Assignment 3: Transformer is All You Need

Due Date: Tuesday, November 18, 2025 at 12:00 AM (midnight, Eastern Time)
Policy: https://columbia-coms4995.github.io/aml-fall2025/#policies

📃 Submission

- A runnable Jupyter/Colab Notebook (`.ipynb`)
- A PDF Report (4+ pages, with clear structure and visualizations)

Important Reminders

Please carefully follow these instructions to ensure your submission can be graded smoothly:

- Submit an actual report: All required visuals (plots, attention maps, samples) must appear directly in your PDF report - do not leave them only in the notebook
- Provide a shareable Colab link: Include the link in your PDF so we can run your notebook if needed
- Keep explanations in the report: Only minimal inline comments should appear in your code
- Use the "**Select Pages**" feature on Gradescope to tag pages corresponding to each part
- Avoid data leakage: Your model should never access the validation set during training or imputation
- Cover all assignment components: The rubric is designed for balanced grading; clearly shows data processing, model setup, and interpretation

🎯 Objective

This assignment introduces the **Transformer architecture** - the foundation of modern Large Language Models (LLMs) - in a computationally lightweight, intuitive way

You will:

- Implement the **core logic** of a Transformer block from scratch - including positional encoding, self-attention, residual connections, and the feed-forward network etc
- Train a Tiny Transformer on a small, toy dataset for the next-token prediction task
- Visualize and interpret the model's internal behavior, focusing on attention patterns

📂 Dataset

Shakespeare Tiny Corpus for the **Next Token Prediction** task

🛠️ Steps

Data Preparation

a. Load the Tiny Shakespeare text

b. Tokenization

Use a `subword-level tokenizer` (e.g., Byte Pair Encoding, WordPiece) to convert text into integer tokens. This captures meaningful text units (like “ing”, “the”, “tion”) rather than single characters, improving both efficiency and coherence

Keep the vocabulary size ≤ 500 for lightweight experiments

c. Sequence formatting

Split the tokenized text into overlapping fixed-length sequences (e.g., 50 tokens)

For each sequence:

- Input: first N tokens
- Target: same sequence shifted by one position (next-token prediction)

Example

- Input : [71, 32, 98, 101, 11, 111]
- Target : [32, 98, 101, 11, 111, 114]

d. Data split

Use 80% for training and 20% for validation, no test set is needed this time

e. Token embedding

Use `nn.Embedding` for tokens and add positional encodings

Implement a Tiny Transformer

Build a minimal Transformer-based language model including:

- Positional encoding (sinusoidal or RoPE if you want more challenges)
- Self-attention module
- Apply a causal mask to prevent the model from attending to future tokens
- Feed-forward network with residual connections and RMSnorm

Train it using `cross-entropy loss` for next-token prediction

Visualization & Interpretation

Plot training and validation loss curves over epochs
Plot attention heatmaps for sample sequences to visualize which tokens attend to which
Report Perplexity (PPL) on the validation set as the main evaluation metric - lower is better
Optionally, include sample generations to qualitatively assess model fluency and coherence

Discussion & Reflection

Some questions worth thinking about:

- What patterns do you observe in the attention maps?
- Which hyperparameters (learning rate, context length, model size) had the greatest impact on stability?
- How does attention evolve as the model trains over epochs?
- What role do positional encodings play - could the model function without them?
- Reflect on runtime and memory footprint - where are the bottlenecks?

You do not need to answer every question exhaustively - focus on those most relevant to your observations and insights. **The journey of exploration matters more than the final results**

Report Writing

Structure (4+ pages):

- Introduction - problem definition, dataset, and objective
- Methods - model design and training setup
- Results - loss curves, attention maps, generated samples
- Discussion - interpretation and takeaways
- AI Tool Disclosure - describe how any AI tools assisted

🔑 Pointers & Hints

- Use PyTorch only for this assignment
- Keep the model lightweight - 2 Transformer blocks with a hidden size ≤ 128 should be sufficient. Feel free to scale up if your computational budget allows
- Visualize results such as:
    - Attention matrices - visualize which tokens attend to which (use heatmaps)
    - Perplexity for next-token prediction tasks - a standard measure of how well the model predicts unseen text
    - Perplexity = `exp(validation cross-entropy loss)` to be reported.
- For debugging, compare parts of your implementation (e.g., attention output) against the reference from `torch.nn.Transformer` or `transformers.AutoModel`
- Use `torch.no_grad()` during visualization or evaluation to reduce memory and runtime
- Ensure reproducibility by setting random seeds

📊 Grading Rubric

Category	Weight	Description
Implementation & Training	**30**	Correct setup and functioning Tiny Transformer model
Attention Visualization & Analysis	**30**	Quality and clarity of visual interpretation
Experiment Design & Discussion	20	Depth of reflection on architecture and results
Report Clarity & Presentation	15	Organized, well-written, visuals embedded properly
AI Tool Usage Disclosure	5	Transparent acknowledgment of external tools