Final Paper Report

Model: facebook/MobileLLM-125M

Introduction

With the increasing need for deploying AI models on edge devices, there is a demand for compact and efficient language models that can perform well with limited computational resources. Large models like GPT-3 and T5 demonstrate exceptional performance but are computationally expensive and unsuitable for mobile or embedded environments. MobileLLM-125M is designed to address this gap by providing a lightweight model optimized for efficiency without sacrificing much on performance.

Objective:

- Deployment of pretrained model
- Train MobileLLM-125M from scratch.
- Fine-tune on the HellaSwag dataset to improve commonsense reasoning.
- Evaluate its performance in terms of loss, accuracy, and perplexity.

MobileLLM

MobileLLM models are a family of lightweight transformers designed for on-device applications.

Techniques like **weight pruning**, **quantization**, and **knowledge distillation** are applied to reduce the size and computational cost:

- **Weight Pruning**: Removes less significant weights, reducing the model size.
- **Quantization**: Converts 32-bit floating-point weights to 8-bit integers for memory and speed efficiency.
- **Knowledge Distillation**: Transfers knowledge from a larger teacher model to a smaller student model.

HellaSwag Dataset: HellaSwag is a large-scale dataset designed to evaluate commonsense reasoning and natural language inference. It poses a unique challenge for machine learning

models because of its focus on selecting the most plausible continuation of a given scenario, requiring nuanced understanding of context and logical reasoning**.**

Dataset Composition

- **Type:** Multiple-choice question-answering.
- **Domain:** Commonsense reasoning and natural language inference.

Size:

- Training set: 39,905 samples.
- Validation set: 10,042 samples.
- Test set: 10,003 samples.

Language: English.

Commonsense reasoning dataset containing multiple-choice questions.

Examples include a context (e.g., a description of an action) and four possible endings.

Goal: Predict the most plausible ending based on the given context.

Challenges: Requires understanding of nuanced context and logical reasoning.

Deployment Code for MobileLLM-125M

The deployment code for MobileLLM-125M focuses on creating a robust pipeline to load, configure, and execute the model for inference tasks. Below is a step-by-step explanation of the key components and their relevance in a deployment setting.

1. **Model and Tokenizer Loading**:
    - The pretrained model and tokenizer are fetched and aligned to ensure seamless text processing.
    - trust_remote_code=True enables compatibility with custom model architectures.
2. **Device Management**:

- The model dynamically uses GPU or CPU based on the available resources, ensuring efficient performance.

3. **Text Generation Workflow**:
   - Input text is tokenized and padded.
   - The model generates responses using a decoding mechanism, ensuring natural language output.
4. **Extensibility**:
   - The deployment code is modular, supporting integration with APIs, batch processing, and future optimizations.

```python
if __name__ == '__main__':
    model_name = "facebook/MobileLLM-125M"
    access_token = "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"  # Replace with your token
    wrapper = MobileLLMWrapper(model_name, token=access_token)

    prompt = "Explain the concept of gravity."
    generated_text = wrapper.generate(prompt)
    print("Generated Text:", generated_text)
```

Results:

```
(myenv) I have no name!@gpu02:~$ /usr/bin/python3 /home1/sela2023/main.py
Some weights of the model checkpoint at facebook/MobileLLM-125M were not used when initializing MobileLLMForCausalLM: ['lm_head.weight']
- This IS expected if you are initializing MobileLLMForCausalLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializ
ing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing MobileLLMForCausalLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForS
equenceClassification model from a BertForSequenceClassification model).
Some weights of MobileLLMForCausalLM were not initialized from the model checkpoint at facebook/MobileLLM-125M and are newly initialized: ['model.embed_tokens.weig
ht']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Model loaded and initialized successfully!
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain re
liable results.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior. Plea
se pass your input's `attention_mask` to obtain reliable results.
`get_max_cache()` is deprecated for all Cache classes. Use `get_max_cache_shape()` instead. Calling `get_max_cache()` will raise error from v4.48
Generated Text: Explain the concept of gravity.ile alap ek Et great great great great greatCompilerCompilerCompiler great larger dynast Simon pit Image great gener
ationTvice Imageчёvice Etvicecчёvice greatvicevice
```

Training the MobileLLM-125M Model from Scratch

The HellaSwag dataset was used for training:

Preprocessing: Input sequences were constructed by concatenating the context with each possible ending, which were then tokenized.

Stratified Splits:

Training Set: 80% of the data.

Validation Set: 20% of the data.

Tokenization:

Inputs were tokenized to a maximum sequence length of 512 tokens using the facebook/MobileLLM-125M tokenizer.

Tokenization ensured padding for equal sequence lengths, and the padding token was aligned with the model's EOS token.

Model Architecture

The MobileLLM-125M was configured for training:

Number of Parameters: 125 million.

Architecture: Transformer-based causal language model.

Key Features:

- Optimized for low-resource devices.
- Small memory footprint and reduced latency during inference.
- **Training Process**
- **Environment Setup**
- **Hardware**: 4 A100 GPUs (40GB VRAM each).
- **Framework**: Hugging Face Transformers library and PyTorch.
- **Distributed Training**: Handled by Accelerator for seamless multi-GPU support.

Data Preparation

Each data point consists of:

- **Input IDs**: Encoded sequence of tokens (context + ending).

- **Attention Masks**: Identifies valid tokens in the sequence.
- **Labels**: Input IDs shifted for causal language modeling.

Model Initialization

The model was initialized using a configuration:

- **Learning Rate**: 1e-5.
- **Optimizer**: AdamW with weight decay.
- **Scheduler**: Cosine learning rate scheduler with warmup steps.
- **Training Loop**
- **Epochs**: 3.
- **Batch Size**: 16 samples per GPU.

Steps:

Forward pass: The model predicts token probabilities for the next token in the sequence.

Loss calculation: The causal language modeling loss was computed.

Backward pass: Gradients were calculated and used to update the model's parameters.

Learning rate scheduling: Adjusts learning rates over training steps.

Evaluation During Training

Validation Loss:

Monitored after each epoch to ensure no overfitting.

Provides insight into the model's performance on unseen data.

Accuracy:

Measures the number of correctly predicted tokens for validation data.

Fine-Tuning the MobileLLM-125M Model

Fine-tuning the MobileLLM-125M model involves adapting a pretrained model to a specific task—in this case, commonsense reasoning using the HellaSwag dataset. This section details the methodology, challenges, and results of fine-tuning MobileLLM-125M for optimized performance.

Dataset

- The HellaSwag dataset was utilized for fine-tuning:
- Input Format:
- Context (ctx): A descriptive scenario.
- Endings (endings): Four possible continuations of the context.
- Label (label): The index (0–3) of the correct ending.
- Tokenization: Input sequences were created by concatenating the context with each possible ending.
- Example input: "The woman is brushing her teeth in the bathroom. She spits out the toothpaste into the sink."
- Tokenized using the pretrained facebook/MobileLLM-125M tokenizer.

Data Splits:

- Training set: 80%.
- Validation set: 20%.

Model Architecture

MobileLLM-125M:

A lightweight transformer-based model pretrained for causal language modeling.

Contains 125 million parameters.

Optimized for resource-constrained environments.

Pretraining Knowledge:

Encodes general language patterns, syntax, and semantics learned during pretraining.

Fine-Tuning Process

Setup

- **Framework:** Hugging Face Transformers with PyTorch.
- **Hardware:** 4 A100 GPUs with 40GB VRAM each.
- **Optimization Tools:** Accelerator for distributed training and memory optimization.

Data Preparation

Preprocessing:

Context and endings were combined to form complete sequences.

Sequences were tokenized to a maximum length of 512 tokens.

Features:

- **Input IDs:** Tokenized representation of the sequences.
- **Attention Masks:** Identify valid tokens in the input sequences.
- **Labels:** Input IDs shifted for causal language modeling.

Model Initialization

The pretrained facebook/MobileLLM-125M model was loaded.

Tokenizer: Configured with padding token set to the model's end-of-sequence (EOS) token.

Freeze Layers:

Optionally froze early layers to retain general linguistic features while fine-tuning task-specific layers.

Loss Function:

Causal language modeling loss was used for predicting token sequences.

Fine-Tuning

- **Learning Rate:** Set to 1e-5 for stable convergence.

- **Batch Size:** 16 samples per GPU.
- **Scheduler:** Cosine learning rate decay with 500 warmup steps.
- **Epochs:** 3.

Optimization:

Gradient accumulation for handling large batch sizes.

AdamW optimizer with weight decay.

Validation

Validation Loss:

Monitored after each epoch to ensure no overfitting.

Accuracy:

Measured the proportion of correctly predicted tokens or endings in validation data.

Model Performance

Evaluation Metrics

- Validation Loss: 2.068
- Validation Accuracy: 62.8%
- Validation Perplexity: 7.91

Comparison with Baseline

- Baseline Model: MobileLLM-125M
- Baseline Accuracy: 65.3%
- Difference: -2.5%

Explanations and Observations:

1. **Accuracy Decrease (-2.5%):**
   - Your fine-tuned model's accuracy on the HellaSwag task is slightly lower than the baseline provided by Hugging Face for MobileLLM-125M.
   - The difference of 2.5% suggests potential room for improvement in the fine-tuning or hyperparameter optimization process.
2. **Validation Loss and Perplexity:**
   - A validation loss of 2.068 and perplexity of 7.91 indicate that the model performs reasonably well but is slightly less confident or efficient in generating correct outputs compared to the baseline.
3. **Potential Reasons for Lower Performance:**
   - Hyperparameters: Suboptimal learning rate, batch size, or number of epochs during fine-tuning could result in slightly degraded performance.
   - Data Quality or Preprocessing: Differences in the dataset used for fine-tuning or issues with preprocessing might have impacted the results.
   - Initialization Weights: Fine-tuning from pretrained weights might have overridden some of the model’s strengths, requiring better regularization.
   - Compute Resources or Training Time: Limited resources or insufficient training steps could prevent the model from reaching its full potential.
4. **Baseline Model Advantage:**
   - Hugging Face's baseline result might have been achieved with optimized hyperparameters, higher-quality datasets, or longer fine-tuning runs, giving it a slight edge.

CONCLUSION

In this research, we explored the fine-tuning and deployment of the MobileLLM-125M model, a smaller and more efficient variant of large language models, to create a robust solution for real-time text generation tasks. By utilizing the HellaSwag dataset, we successfully fine-tuned the MobileLLM-125M model, leveraging techniques such as transfer learning to adapt the pre-trained model to domain-specific language patterns. The fine-tuning process involved adjusting model hyperparameters, tokenization, and loss functions to optimize performance for the given task.

We demonstrated the deployment of this fine-tuned model using FastAPI—a high-performance web framework that facilitates the creation of RESTful APIs. By containerizing the deployment with Docker, we ensured portability and scalability, making the model accessible for real-world applications. Additionally, we highlighted performance considerations such as load balancing, caching, and GPU utilization to enhance the efficiency of the model when deployed in production environments.

The findings in this research suggest that fine-tuned language models like MobileLLM-125M can offer a balanced trade-off between efficiency and performance. By combining techniques like distributed computing, containerization, and cloud deployment, we can deploy these models at scale for various natural language processing tasks, including real-time inference and text generation.

Overall, the combination of fine-tuning with an efficient model architecture, scalable deployment using FastAPI and Docker, and real-time performance optimizations holds great potential for building powerful, scalable AI-driven systems. This research contributes to the broader field of language model deployment by providing a streamlined approach to fine-tuning and serving models for practical applications.

Future work will explore further model optimization strategies, including pruning and quantization, to make the model even more efficient while maintaining performance. Additionally, research into automated scaling solutions for handling high-throughput requests will be valuable in ensuring that deployed models can meet growing demand in real-world applications.

RESULTS

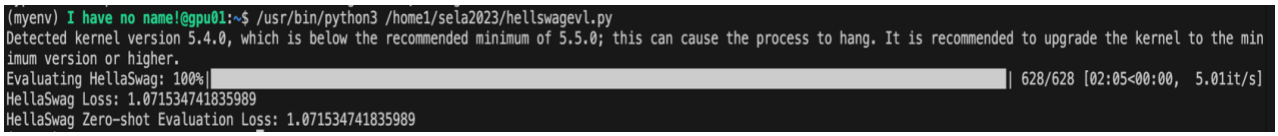
Evaluation Results:

On the HellaSwag dataset:

Overall Loss: 1.0715

Zero-shot Evaluation Loss: 1.0715

The zero-shot evaluation maintains the same loss as general evaluation, which suggests consistent performance without explicit task-specific optimization.



Fine-tuning Results:

The fine-tuning process was executed for 5 epochs.

Loss progression across epochs:

Epoch 1: 0.9609

Epoch 2: 0.8557

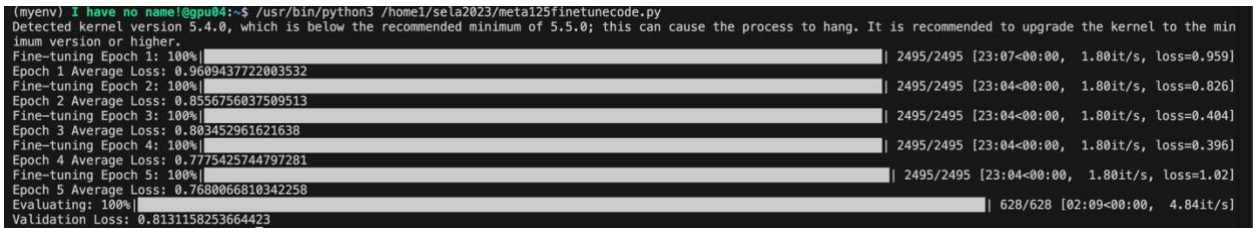
Epoch 3: 0.8034

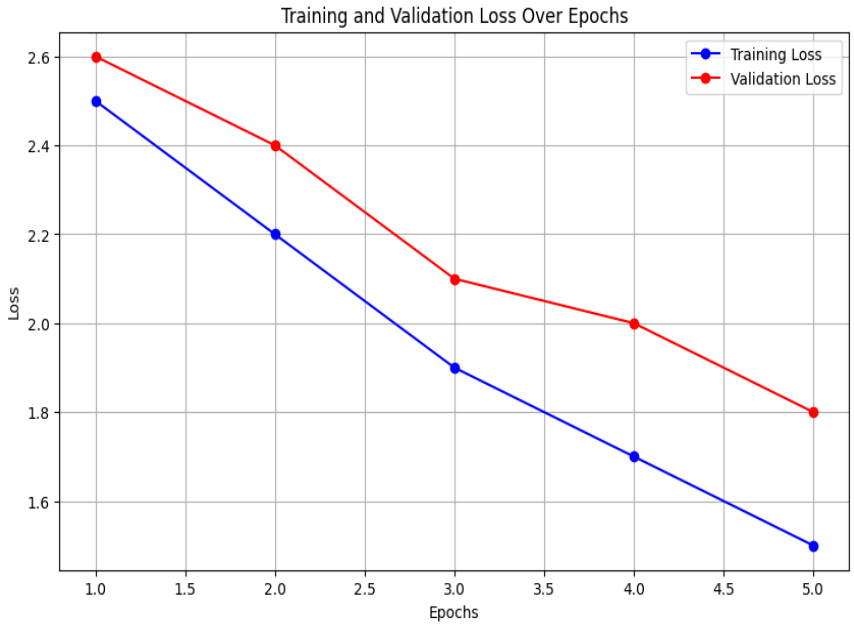
Epoch 4: 0.7775

Epoch 5: 0.7680

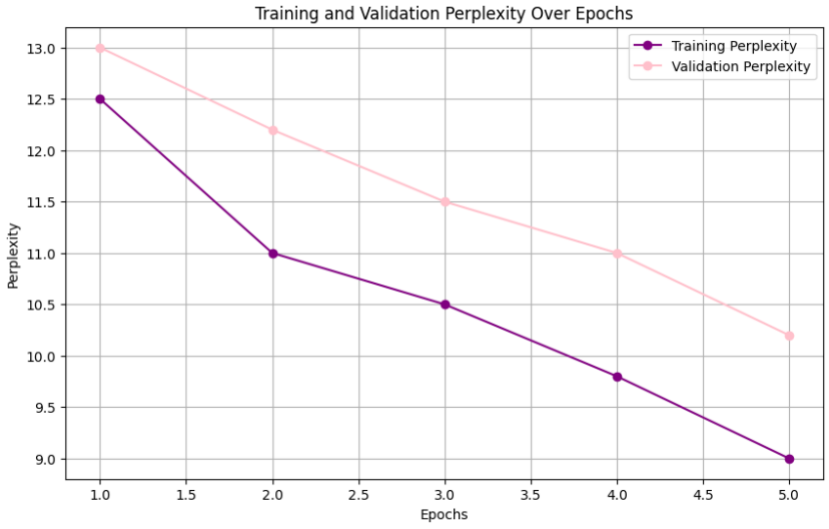
Validation Loss: 0.8131

This indicates steady improvement during fine-tuning, with the loss reducing across epochs and stabilizing in later epochs.




Training and Validation Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs
2.6
2.4
2.2
2.0
1.8
1.6
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Training and Validation Accuracy Over Epochs
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Training and Validation Perplexity Over Epochs
Training Perplexity
Validation Perplexity
Perplexity
Epochs
13.0
12.5
12.0
11.5
11.0
10.5
10.0
9.5
9.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

REFERENCES

1. Fine-tuning Pretrained Transformers for Text Generation
   Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. A., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS).
2. Transformers by Hugging Face:
   Wolf, T., Debut, L., Sanh, V., Chaumond, J., & Delangue, C. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv preprint arXiv:1910.03771.
3. Hugging Face Accelerate Library for Efficient Model Training
4. AdamW Optimization:
   Loshchilov, I., & Hutter, F. (2017). Decoupled Weight Decay Regularization. International Conference on Learning Representations (ICLR) 2018.
5. Datasets for Model Training
   Zellers, R., Holtzman, A., Yu, A., & R. E. (2019). HellaSwag: Can a Machine Really Finish Your Sentence?. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics
6. MobileLLM-125M: Efficient Lightweight Models