-tuning pipeline:

`data_pipeline(dataset)`: Loads the specified dataset from disk at the expected processed directory. The dataset should contain `.wav` audio files paired with their corresponding text transcrip-

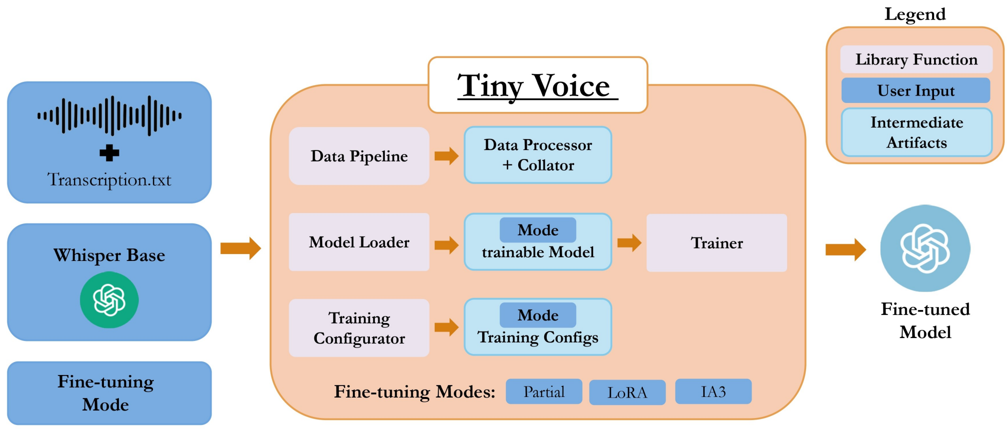


Figure 2: Architecture of the Tiny Voice library.

tions[1]. This function returns a `DatasetDict` with training, validation, and test splits, as well as a `WhisperProcessor` object for feature extraction and tokenization.

`load_model(peft)`: Loads a pre-trained Whisper model and configures it for one of the supported parameter-efficient fine-tuning methods: `"partial"`, `"lora"`, or `"ia3"`. Each strategy wraps or modifies the base model differently to reduce the number of trainable parameters.

`setup_training_args(peft)`: Instantiates a `Seq2SeqTrainingArguments` object tailored to the chosen fine-tuning mode. This includes hyperparameters such as learning rate, training schedule, batch size, and evaluation frequency. It also invokes gradient checkpointing.

`train_model(model, data, processor, peft, training_args)`: Coordinates the complete training process, including data collation, forward/backward passes, evaluation on held-out data, and model checkpointing. If the IA3 method is selected, a custom trainer is invoked to handle the adapter-specific forward and loss computations.

Internally, the library leverages Hugging Face's `transformers` ecosystem, including its Trainer API and dataset abstractions. Each module in the Tiny Voice pipeline is designed to be lightweight and hides most of the underlying configuration from the user, making it easy to use out of the box. Despite this simplicity, the library remains highly performant: we conducted extensive experiments to identify the optimal configurations for each fine-tuning method, ensuring strong and generalizable empirical results across different scenarios.

4 Model and Data

The model fine-tuned in this paper is OpenAI's Whisper Base model (3). This transformer encoder-decoder model is relatively small, with 74 million parameters, allowing us to efficiently experiment with and evaluate various fine-tuning techniques.

The dataset used for fine-tuning is AfriSpeech-200(4). It comprises 200 hours of Pan-African speech for clinical and general-domain English-accented ASR, covering 120 African accents from 13 countries and featuring 2,463 unique African speakers(11). The full dataset is 200 GB in size. To ensure tractability, we focus on the subset with the IsiZulu accent from South Africa, which includes 1,048 audio clips from 48 speakers, totaling 2.88 hours of speech data.

This subset is divided into 775 training samples, 169 validation samples, and 100 test samples, with each sample being a 30-second WAV file. Figure 3 in Appendix A illustrates the age, domain, and gender distributions within the dataset. The age distribution is primarily concentrated between 19 and 40 years old. The domain is fairly evenly split between clinical and general speech, while the

[1]We currently only support Hugging Face-style datasets, but the code can be easily extended to accept arbitrary user-curated data.

gender distribution is predominantly female. Transcriptions of an example training, validation, and testing clip are provided in Figure 6 in Appendix A.

5 Metrics

Given the context of the problem, there are 2 dimensions of metrics to evaluate our fine-tuning strategy: 1) fine-tuning performance, focusing on the degree to which strategy improves the model; and 2) System and hardware performance, which reflects how well the strategy accommodates hardware constraints.

5.1 System and Hardware Performance

The nature of fine-tuning on edge devices imposes many hardware constraints and challenges that GPU fine-tuning does not frequently encounter. The following metrics are used to account for these challenges.

Training Throughput: The efficiency of training is imperative given the significant computational constraint of a CPU. Therefore, the metric of interest here is the training throughput, measured as the *average number of samples processed per second*. This metric varies depending on the capabilities of the computer used, making universal benchmarks less relevant. However, within this paper, where all data are processed on the same system, it allows for meaningful internal comparisons of different fine-tuning approaches.

Peak Memory Utilization: Edge devices have limited RAM, and a fine-tuning strategy must be able to operate under these constraints. Since the target audience of out fine-tuning strategy is consumer individuals, it is expected that the devices they will be equipped with will not exceed 8 GB to 16 GB of RAM. Thus, a constraint for peak memory utilization will be 8 GB.

CPU Utilization: Fine-tuning is a heavy task that is power-intensive. CPUs are more prone to overheating when performing highly parallel, multi-core operations. Thus, monitoring the temperature of a CPU is imperative. As the CPUs' temperature varies from individual device, fans and placement location, the more comparable and stable metric *% CPU utilization* (average percentage across all cores) is used as its proxy since CPUs with more average utilization typically dissipates more heat and have higher temperature.

5.2 Fine-tuning Performance

Word Error Rate (WER): The transcription accuracy of the model is evaluated using Word Error Rate (WER), a standard metric in speech recognition. It is computed as:

$$\text{WER} = \frac{\text{Substitutions} + \text{Insertions} + \text{Deletions}}{\text{Number of Words Spoken}}.$$

Substitutions occur when a spoken word is replaced incorrectly in the transcription; insertions refer to words added in the transcription that were not spoken; deletions represent spoken words that were omitted from the transcription.

6 Experiment and analysis

6.1 Experimental Setup

All experiments were first conducted on an Ubuntu machine equipped with an Intel Core i7 (12th Gen) CPU and 32 GB of RAM and later verified by a device with only 16 GB of RAM. Unless otherwise specified, we use the `IziZulu` dataset for fine-tuning and evaluation. Since we do not need to perform extensive hyper-parameter tuning, we discard the validation set and directly evaluate on the test set. Gradient checkpointing is enabled in all runs to reduce memory usage, while Intel's PyTorch Extension (IPEX) is disabled due to its lack of benefit on non-Xeon CPUs. See Appendix B for more detail on gradient checkpointing and IPEX.

We define two baselines to contextualize results: the *no fine-tuning* baseline represents the pre-trained Whisper model without any adaptation, which all methods are expected to outperform. The *full* *fine-tuning* baseline serves as the upper bound for performance, and parameter-efficient methods should aim to match it as closely as possible.

For partial fine-tuning, we update only the final feedforward layers (two fully connected layers and the layer normalization) in both the encoder and decoder. This setup was found to be more effective than tuning only the final encoder or decoder layer in isolation. For LoRA, we set the rank to 4, as this is the lowest value that does not lead to performance degradation. LoRA updates the query and value projection weights in each attention block. For IA3, we update the query, key, and value projections, as well as all fully connected layers in the transformer.

Training is performed for a maximum of 200 steps. The learning rate and scheduling strategy are tailored to each method: partial fine-tuning and LoRA use a learning rate of 1×10^{-3}, while IA3 uses 2×10^{-3}. Warm-up steps are set to 20 for partial fine-tuning and IA3, and 0 for LoRA. A linear scheduler is used for partial fine-tuning, while LoRA and IA3 both use cosine decay. We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e - 8$. We used Cross Entropy as the training objective and Word Error Rate (WER) as the evaluation metric.

We use the Weights & Biases (WandB) logger to track both transcription accuracy and system-level metrics such as memory usage, CPU utilization, and throughput.

6.2 Transcription Accuracy

Table 1: WER comparison across different fine-tuning methods.

Method	**Tunable Parameters** (↓)	**WER (Overall)** (%, ↓)	**WER (General)** (%, ↓)	**WER (Clinical)** (%, ↓)
No finetuning	–	42.21	43.04 (+0.83)	41.58 (–0.63)
Full finetuning	72M (100%)	27.61	27.42 (–0.19)	**27.75 (+0.14)**
Partial finetuning	4.2M (5.8%)	36.10	33.61 (–2.49)	38.01 (+1.91)
LoRA	147K (0.2%)	**27.55**	**27.00 (–0.55)**	27.97 (+0.42)
IA3	**50K (0.07%)**	38.36	39.24 (+0.88)	37.69 (–0.67)

The transcription accuracy for different fine-tuning methods is summarized in Table 1. We evaluate model performance on the full test set, as well as separately on general-domain and clinical-domain subsets, which has similar distributions across both training and test set (see appendix A). The last two columns of the table report the WER on domain-specific data, along with the accuracy difference relative to the overall WER in parentheses.

All three fine-tuning methods outperform the pre-trained model. Notably, LoRA, with only 0.2% of the parameters used in full fine-tuning, achieves even better accuracy than full fine-tuning. In contrast, IA3, while being more lightweight at 0.07% of the total parameters, results in a significant degradation in WER (38.36), indicating a potential trade-off between parameter efficiency and performance.

Partial fine-tuning yields moderate performance with a WER of 36.10. However, it exhibits a notable bias toward general-domain data, achieving 2.49% lower WER than its overall average, while performing worse on clinical speech. In comparison, LoRA maintains consistent WER across both general and clinical domains, suggesting better robustness and generalization.

6.3 System-level Performance

Table 2 reports system-level performance metrics for each fine-tuning method, including training throughput, memory usage, and CPU utilization. Note that full fine-tuning requires 8.8 GB of system memory, which is not always available on consumer-grade laptops. Among all methods, partial fine-tuning achieves the highest training speed (1.7 samples/sec) and lowest memory usage (4.4 GB). However, it comes with relatively high CPU utilization (47.6%), which may pose challenges in thermally limited devices. In contrast, both LoRA and IA3 exhibit significantly lower CPU utilization, with IA3 consuming the least at 34.1%.

Table 2: System-level performance metrics for different fine-tuning methods on CPU.

Method	**Training Throughput** (samples/sec, ↑)	**Memory Usage** (GB, ↓)	**CPU Utilization** (%, ↓)
Full fine-tuning	0.85	8.76	49.23
Partial fine-tuning	**1.72**	**4.36**	47.60
LoRA	0.84	5.88	36.92
IA3	0.90	5.45	**34.11**

Before running the experiments, we hypothesized that reducing the number of tunable parameters would lead to higher throughput, lower memory usage, and reduced CPU utilization. However, the results in Table 2 do not fully support this assumption. In particular, although LoRA and IA3 require significantly fewer trainable parameters than partial fine-tuning, they both exhibit lower throughput and higher memory usage.

A reasonable explanation for this discrepancy lies in the depth of gradient propagation required by each method. Partial fine-tuning only updates the final fully connected layers of the model, meaning that gradients do not need to be propagated back through the entire network. This substantially reduces the number of backward operations and eliminates the need to cache intermediate activations, resulting in faster training and lower memory consumption. In contrast, both LoRA and IA3 inject learnable parameters into earlier layers of the model, such as the attention modules. As a result, backpropagation must traverse the full model depth, leading to increased computational and memory demands despite the smaller number of tunable parameters.

6.4 Adaptability to Different Dataset Size

We are interested in exploring how well the fine-tuning methods generalize under varying amounts of training data. The primary dataset used in our experiments, `IziZulu`, contains approximately 3 hours of speech data. To further evaluate adaptability, we test all three methods on two additional subsets from AfriSpeech-200: the Swahili accent and the Isixhosa accent (4). The Swahili dataset is a high-resource dataset contains 15 hours of data (about 5 times of IsiZulu dataset) and the Isixhosa dataset is a low-resource dataset with only 35 minutes of data (about 5 times smaller than the IsiZulu dataset). Their demographic distributions are also in Appendix A, which are generally similar to the distributions for the IsiZulu accent.

Table 3: WER (%) across datasets of varying sizes.

Method	**WER (Low-Resource)** (Isixhosa, 35 min)	**WER (Mid-Resource)** (IziZulu, 3 hrs)	**WER (High-Resource)** (Swahili, 15 hrs)
No fine-tuning	42.91	42.21	46.07
Full fine-tuning	30.50	27.61	**28.32**
Partial fine-tuning	36.52	36.10	36.92
LoRA	**28.37**	**27.55**	28.66
IA3	40.43	38.36	41.29

Table 3 shows how each fine-tuning method performs across datasets of different sizes. The IziZulu results are repeated from earlier experiments for reference. Models were trained for 5 epochs (100 iterations) in the low-resource regime and 1 epoch (600 iterations) in the high-resource regime. LoRA consistently demonstrates strong performance across all three regimes, outperforming full fine-tuning in the low-resource setting and closely matching it in both mid- and high-resource scenarios.

Full fine-tuning performs well in the mid- and high-resource settings, but its effectiveness diminishes in the low-resource regime. This aligns with expectations, as tuning a large number of parameters generally requires more training data to achieve robustness and generalization. In contrast, IA3 shows consistently weaker performance in both low- and high-resource conditions. This is somewhat surprising, as we would expect a more parameter-efficient method to perform better when training data is limited.

6.5 Robustness to Different Hyper-parameters

When fine-tuning on CPUs, computational constraints often make exhaustive hyper-parameter tuning infeasible. Therefore, a good fine-tuning method for CPU deployment should exhibit robustness to hyper-parameter variations and ideally perform well with default settings—in other words, it should "work out of the box." The most impactful hyper-parameters for fine-tuning performance are the learning rate, learning rate warm-up steps, and learning rate decay schedule.

To evaluate the robustness of each method to these factors, we first fixed the learning rate warm-up steps to 20 and used a cosine annealing scheduler. We then sampled seven learning rates logarithmically spaced between 10^{-5} and 10^{-2}: `[1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2]`. We define a learning rate as *convergent* if the resulting fine-tuned model achieves a lower WER than the pre-trained model baseline—i.e., if fine-tuning yields actual improvements.

To further test robustness against learning rate warm-up and scheduling strategies, we conducted an ablation study by disabling each component independently and observing the change in WER. A degradation greater than 2% was considered a significant sensitivity to that component. The result is summarized in table 4.

Table 4: Robustness to learning rate schedule and warm-up.

Method	**LR Convergence Range**	**Best LR**	**WER with Best LR**	**WER w/o Warm-up**	**WER w/o Cosine Decay**	**Warm-up / Decay Helps?**
Partial finetuning	[1e-5, 1e-3]	1e-3	36.10	42.94 (+6.84%)	54.12 (+18.02%)	Yes / Yes
LoRA	[5e-4, 1e-3]	1e-3	27.55	28.16 (+0.61%)	28.58 (+1.03%)	No / No
IA3	[1e-3, 5e-3]	2e-3	38.36	41.42 (+3.06%)	41.97 (+3.61%)	Yes / Yes

Among the methods, LoRA demonstrates the highest robustness: it converges over a relatively narrow but practical LR range and shows minimal performance degradation when warm-up or cosine decay is removed. In contrast, partial fine-tuning is highly sensitive to both warm-up and decay. Without warm-up, its WER worsens by over 6.8%, and without cosine decay, performance degrades significantly by over 18%. IA3 falls somewhere in between: it is more robust than partial fine-tuning, but still experiences moderate sensitivity to both warm-up and decay, with over 3% degradation in each case. These results are consistent with our empirical experience: achieving competitive performance with partial fine-tuning and IA3 required significantly more hyper-parameter tuning, whereas LoRA typically performed well with minimal tuning effort.

7 Conclusion

In this work, we presented `Tiny Voice`, a lightweight and CPU-friendly library for fine-tuning Whisper ASR models. First, we implemented three efficient fine-tuning methods—partial fine-tuning, LoRA, and IA3—within a unified and user-friendly API. Second, we demonstrate the effectiveness of our library by fine-tuning the Whisper Model on the Afrispeech dataset. Third, we conducted a comprehensive empirical evaluation across a variety of settings to study the training dynamics, generalization, and system-level behavior of each method.

Our experiments show that LoRA consistently offers the best trade-off between accuracy, memory usage, and robustness. Partial fine-tuning, while computationally efficient, is highly sensitive to hyper-parameter settings. IA3 shows more robustness than partial fine-tuning but lags behind LoRA in overall performance.

Looking ahead, we plan to extend this work in several directions. First, we aim to directly implement high-performance fine-tuning kernels optimized for Intel Core CPUs. Second, we will expand the library to support a wider range of fine-tuning techniques. Third, we plan to implement hardware-native kernels for quantization-aware training, which is currently unsupported on Intel Core processors. Finally, we aim to develop a more flexible user interface to support diverse use cases in low-resource environments.

8 Gen AI Use

GPT4o was used to revise this report. Specifically, it was used to correct grammar and wording in sentences. The prompt used was "I need you to help me revise my report draft. For each paragraph, correct any grammar mistakes. For any awkward sentences, make it sound natural". For code, Github Copilot was used for auto-completion in certain scenarios.

References

[1] Y. Dong, H. Zhang, C. Li, S. Guo, V. C. M. Leung, and X. Hu, "Fine-tuning and deploying large language models over edges: Issues and approaches," 2024. [Online]. Available: https://arxiv.org/abs/2408.10691

[2] T. Kim, Y. Wang, V. Chaturvedi, L. Gupta, S. Kim, Y. Kwon, and S. Ha, "Llmem: estimating gpu memory usage for fine-tuning pre-trained llms," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '24, 2024. [Online]. Available: https://doi.org/10.24963/ijcai.2024/699

[3] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022. [Online]. Available: https://arxiv.org/abs/2212.04356

[4] T. Olatunji, T. Afonja, A. Yadavalli, C. C. Emezue, S. Singh, B. F. P. Dossou, J. Osuchukwu, S. Osei, A. L. Tonja, N. Etori, and C. Mbataku, "AfriSpeech-200: Pan-African accented speech dataset for clinical and general domain ASR," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1669–1685, 2023. [Online]. Available: https://aclanthology.org/2023.tacl-1.93/

[5] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018. [Online]. Available: https://arxiv.org/abs/1801.06146

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[7] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[8] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, and P. Luo, "Efficientqat: Efficient quantization-aware training for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2407.11062

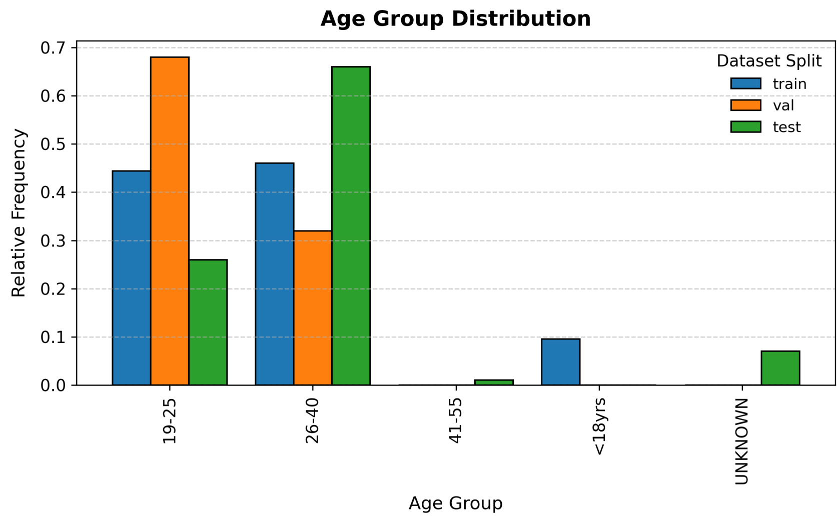
[9] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost, 2016," *arXiv preprint arXiv:1604.06174*, 2016.

[10] I. Corporation, *Accelerate with Intel Extension for PyTorch*, 2024, accessed: 2025-03-10. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/accelerate-with-intel-extension-for-pytorch.html

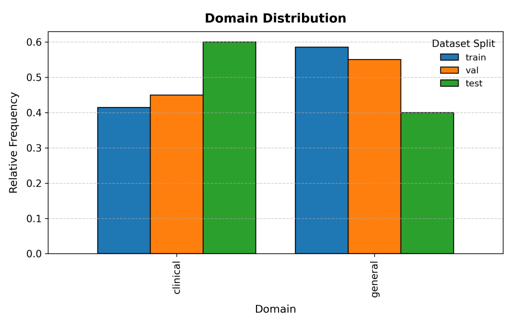
[11] I. Health, "Afrispeech-200 dataset," 2024, accessed: 2025-03-10. [Online]. Available: https://huggingface.co/datasets/intronhealth/afrispeech-200

A Exploratory Data Analysis

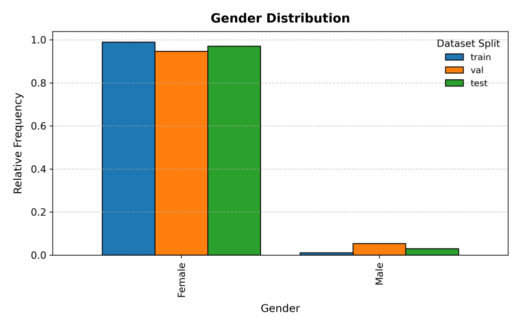
This section demonstrates the age, domain, and gender distributions within the subset of Isizulu accent (main dataset), Swahili accent, and Isixhosa accent from the AfriSpeech-200 dataset (4). It also shows the transcriptions of an example training, validation, and testing clip from the Isizulu accent dataset.



(a) Age Group Distribution of Isizulu accent dataset

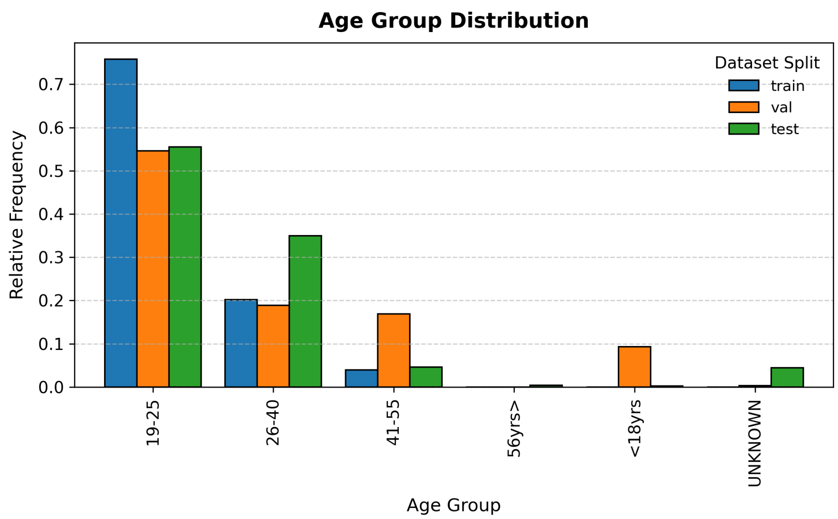


(b) Domain Distribution of Isizulu accent dataset

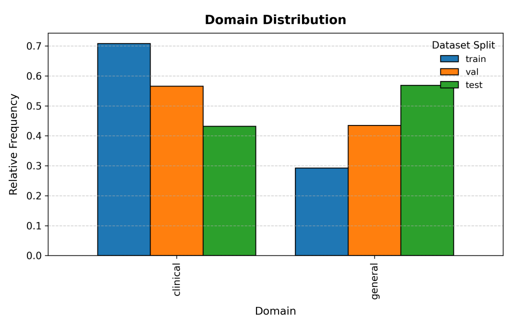


(c) Gender Distribution of Isizulu accent dataset

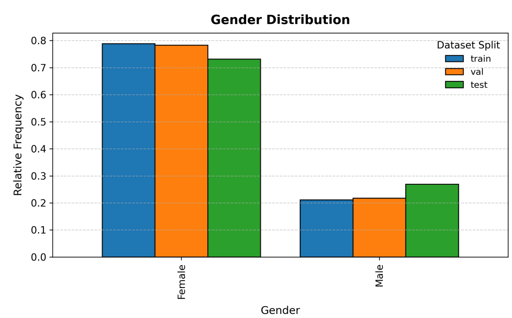
Figure 3: Dataset demographic distributions of the Isizulu accent dataset: (a) Age group distribution, (b) Domain distribution, and (c) Gender distribution. The age distribution is primarily concentrated between 19 and 40 years old. The domain distribution is fairly balanced between clinical and general speech, while the gender distribution is predominantly female.



(a) Age Group Distribution of Swahili accent dataset

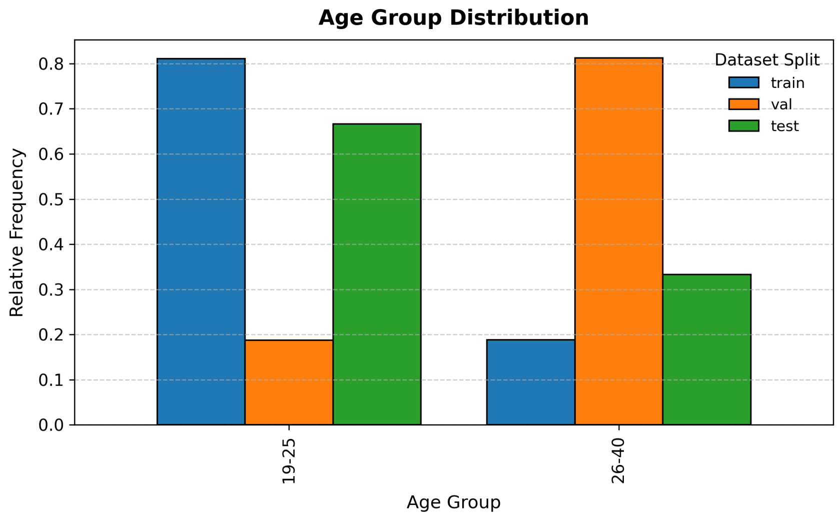


(b) Domain Distribution of Swahili accent dataset

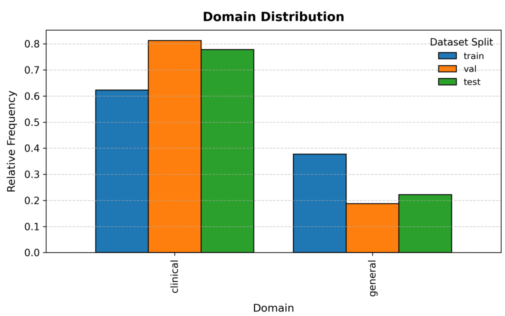


(c) Gender Distribution of Swahili accent dataset

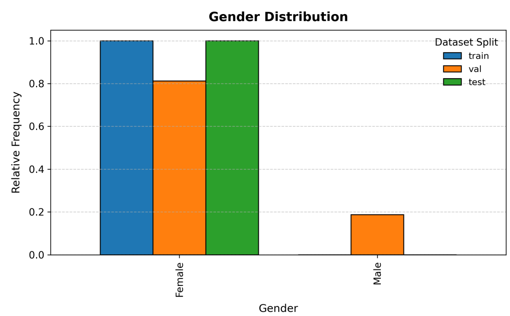
Figure 4: Dataset demographic distributions of the Swahili accent dataset: (a) Age group distribution, (b) Domain distribution, and (c) Gender distribution. The age distribution is primarily concentrated between 19 and 40 years old while having a few <18 years old or 41-55 years old. The domain distribution is fairly balanced between clinical and general speech, while the gender distribution is mainly female.



(a) Age Group Distribution of Isixhosa accent dataset



(b) Domain Distribution of Isixhosa accent dataset



(c) Gender Distribution of Isixhosa accent dataset

Figure 5: Dataset demographic distributions of the Isixhosa accent dataset: (a) Age group distribution, (b) Domain distribution, and (c) Gender distribution. The age distribution is concentrated between 19 and 40 years old. For domain distribution, there's more clinical data then general and the gender distribution is predominantly female.

Training	**Validation**	**Test**
The hedge, which he started building on March 3, cost him roughly 27 million and scored big as stock and debt markets floundered on fears of the coming pandemic—fears, critics say, that he helped stoke.	*The other 17 with symptoms never got tested, either because tests were not available or—like Comstock and Owen—the singers were under the impression that only people in dire condition were eligible.*	*I doubt these immoral scientists and government officials will allow any biometric examination of that DNA.*

Figure 6: Transcriptions of an example training, validation and testing clip from the isizulu accent (main dataset)

B Effect of IPEX and Gradient checkpointing

We also investigate the impact of Intel's PyTorch Extension (IPEX) and gradient checkpointing on system-level performance. Specifically, we assess how these optimizations affect training throughput and memory usage. Since LoRA emerged as the most promising fine-tuning method in our previous experiments, we focus this analysis exclusively on LoRA. Table 5 summarizes the performance of LoRA with and without these two system-level optimizations.

Table 5: Effect of IPEX and gradient checkpointing on training throughput and memory usage.

Configuration	**Training Throughput** (samples/sec, ↑)	**Memory Usage** (GB, ↓)
LoRA (baseline)	**1.18**	10.02
LoRA with IPEX	1.17	9.06
LoRA with Gradient checkpointing	0.84	**5.88**

Surprisingly, enabling IPEX results in slightly reduced training throughput and only a modest improvement in memory usage. One plausible explanation is that many of the optimizations offered by IPEX—such as support for `bf16` and `fp16` precision—are only available on high-end Intel CPUs, such as the Xeon series. When running on consumer-grade CPUs (e.g., Intel Core), these operations may be emulated in `fp32`, which doesn't provide any potential speedup. Additionally, using IPEX introduces overhead by replacing PyTorch's native CPU kernels with Intel-provided implementations, which could lead to additional function call overheads.

In contrast, enabling gradient checkpointing leads to a noticeable drop in training throughput but yields a substantial reduction in memory usage. This trade-off is particularly beneficial for memory-constrained environments. As a result, in all experiments, we use gradient checkpointing by default but disable IPEX.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We claim that we create a library for user's convenient use, where we actually created it and it is visible in github

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed how the library is mainly working only for Whisper family of model.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: No theoretical result.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: All code on github, can just use and link is provided at front.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code is on github, the dataset is described and referenced in paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: See 6.1 Experimental Setup section and section 4 model and data and the code is also provided.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The output is deterministic, cannot have error bar.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The type (CPU/GPU) is included, the memory needed and training time per sample is also included.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [NA]

   Justification: We do not read it because there's nothing that can violate ethics anyway

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: We don't think there is societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: There's no such risk as it is just a library, all models are already availuable online.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited Whisper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: No new assets released.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.