DEEP LEARNING MINI PROJECT | 16.12.2025 | TEAM L - BHANUPRIYA RANJIT & MAHIDHAR REDDY VAKA |

Character-Level Transformer Language Model (Mini GPT)

University of Geneva

CONTENTS

Objective and Dataset

Objective:

- Build a GPT-style decoder-only Transformer
- Train it as a character-level language model
- Predict the next character given a sequence
- Generate Shakespeare-style text using autoregressive sampling

Dataset:

- Complete works of William Shakespeare, including his plays, poems, and sonnets
- 65-unique character vocabulary
- 90% training, 10% validation

Model Configurations:

- Block size , N = 128
- Batch size, B= 128
- Embedding size = 768
- Layers = 12
- Heads = 8

Tokenization

Character-Level Tokenization:

- Every unique character gets an integer ID
- stoi: char → index (used during training)
- itos: index → char (used during generation)

Input/Target Creation:

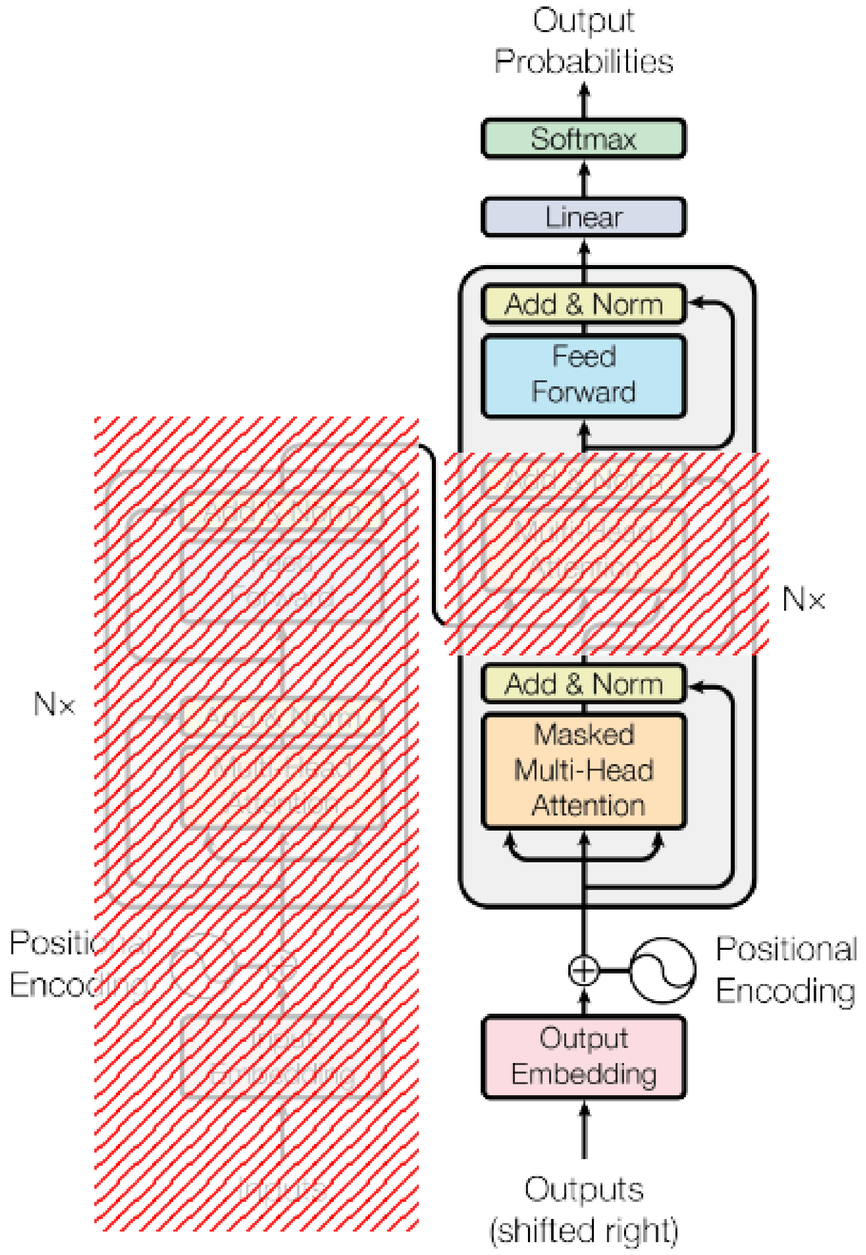
For each sliding window of length 128:

- input, x = chars 0 to 127,
- target, y = chars 1 to 128 (shifted by 1 position)

Train/Validation Split:

- Build the full vocabulary once for both training & validation data- typically ~65 characters for Shakespeare
- Split with 90% training, 10% validation
- Train set: Used to learn model weights
- Validation set: Used to check generalization
- Prevents overfitting

Model Architecture - Decoder only transformer



GPT (Radford, 2018)

Model Architecture - Decoder only transformer

Token Embeddings

- Converts each character ID → 768-dimensional vector
- Learns representation of characters during training

Position Embeddings

- Attention has no inherent notion of order - Inject sequential order
- 768 dimensional vector - Added to token embeddings

Stacked Transformer Blocks (12 layers)

- Each block contains:
    - Multi-Head Causal Self-Attention
    - Residual connections
    - LayerNorm
    - Feedforward MLP (GELU activation)

Output Head

- Final LayerNorm
- Linear layer maps 768 → 65 logits
- Softmax converts logits → probability distribution over next character

Causal Self Attention in Transformer Block

Attention Process

Given input embeddings X:

- Compute query (Q), key (K), value(V):

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V$$

- Compute attention scores:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

- Apply causal mask:

$$S_{ij} = \begin{cases} S_{ij}, & j \leq i \\ -\infty, & j > i \end{cases}$$

- Apply softmax:

$$A = \text{softmax}(S)$$

- Weighted sum of values (Attention Output): Output = AV

Multi-Head Attention

- 8 heads run in parallel
- Each head learns different patterns
- Heads concatenated
- projected back to original embedding dimension(768)

Causal Masking

- Ensures model can only use past information
- Necessary for next-character prediction

Feed Forward MLP in Transformer Block

Why Multi Layer Perceptron?

- Attention → Look around and gather information from other tokens
- MLP → Process this information token-by-token independently

What happens in the MLP part of Transformer block?

- Expands the feature dimension (768 → 3072) : gives model more space to learn richer features
- Applies a non-linear Activation (GELU): $GELU(x)=x\cdot\Phi(x)$
- Projects back down (3072 → 768): output shape matches attention output
- Applies Dropout : randomly zeroes some activations.

MLP Formula:

$MLP(x)=W2(GELU(W1x+b1))+b2$ where: $W1:768\rightarrow3072$, $W2:3072\rightarrow768$

Complete Model-Training set up

GPT Model Architecture

- Token & Position Embeddings
- 12 Decoder Blocks: LayerNorm → Causal Multi-Head Attention→LayerNorm→ GELU MLP
- Final LayerNorm
- LM Head → predicts next character

Training Pipeline

- Each Iteration: Get a batch → Forward Pass → Compute Cross entropy loss →Backpropagation →Gradient Clipping →Update Weights using AdamW optimizer
- Adam + decoupled weight decay, more stable than Adam
- Dropout = 0.3 for regularization
- 6000 training iterations

Evaluation

- Every 100 steps: Compute train/val loss → Compute Perplexity →save checkpoint if validation loss improves

Generation Flow (Inference stage)

- Feed current context → Take the last token logits → Apply temperature(0.8) & top-k filter (40)→Convert logits to probabilities →Sample next character using multinomial sampling→Append character and repeat

Optimizations Used

Dropout

- Prevents over fitting - randomly drops units during training
- Attention Dropout
- Residual Dropout

Gradient Clipping

- Limits the maximum gradient norm (we use 1.0)
- Prevent exploding gradients during back prop, stabilizes training

AdamW Optimizer

- Used for weight updates (learning rate = 1e-4)
- Adam + decoupled weight decay - better generalization
- Fixes Adam’s tendency to overfit with L2 regularization

Layer Normalization

- Normalizes activations across embedding dimensions
- Prevents gradients from exploding/vanishing

Residual Connection

- Each block adds its input back after attention and MLP
- Helps gradients flow easily → avoids vanishing gradients.

Temperature Scaling

- temp < 1 → sharper distribution → more deterministic text
- temp > 1 → flatter distribution → more creative/random output.
- We use temp = 0.8 → balanced creativity & coherence.

Results and Observations

- **Quality of Generated Text**
  - Shows it learned vocabulary, style, and dialogue format
  - Model captures line breaks, speaker names, and old English phrasing

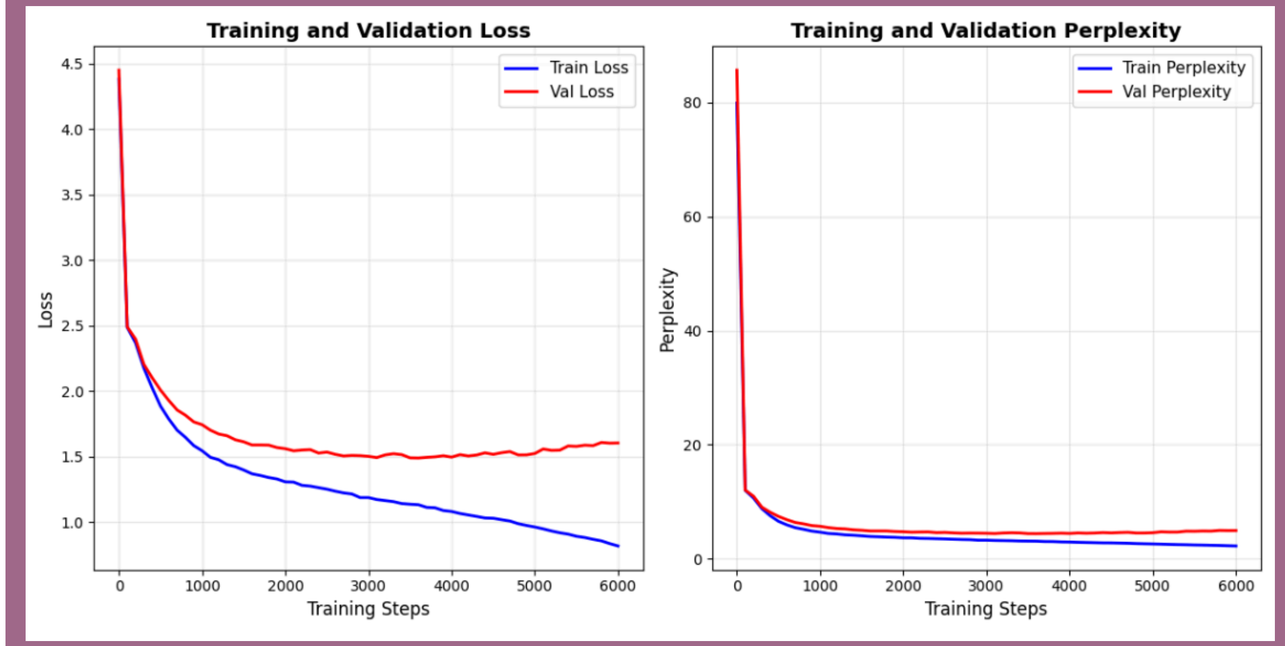
- **Training and Validation Loss Behaviour**
  - Training loss decreases smoothly → model is learning progressively
  - Validation loss decreases until ~3600 steps, then begins to rise → Overfitting (small dataset,model size is large)
  - Mitigation options: reduce training steps, increase dropout

- **Training and Validation Perplexity Behaviour**
  - Perplexity drops sharply from ~80 → ~5 in early steps → Model quickly learns basic character patterns
  - Shows the model makes confident predictions for next characters

```
Seed: 'O God, O God!'
O God, O God! 'Judgm, all my words and blood
Is dishonourable steps, for a white Edward's daughter
Will be content of the devil.

DUKE OF YORK:
So that bid me not, my lord,
And therefore he is my son's lies, shall
```

- Shakespeare style text generated by the model



- Loss and Perplexity Plots

Saving and Loading Model

- **Why Do We Save the Model?**
  - Prevent losing training progress
  - Store the best-performing checkpoint
  - Reuse the trained model anytime (for inference)
  - Avoid retraining (expensive and slow)

- **What Gets Saved (Checkpoint Contents)?**
  - model_state_dict → trained GPT weights
  - optimizer_state_dict → AdamW internal states
  - config → all hyperparameters
  - stoi / itos → vocabulary maps (char → index)
  - Training curves → train/val losses + steps
  - best_val_loss → best checkpoint tracking

- **Saving the Best Model (During Training)**
  - Saves only when validation loss improves
  - Stores everything needed to resume or run inference
- **Loading the Saved Model (for Generation)**
  - load_state_dict() restores learned weights
  - model.eval() disables dropout for stable text generation

```
if losses['val'] < best_val_loss:
    best_val_loss = losses['val']
    best_step = iter
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'config': config,
        'stoi': train_dataset.stoi,
        'itos': train_dataset.itos,
        'iter': iter,
        'best_val_loss': best_val_loss,
        'train_losses': train_losses,
        'val_losses': val_losses,
        'steps': steps,
    }, '/content/drive/MyDrive/best_model.pt')
```

- Code used for saving the model -During training

```
best_checkpoint = torch.load('/content/drive/MyDrive/best_model.pt', weights_only=False)
model.load_state_dict(best_checkpoint['model_state_dict'])

model.eval()
```

- Loading the saved model for generation

Conclusion

Built a decoder-only GPT-style Transformer from scratch

Trained on the Shakespeare corpus using AdamW, dropout, and gradient clipping for stability

Implemented full pipeline: tokenization → training → evaluation → inference → checkpointing

Achieved good results with low validation perplexity and Shakespeare-like text generation

Thank You