Assignment 2: Building a Small-Scale Foundation Model from Scratch

1 Background and Motivation

Training a small-scale transformer from scratch helps students understand the core architecture and dynamics of foundation models. By implementing a mini-GPT and performing next-token prediction, students learn how tokenization, model architecture, and hyperparameters impact learning and generalization.

2 Learning Objectives

After completing this assignment, students will be able to:

1. Implement a transformer-based language model using PyTorch.
2. Train a model from scratch on preprocessed datasets for next-token prediction.
3. Track training metrics such as loss and perplexity.
4. Experiment with hyperparameters (learning rate, batch size, sequence length, number of layers).
5. Save and load model checkpoints.
6. Visualize training dynamics and interpret results.

3 Model Requirements

- Implement a basic transformer-based language model (mini-GPT) with:
  - 1–2 transformer layers
  - Embedding dimension: 64–256
  - Multi-head attention: 2–4 heads
  - Positional encoding or embeddings
- Output logits for next-token prediction.
- Apply layer normalization and appropriate activation functions.

4 Dataset Requirements

- Use preprocessed dataset from Assignment 1 (tokenized and cleaned).
- Sequence length: 32–128 tokens per input block.
- Ensure data batching and shuffling for efficient training.

5 Training Requirements

- Implement training loop in PyTorch:
  - Forward pass, loss computation, backward pass, optimizer step
  - Cross-entropy loss for next-token prediction
  - Track average loss per epoch
- Evaluate model performance using **perplexity**.
- Save model checkpoints after training.
- Experiment with hyperparameters such as:
  - Learning rate (e.g., 1e-3, 5e-4)
  - Batch size (e.g., 16, 32, 64)
  - Number of layers (1–2)
  - Embedding size (64–256)

6 Deliverables

1. Python scripts or Jupyter notebook containing:
   - Model implementation
   - Training loop
   - Checkpoint saving/loading
   - Loss and perplexity logging
2. Model checkpoint file (e.g., `mini_gpt_checkpoint.pt`)
3. Visualizations of training loss and perplexity curves
4. Short report (2–4 pages) including:
   - Model architecture and parameters
   - Dataset details
   - Training setup and hyperparameter experiments
   - Observations and challenges

7 Evaluation Criteria

Criterion	Weight	Description
Model Implementation	25%	Correct transformer architecture, embedding, attention layers
Training Correctness	25%	Proper loss computation, backpropagation, optimizer usage
Metrics and Analysis	20%	Perplexity calculation, loss tracking, hyperparameter experiments
Code Quality	15%	Readability, modularity, comments, reproducibility
Report and Visualizations	15%	Clear explanation of model, dataset, training, and results