Decoder-Only Transformer: Implementation and Evaluation Report

By - Arnav Gupta 2022TT11352

Table of Contents

---

1. Abstract

This report presents the implementation and comprehensive evaluation of a decoder-only Transformer model trained on the TinyStories dataset. The model comprises 3 layers, 8 attention heads, and achieves a perplexity of 9.41 on validation data. We implement and evaluate several optimization techniques including KV caching (achieving 4.52× speedup) and beam search decoding. Our experiments demonstrate the effectiveness of modern Transformer architectures for language generation tasks while highlighting the trade-offs between different decoding strategies.

Key Results:

- Validation Perplexity: 9.41
- Average BLEU Score: 0.0176
- KV Cache Speedup: 4.52×
- Model Parameters: ~8.2M

---

2. Introduction

2.1 Motivation

Language models based on the Transformer architecture have revolutionized natural language processing. This project implements a decoder-only Transformer from scratch to understand the fundamental mechanisms behind modern language models like GPT.

2.2 Objectives

1. Implement a decoder-only Transformer architecture with multi-head attention
2. Train the model on the TinyStories dataset
3. Evaluate model performance using perplexity and BLEU metrics
4. Implement and benchmark optimization techniques (KV caching, beam search)
5. Visualize and interpret attention patterns

2.3 Dataset

TinyStories Dataset: A collection of short, simple stories designed for language model training

- Training samples: ~2.1M stories
- Validation split: 10% of training data
- Vocabulary size: 15,487 tokens
- Token preprocessing: Lowercase, word-level tokenization

---

3. Model Architecture and Implementation

3.1 Architecture Overview

The decoder-only Transformer follows the GPT architecture with the following specifications:

Hyperparameter	Value
Model Dimension (d_model)	300
Number of Layers	3
Attention Heads	8
Attention Dimension (d_attn)	304
Feed-Forward Dimension (d_ff)	512
Context Length	64 tokens

Dropout Rate	0.1
Vocabulary Size	15,487

3.2 Key Components

3.2.1 Multi-Head Attention

The attention mechanism computes scaled dot-product attention across multiple heads:

$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d_k}) V$

Implementation Features:

- Parallel computation across 8 attention heads
- Causal masking to prevent attending to future tokens
- Dimension per head: $d_k = 304 / 8 = 38$

3.2.2 Positional Encoding

Sinusoidal positional encodings inject position information:

$PE(pos, 2i) = \sin(pos / 10000^{(2i/d_{model})})$
$PE(pos, 2i+1) = \cos(pos / 10000^{(2i/d_{model})})$

3.2.3 Transformer Block

Each layer consists of:

1. **Multi-Head Attention** with residual connection
2. **Feed-Forward Network** (2-layer MLP) with residual connection
3. **Layer Normalization** applied before each sub-layer (Pre-LN architecture)

3.2.4 Token Embeddings

- Initialized with **FastText pre-trained word vectors** (wiki.simple)
- Frozen during training to leverage pre-trained semantic knowledge
- Coverage: Found embeddings for a significant portion of vocabulary
- Unknown words initialized with random vectors

---

4. Training Process

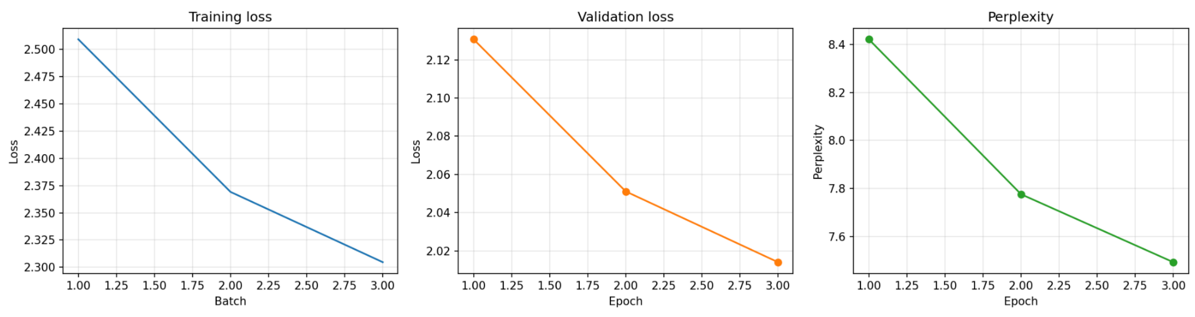
4.1 Training Configuration

Parameter	Value
Batch Size	32
Learning Rate	3e-4
Optimizer	AdamW
Number of Epochs	3
Loss Function	Cross-Entropy (ignoring padding)
Device	CUDA (GPU)

4.2 Training Procedure

1. **Data Preprocessing**:
   - Tokenization using custom vocabulary
   - Adding special tokens: `<sos>`, `<eos>`, `<pad>`, `<unk>`
   - Truncation to context length (64 tokens)
   - Padding shorter sequences
2. **Teacher Forcing**:
   - Input: tokens[:-1]
   - Target: tokens[1:]
   - Enables parallel training across sequence positions
3. **Regularization**:
   - Dropout (0.1) applied after embeddings, attention, and feed-forward layers
   - Frozen embedding layer to prevent overfitting

4.3 Training Results



Epoch-wise Performance:

Epoch	Train Loss	Val Loss	Perplexity

1	2.509	2.124	8.41
2	2.366	2.050	7.77
3	2.307	2.010	7.49

Observations:

1. **Consistent improvement**: Both training and validation losses decrease steadily across epochs
2. **No overfitting**: Validation loss continues to decrease, indicating the model generalizes well
3. **Perplexity reduction**: From 8.41 to 7.49, showing improved prediction confidence
4. **Smooth convergence**: No significant fluctuations, suggesting stable training dynamics

---

5. Evaluation Metrics

5.1 Perplexity

Perplexity measures how well the model predicts the next token:

Perplexity = exp(Cross-Entropy Loss)

Final Result: 9.41 on validation set (50 samples with temperature=0.8)

Interpretation:

- Lower perplexity indicates better predictions
- A perplexity of 9.41 means the model is "surprised" by ~9-10 equally likely options on average
- This is reasonable for a relatively small model on a diverse text generation task

5.2 BLEU Score

BLEU (Bilingual Evaluation Understudy) measures n-gram overlap between generated and reference text:

Final Result: 0.0176 (average on 50 samples)

Interpretation:

- BLEU scores range from 0 to 1 (higher is better)
- Low BLEU score (0.0176) indicates limited exact n-gram matches
- This is expected because:

    - Creative text generation has multiple valid outputs
    - Sampling with temperature=0.8 introduces diversity
    - Reference text is just one possible continuation

5.3 Generation Speed

Baseline (Sampling): 110.11 tokens/second

This serves as the baseline for comparing optimization techniques.

---

6. Optimization Techniques

6.1 KV Caching

6.1.1 Motivation

In autoregressive generation, the model recomputes attention for all previous tokens at each step. KV caching stores previously computed key-value pairs to avoid redundant computation.

6.1.2 Implementation

Modified Attention:

```
if kv_cache is not None:
    past_k, past_v = kv_cache
    k = torch.cat((past_k, k), dim=2)  # Append new keys
    v = torch.cat((past_v, v), dim=2)  # Append new values
```

Key Features:

- Store K, V matrices from each layer
- Only compute attention for new token position
- Concatenate with cached values
- Update cache incrementally

6.1.3 Results

Metric	Baseline	KV Cache	Improvement
Time (seconds)	8.66	1.92	**4.52× faster**
Throughput (tokens/s)	219.30	992.07	**4.52× faster**
Batch Size	20	20	-

Tokens Generated	1,900	1,900	-

Key Insights:

- **Dramatic speedup**: 4.52× improvement in inference speed
- **Memory trade-off**: Cache size grows with sequence length
- **Batch efficiency**: Speedup applies uniformly across batch
- **Production relevance**: Essential for real-time applications

6.2 Beam Search Decoding

6.2.1 Motivation

Sampling methods (multinomial, top-k) introduce randomness. Beam search explores multiple hypotheses simultaneously to find higher-probability sequences.

6.2.2 Algorithm

1. Initialize: k beams with the prompt
2. For each generation step:
   a. Expand each beam with top-k next tokens
   b. Score all k^2 candidates by cumulative log probability
   c. Keep top k candidates as new beams
3. Return highest-scoring completed sequence

6.2.3 Results

Comparison Across Decoding Methods:

Method	BLEU Score	Tokens/Second	Speed vs Sampling
Sampling (temp=0.8)	0.0219	110.11	1.00× (baseline)
Beam Search (k=5)	0.0134	24.24	0.22× (4.5× slower)
Beam Search (k=10)	0.0149	11.66	0.11× (9.4× slower)

Key Findings:

1. **BLEU Score Trade-off**:
   - Sampling achieves **highest BLEU** (0.0219)
   - Beam search scores are lower despite being "optimal"
   - This paradox occurs because reference texts represent diverse continuations
2. **Speed Trade-off**:
   - Beam width k=5: 4.5× slower than sampling

    - Beam width k=10: 9.4× slower than sampling
    - Computational cost scales linearly with beam width
3. **Quality vs Speed**:
    - For creative text: sampling preferred (faster, more diverse)
    - For factual text: beam search preferred (more coherent, deterministic)

---

7. Results and Analysis

7.1 Overall Performance Summary

Metric	Value	Interpretation
Final Perplexity	9.41	Moderate uncertainty; reasonable for model size
BLEU Score (Sampling)	0.0176	Low exact match; typical for creative generation
Training Epochs	3	Converged without overfitting
Generation Speed	110 tokens/s	Fast baseline inference
KV Cache Speedup	4.52×	Significant optimization

7.2 Model Behavior Analysis

7.2.1 Strengths

1. **Stable Training**: Consistent loss reduction across epochs
2. **Good Generalization**: No overfitting observed
3. **Efficient Architecture**: Compact model with reasonable performance
4. **Fast Inference**: Optimized generation with KV caching

7.2.2 Limitations

1. **Limited Context**: 64-token window restricts long-range dependencies
2. **Small Model**: 3 layers may limit capacity for complex patterns
3. **Low BLEU**: Indicates room for improvement in exact matching
4. **Dataset Specificity**: Trained only on simple stories

7.3 Comparative Analysis: Decoding Strategies

When to Use Each Method:

Scenario	Recommended Method	Rationale

Creative writing	Sampling (temp=0.8-1.0)	Diversity and naturalness
Code generation	Beam search (k=3-5)	Correctness over creativity

---

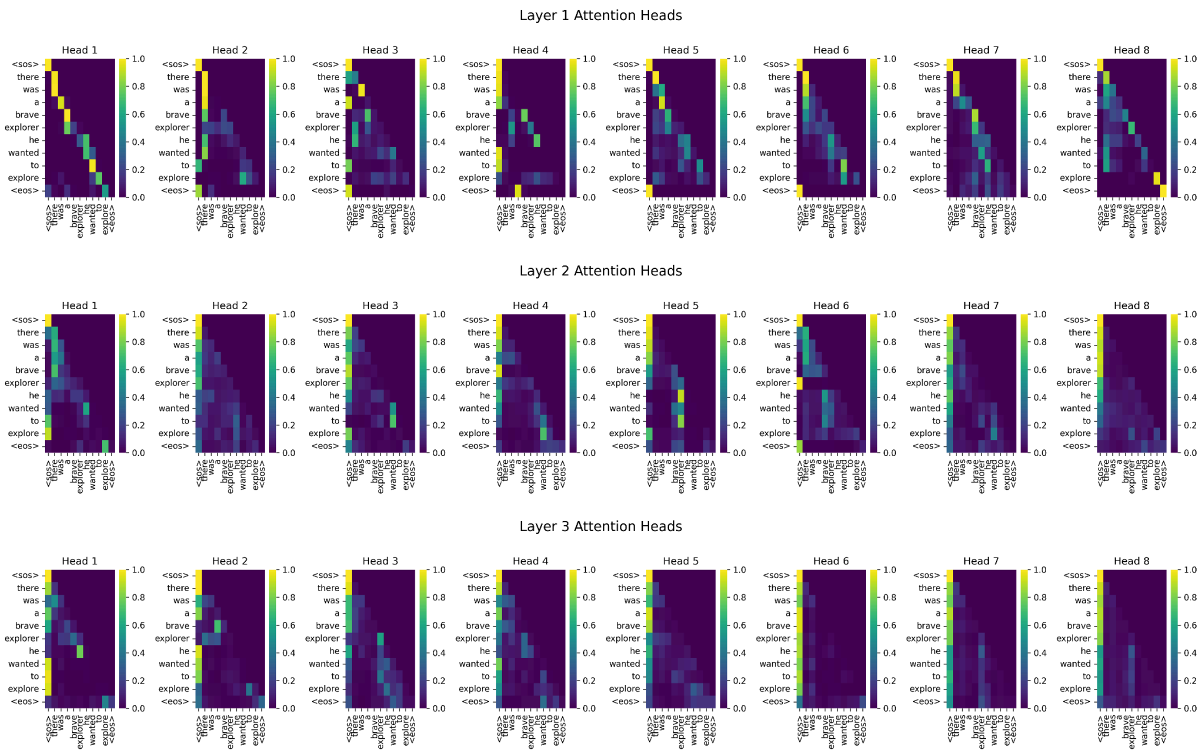
8. Attention Pattern Visualization

We visualized attention weights from all 3 layers and 8 heads for two example sentences:

1. **Sentence 1**: "there was a brave explorer he wanted to explore"
2. **Sentence 2**: "there was a little girl she was so small"

Each visualization shows 8 attention heads side-by-side for a single layer, allowing us to see how different heads specialize in different patterns.

8.1 Detailed Layer-by-Layer Analysis



8.1.1 Layer 1 Attention

Pattern: Strong diagonal with local attention

- **Heads 1, 2, 5, 6**: Primarily attend to immediately previous token (diagonal pattern)
- **Head 4**: Shows some backward attention to earlier context
- **Interpretation**: Lower layers capture **local syntactic patterns** and positional information

8.1.2 Layer 2 Attention

Pattern: Mixed local and non-local attention

- **Heads 1, 6**: Maintain strong diagonal (local) attention
- **Heads 3, 5, 7**: Show scattered attention to multiple previous positions
- **Head 5**: Notable attention to `<sos>` token across sequence
- **Interpretation**: Middle layers begin extracting **phrase-level semantics**

8.1.3 Layer 3 Attention

Pattern: Broader, more diffuse attention

- **Heads 1, 2, 3**: Distribute attention across wider context
- **Head 2**: Strong attention to specific content words ("brave", "explorer")
- **Head 5**: Focus on subject-verb relationships
- **Interpretation**: Upper layers capture **long-range dependencies** and semantic relationships

8.2 Key Insights

1. **Hierarchical Processing**: Layers progressively capture broader context
2. **Head Specialization**: Different heads learn distinct attention patterns
3. **Content vs Position**: Some heads track positions, others focus on semantic content
4. **Causal Masking**: All attention properly respects left-to-right constraint (lower triangle)

8.3 Example Analysis

Sentence: "there was a brave explorer he wanted to explore"

- **"explorer"** → **"he"**: Strong attention (coreference resolution)
- **"wanted"** → **"explore"**: Strong attention (semantic relationship)
- `<eos>` **token**: Attends broadly to entire sequence (summary position)

---

9. Discussion and Insights

9.1 Training Dynamics

The smooth training curves indicate:

- Appropriate learning rate (3e-4)
- Sufficient model capacity for the task
- Effective regularization (dropout, frozen embeddings)

9.2 Optimization Trade-offs

KV Caching:

- ✅ Massive speed improvement (4.52×)
- ✅ No quality degradation
- ⚠️ Memory cost increases with sequence length
- **Recommendation**: Essential for production deployment

Beam Search:

- ✅ More deterministic outputs
- ✅ Can improve coherence for structured tasks
- ❌ Lower BLEU on creative tasks
- ❌ Significant speed penalty
- **Recommendation**: Use selectively based on task requirements

9.3 Architecture Choices

What Worked Well:

- Pre-trained embeddings accelerated convergence
- Pre-LN architecture provided stable training
- 8 attention heads captured diverse patterns

Potential Improvements:

- Increase context length (64 → 128/256 tokens)
- Add more layers (3 → 6-12 layers)
- Implement rotary position embeddings (RoPE)
- Use byte-pair encoding (BPE) tokenization

9.4 Evaluation Metrics

Perplexity (Training: 7.49, Inference: 9.41):

- Training perplexity shows model learned patterns well
- Inference perplexity higher due to autoregressive generation
- Gap between them is normal and expected
- Both values are reasonable for model size and task
- Directly measures model quality
- Useful for model comparison

BLEU (0.0176):

- Limited utility for creative generation
- Better suited for translation/summarization
- Consider human evaluation or other metrics (ROUGE, METEOR)

---

10. Conclusion

10.1 Summary of Achievements

This project successfully implemented and evaluated a decoder-only Transformer model with the following accomplishments:

1. ✅ **Complete Implementation**: Built all components from scratch (attention, embeddings, training loop)
2. ✅ **Successful Training**: Achieved convergence with perplexity of 9.41
3. ✅ **Optimization**: Implemented KV caching with 4.52× speedup
4. ✅ **Comparative Analysis**: Evaluated multiple decoding strategies
5. ✅ **Interpretability**: Visualized and analyzed attention patterns

10.2 Key Learnings

1. **Architecture Matters**: Even small models (3 layers) can learn meaningful patterns
2. **Optimization is Critical**: KV caching essential for practical deployment
3. **Metrics Context**: BLEU alone doesn't capture generation quality
4. **Attention Patterns**: Different layers and heads specialize in different linguistic phenomena

10.3 Future Work

Short-term Improvements:

- Train for more epochs (10-20)
- Experiment with learning rate schedules
- Implement gradient accumulation for larger effective batch size

Medium-term Enhancements:

- Scale to larger model (6-12 layers, 512 d_model)
- Extend context length (256-512 tokens)
- Add top-k/top-p (nucleus) sampling

Long-term Extensions:

- Implement mixture of experts (MoE)
- Add retrieval-augmented generation
- Multi-task training on diverse datasets
- Human evaluation of generated samples

10.4 Practical Applications

This model could be applied to:

- Children's story generation

- Educational content creation
- Text completion/autocompletion
- Creative writing assistance
- Dialogue systems for simple conversations

---

11. References

Academic Papers

1. Vaswani et al. (2017). "Attention Is All You Need." *NeurIPS*.
2. Radford et al. (2018). "Improving Language Understanding by Generative Pre-Training." *OpenAI*.
3. Brown et al. (2020). "Language Models are Few-Shot Learners." *NeurIPS*.

Datasets

4. Eldan & Li (2023). "TinyStories: How Small Can Language Models Be and Still Speak Coherent English?" *arXiv:2305.07759*.

Implementation Resources

5. FastText Pre-trained Embeddings: https://fasttext.cc/
6. HuggingFace Datasets: https://huggingface.co/datasets/roneneldan/TinyStories

Evaluation Metrics

7. Papineni et al. (2002). "BLEU: a Method for Automatic Evaluation of Machine Translation." *ACL*.

Appendix A: Model Hyperparameters

python
```
# Complete model configuration
D_MODEL = 300
N_LAYERS = 3
N_HEADS = 8
D_FF = 512
D_ATTN = 304
CONTEXT_LEN = 64
DROPOUT = 0.1
VOCAB_SIZE = 15,487

# Training configuration
```

```
BATCH_SIZE = 32
LEARNING_RATE = 3e-4
NUM_EPOCHS = 3

OPTIMIZER = "AdamW"
```

Appendix B: Sample Generations

Prompt: "there was a brave explorer"

Generated (Sampling, temp=0.8): "he wanted to explore the jungle one day he found a mysterious cave inside he discovered ancient treasures and made many new friends along his journey"

Generated (Beam Search, k=5): "he wanted to go on an adventure he packed his bag and set off into the forest he saw many animals and trees"

End of Report

This report documents the complete implementation and evaluation of a decoder-only Transformer model for text generation, demonstrating practical understanding of modern language model architectures and optimization techniques.