SwarmBee: Swarm Intelligence meets GenAI on Edge

Phase - II

Contents

Introduction

The rising popularity of **LLMs** attracted widespread attention from the public, industry, and academia, leading to their application to various domains. LLMs like GPT-4, LLaMA, Claude and Gemini show much **greater context** and **generalization** and **memory capabilities** when compared to previous language models. Hence, they are used in tasks like text generation, translation and question-answering.

Many enterprises deploy these models on cloud as they offer benefits such as scalability, efficiency, and advanced computational capabilities.

However, the heavy reliance of these LLMs on **cloud computing**, leads to **latency**, **high bandwidth cost**, and **privacy issues**. A way to combat these challenges is to deploy these models on the devices at the network edge, closer to the data sources.

But, deploying these models on the edge devices has its own concerns. **LLMs are resource-intensive**, i.e., they require significant computational power and memory, which are major limiting factors in edge devices.

Hence, we try to use optimization techniques like **genetic algorithm** and compress these LLMs using techniques like **pruning**.

Problem Statement

Design a framework to optimize large language models and deploy them on the Edge for Generative AI purposes, using Federated learning and Metaheuristic algorithms.

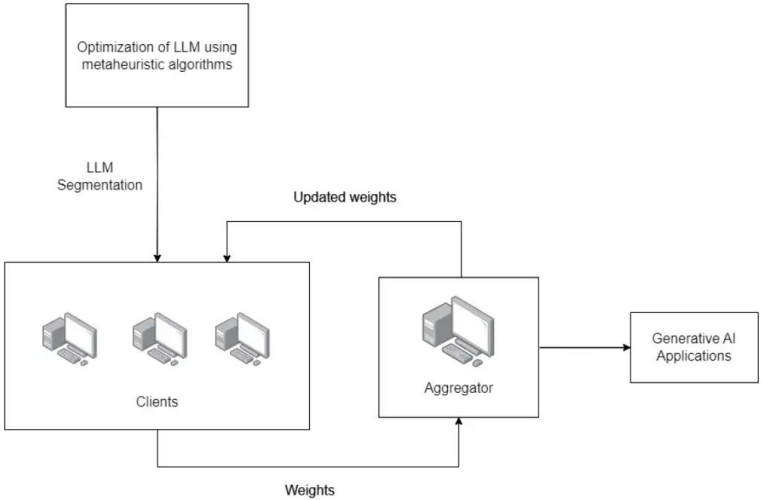
Objectives

1. Design an architecture that facilitates deployment of Large Language models on Edge for Generative AI applications using principles of Federated Learning and Swarm Intelligence.
2. Optimize pre-existing Large Language models to reduce the model size, minimize response time and achieve significant inference performance using metaheuristic algorithms.
3. Address the research gap in the existing literature on deploying Large Language models on the Edge.

Proposed Methodology

The methodology involves the following steps:

1. **Model Optimization:**
   - Select a pre-trained LLM and implement metaheuristic algorithms to optimize the LLM's architecture or hyperparameters for improved performance and efficiency.
   - Evaluate the optimized model’s suitability to run on resource-constrained edge devices and compare its performance with the original foundation model.
2. **Model Transfer:**
   - Transfer the optimized LLM's weights to edge devices.



3. **Federated Learning:**
   - Implement federated learning to collaboratively train the LLM on multiple edge devices without sharing raw data.
   - Select an edge device as an aggregator.
   - Each device updates its local model based on its data and shares model updates with the selected edge device which aggregates the updates and distributes them back to the other edge devices.

4. **Deployment and Application:**
   - Deploy the optimized and locally trained LLM on edge devices.
   - Utilize it for various generative AI applications, such as, text generation, named entity recognition, question answering, text summarization and machine translation.

Implementation Details

ANN:

- Implemented Genetic Algorithm to optimize an ANN model trained on the diabetes dataset.
- Ran the genetic algorithm for 100 generations with the population size of 20.
- Calculated the fitness function and saved the optimal weights.

Implementation Details

ANN:

When compared with other optimizers, following are the test accuracies:

Optimizers	Test Accuracy
SGD	0.6354
RMSProp	0.7712
Adam	0.7860
Genetic Algorithm	**0.7847**

Implementation Details

CNN:

Trained a CNN model for image classification on MNIST dataset and used Genetic algorithm to optimize the training process.

Ran the genetic algorithm for 4 generations with the population size of 10 where each individual was an instance of the model.

	Training Set	Validation Set
Accuracy	0.9958	0.9853
Loss	0.0120	0.0710

Implementation Details

CNN:

Trained a CNN model for image classification on the CIFAR10 dataset and used Genetic algorithm for hyperparameter tuning.

➢ Achieved an cross validation accuracy of 79.10% within 3 generations.

*Whereas, when the model is trained for 60 epochs using the model.fit() method, it achieves validation accuracy of 75.24%

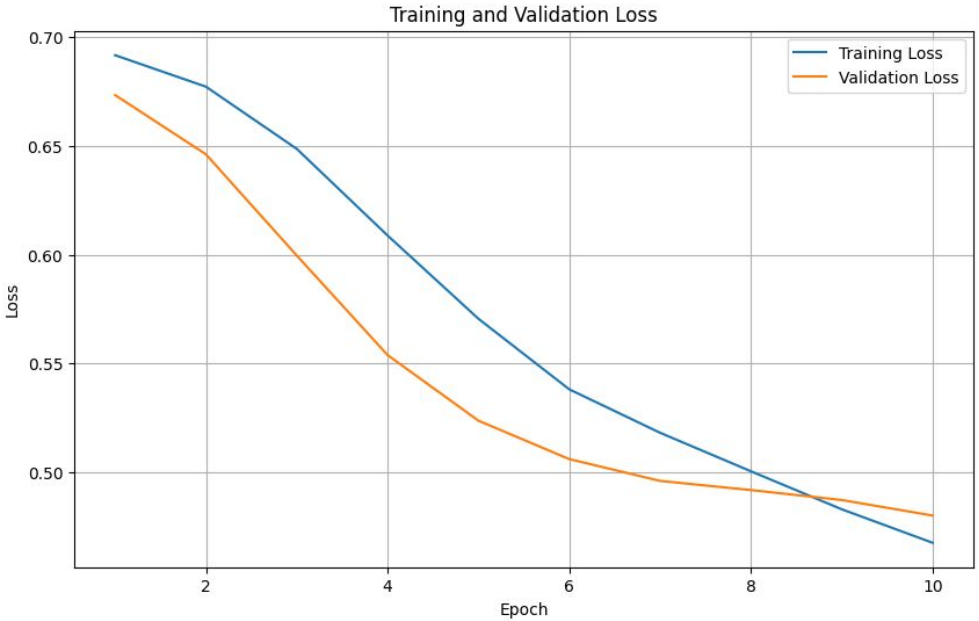
Implementation Details

TinyBert:

To perform fine-tuning of a pre-trained Tiny-BERT model for sentiment classification on a subset of the Rotten Tomato dataset.

The model is a pre-trained BERT model with 4 layers, 312 hidden units per layer, 12 attention heads, and 4 million parameters.

Epochs	Training Loss	Validation Loss
1	0.6918	0.6918
2	0.6773	0.6462
3	0.6487	0.5996
4	0.6088	0.5538
5	0.5705	0.5236
6	0.5381	0.5056
7	0.5181	0.4959
8	0.5004	0.4917
9	0.4829	0.4871
10	0.4675	0.4800



Implementation Details

ALBERT:

Pruned and fine-tuned the albert-base-v2 model for classifying sentences from the IMDB dataset.

The model is a lite version of BERT — with 12 layers, 768 hidden dimensions, 12 attention heads, and 11 million parameters.

Implementation Details

ALBERT:

Pruned evaluation loss: **0.6958**

Loss on Fine-tuning:

Epoch	Training Loss	Validation Loss
1	0.3991	0.4139
2	0.3532	0.2909
3	**0.2754**	**0.3286**

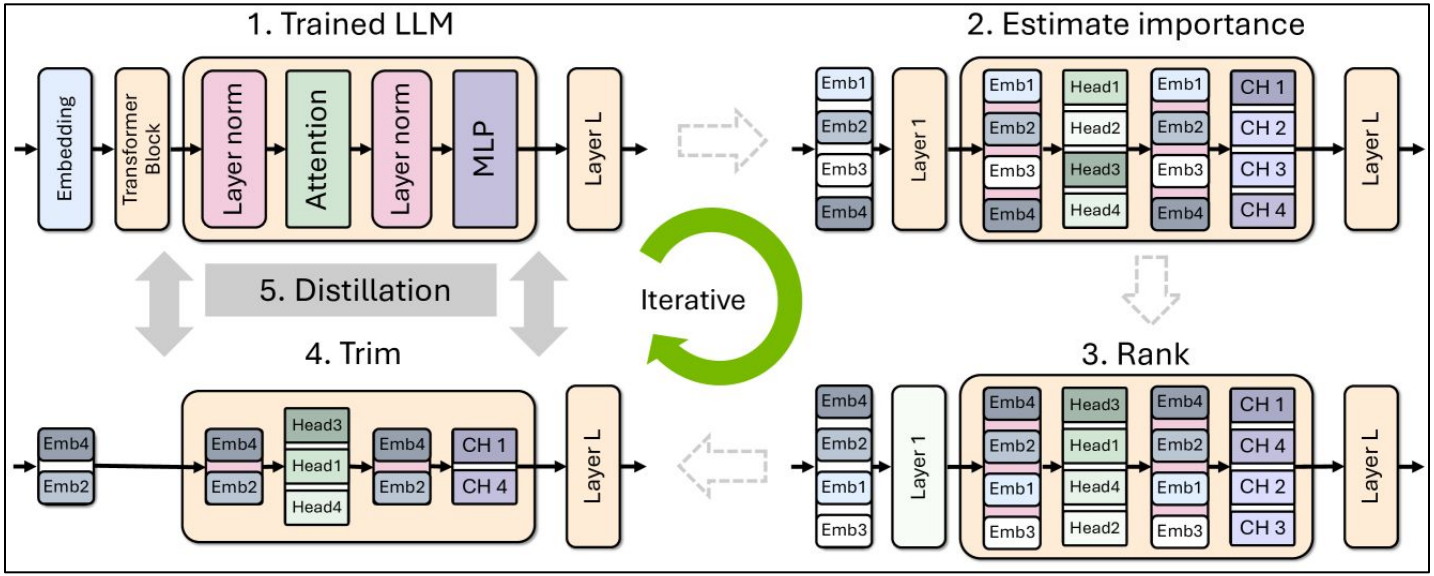
Implementation Details

GPT:

- Explored techniques for compressing large language models (LLMs) through a combination of pruning and knowledge distillation.
- Built a GPT model from scratch with 4 transformer blocks, 4 attention heads and, an embedding size of 256.
- Trained the model on the Shakespeare dataset.
- Performed attention head pruning and embedding pruning on the GPT model.
- Further, we used knowledge distillation to retrain the model after pruning.

Pruning Methodology:

From the paper: Compact Language Models via Pruning and Knowledge Distillation [https://arxiv.org/html/2407.14679v1]



Implementation Details

GPT (3.22M parameters):

Ratio of the pruned weights to the base model:	35.96%
Base loss after the initial training:	2.0135
Pruned evaluation loss (before retraining):	2.7352
Pruned evaluation loss (after retraining):	2.1715

Challenges

faced during implementation

- ⇨ Pruning without significant degradation in performance
- ⇨ Defining an appropriate fitness function for the Genetic Algorithm
- ⇨ Computational requirements
  - Colab (free version) has a dynamic usage limit

Conclusion

- Genetic Algorithms offers a robust approach to hyperparameter tuning, saving resources while enhancing model performance. It's faster than GridSearch and performs better than RandomSearch.
- Genetic Algorithms can potentially find good solutions, but they may not always converge to the optimal solution.
- Fine-tuning the model after pruning is the primary challenge.

Future Work

- Further look into other Pruning methods for LLM Compression (depth pruning, one-shot pruning)
- Research on Model Segmentation and Deploying LLMs locally

Thank You