Building and Training a 50-Million-Parameter Decoder-Only Language Model on Consumer Hardware

William Peytz

August 2026

Abstract

This report describes the design and implementation of a 50.8-million-parameter decoder-only Transformer language model in PyTorch. The system was built from first principles for a Windows development environment and trained on an NVIDIA GeForce RTX 2070. It includes a byte-level tokenizer, an in-memory data pipeline, causal self-attention, next-token training, validation, mixed-precision execution, checkpointing, resumption, and autoregressive generation. A 250 MB subset of TinyStories containing 305,373 complete stories was prepared for the experiment. A 10,000-step run processed approximately 20.48 million tokens in 3,862.4 seconds. The final training and validation evaluation losses were 0.6359 and 0.6886, while the lowest sampled validation loss was 0.5985 at step 9,900. Generated text exhibited story structure, dialogue, punctuation, and a learned end-of-story marker, but retained semantic inconsistencies. The report documents the design decisions, results, limitations, and steps required for reproducibility.

1 Introduction

Large language models learn a probability distribution over token sequences by predicting the next token from preceding context. Although contemporary systems may contain billions of parameters and require distributed infrastructure, a smaller model retains the central algorithmic components and can be studied on consumer hardware. The objective of this project was to construct a complete, understandable language-model training system rather than depend on a high-level model framework.

The resulting implementation targets approximately 50 million trainable parameters. It uses a GPT-style decoder-only Transformer derived from the causal self-attention architecture introduced by Vaswani et al. [6]. The project emphasizes a small dependency set, commands that run directly from PowerShell, and transparent implementations of model, data, optimization, evaluation, checkpoint, and generation logic.

2 System Design

2.1 Tokenization and data representation

The tokenizer maps UTF-8 text directly to byte values in the range 0–255. Consequently, the vocabulary contains exactly 256 tokens and can represent any UTF-8 input without an unknown-token symbol or a separately trained vocabulary. This choice simplifies reproducibility and eliminates tokenizer dependencies. Its main disadvantage is sequence inefficiency: many non-ASCII characters and some common text fragments require multiple tokens, while a learned subword tokenizer could represent them more compactly.

For a token sequence $x_1, \ldots, x_T$, training minimizes the autoregressive negative log-likelihood

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t \mid x_1, \ldots, x_{t-1}). \tag{1}$$

The implementation samples random contiguous contexts from the training or validation split. Inputs contain all but the final token in a sampled window; targets are the same window shifted by one position.

2.2 Model architecture

The network contains learned token and position embeddings followed by 12 pre-normalized Transformer blocks. Each block contains eight-head causal self-attention, a four-times-expanded feed-forward network with GELU activation, residual connections, and layer normalization. PyTorch's scaled dot-product attention operation applies the causal mask and selects an efficient available kernel. The final layer normalization feeds a linear language-model head. Input embedding and output-head weights are tied.

Table 1: Default architecture.

Component	Value
Vocabulary size	256 bytes
Context length	256 tokens
Transformer layers	12
Embedding width	592
Attention heads	8
Per-head width	74
Feed-forward width	2,368
Dropout	0.0
Linear-layer bias	Disabled
Trainable parameters	50,784,720

The parameter count was computed directly by summing the number of elements in every trainable tensor. For one block, the attention and feed-forward weight matrices contribute $12d^2$ parameters, and its two layer-normalization operations contribute $2d$, where $d = 592$. Across 12 blocks, with tied token/output embeddings, learned position embeddings, and a final layer normalization, the total is

$$12(12d^2 + 2d) + 256d + 256d + d = 50{,}784{,}720. \tag{2}$$

2.3 Optimization and execution

Weights are optimized with AdamW [3], using a peak learning rate of 3×10^{-4}, weight decay of 0.1, gradient clipping at 1.0, 100 warmup steps, and cosine decay toward 3×10^{-5}. The default micro-batch contains four sequences, but the RTX 2070 experiment used one sequence to reduce memory pressure. Eight gradient-accumulation passes therefore produce an effective batch of eight sequences, or 2,048 tokens per optimizer step at the default context length.

The runtime automatically selects CUDA when available and supports FP32, FP16, and BF16 execution. Checkpoints contain model parameters, optimizer state, current step, configuration, and validation loss. A saved run can therefore resume without discarding optimizer history.

3 Related Work

The model follows the decoder-only autoregressive approach popularized by the GPT family [5], while using the Transformer attention mechanism of Vaswani et al. [6]. TinyStories demonstrated that carefully constrained synthetic data can enable very small models to produce coherent

English narratives [2]. The present work differs in emphasis: it documents a compact, from-scratch PyTorch implementation and a reproducible consumer-GPU run rather than proposing a new architecture or dataset. PyTorch provides the tensor, automatic-differentiation, optimizer, and scaled-attention primitives used by the implementation [4].

4 Dataset

TinyStories was introduced to study how small language models can produce coherent English when trained on a constrained synthetic corpus [2]. The present experiment uses the first 249,999,890 bytes of `TinyStoriesV2-GPT4-train.txt`, downloaded from the official Hugging Face repository at revision `f54c09fd23315a6f9c86f9dc80f725de7d8f9c64` on August 12, 2026. The subset was truncated backward to a complete `<|endoftext|>` boundary. This marker is ordinary multi-byte text under the byte tokenizer, not a single special token. The resulting UTF-8 file contains 305,373 complete stories. A contiguous 90/10 split is applied by the training program. The dataset card identifies the license as CDLA-Sharing-1.0 and describes the V2 file as GPT-4-generated [1].

This subset was selected because its simple vocabulary and narrative structure are appropriate for a 50-million-parameter model. It is not representative of general web text, technical knowledge, dialogue, or diverse writing styles. As a synthetic corpus, it can reproduce biases, stereotypes, or undesirable patterns inherited from its generators and prompts. Results should therefore be interpreted as story-domain language modeling rather than general-purpose language understanding, and generated text should not be assumed safe or factual.

5 Experimental Setup

The experiment was conducted on Windows with an NVIDIA GeForce RTX 2070 containing 8 GB of graphics memory and NVIDIA driver 560.94. Python 3.11.7 and PyTorch 2.13.0+cu126 with the CUDA 12.6 runtime were installed in a project-local virtual environment. Automatic dtype selection chose BF16 for the completed run. CUDA availability and the GPU name were verified before training. A CPU smoke-test configuration first exercised data loading, forward and backward propagation, evaluation, checkpoint writing, checkpoint loading, and text generation using a reduced model. CPU model, system-memory capacity, and peak GPU-memory consumption were not recorded during the experiment and cannot be reconstructed reliably after completion.

Full-model training used a context length of 256, micro-batch size one, and eight gradient-accumulation passes. Evaluation sampled 20 batches from both training and validation splits every 100 steps. The reported step timer is cumulative from training-loop startup and includes work completed before the log line; periodic evaluation and checkpoint writing add further wall-clock overhead.

6 Training Results

The run completed 10,000 optimization steps in 3,862.4 seconds (64 minutes and 22 seconds). With 2,048 sampled tokens per optimizer step, the model processed approximately 20.48 million tokens at an average end-to-end throughput of about 5,300 tokens per second. This corresponds to roughly 9% of the 90% training portion of the prepared corpus; contexts were sampled randomly and therefore do not constitute a non-repeating pass through that portion.

Table 2: Selected evaluation measurements during the 10,000-step run.

Step	Training loss	Validation loss
0	5.5452 (theoretical uniform baseline)	
100	2.3819	2.3879
1,000	1.1637	1.2016
2,500	0.9367	0.8938
5,000	0.7625	0.6790
7,500	0.7094	0.6697
9,000	0.6325	0.6809
9,900	0.6683	**0.5985**
9,999	0.6359	0.6886

The final validation loss corresponds to a byte-level perplexity of approximately $\exp(0.6886) = 1.99$. The lowest observed validation loss, 0.5985, corresponds to approximately 1.82 perplexity. Each evaluation averages only 20 randomly sampled batches, so the individual values are noisy and the minimum should not be interpreted as a stable whole-dataset estimate. The final checkpoint overwrote the step-9,900 checkpoint; consequently, the retained `latest.pt` represents step 9,999 rather than the lowest sampled validation result. Across the run, training and validation losses remained broadly similar, with no clear sustained divergence indicative of severe overfitting.

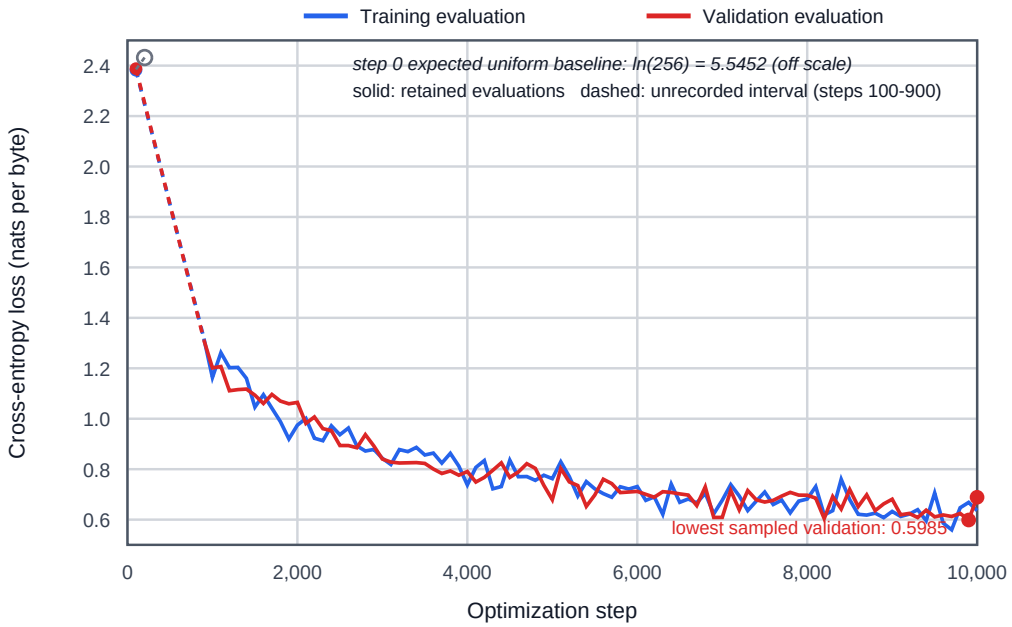

Figure 1: Training and validation loss. Step 0 is the theoretical uniform-prediction baseline $\ln(256) = 5.5452$ and is annotated off scale; step 100 is measured. The dashed interval denotes missing evaluations from steps 200–800, and the solid curves contain 92 retained evaluations from step 900 onward. Each evaluation averages 20 randomly sampled batches.

7 Qualitative Generation

The final checkpoint was sampled with temperature 0.7 and top-k 40. Given the prompt *"Once upon a time, a little girl found"*, the model generated the following first story before producing the learned dataset separator:

> Once upon a time, a little girl found a small sound in her yard. She loved to wear a lot of yummy feathers. One day, she wanted to find something to eat. She asked her mom, "Mom, can you help me find more?"
>
> Her mom said, "Yes, you can find something fun." She sat down and went to the store and took the yummy treat. She was so happy, she wanted to join the treat and eat and look around the park.
>
> `<|endoftext|>`

The sample demonstrates several learned properties: it continues the prompt fluently, maintains a simple narrative register, introduces dialogue, uses mostly correct punctuation, and emits the exact end-of-story marker found in the training corpus. However, phrases such as "small sound," "yummy feathers," and "join the treat" reveal weak semantic consistency. Generation continued into a second story after the marker because the sampling program did not treat the multi-byte marker as a stopping sequence.

To reduce reliance on a single favorable example, three additional unseeded stochastic samples were generated from the same final checkpoint with temperature 0.7 and top-k 40. Their exact random states were not recorded, so the outputs are genuine but not exactly reproducible. Text after the first boundary marker is omitted.

> **Prompt:** The dragon was afraid because
>
> **Output:** The dragon was afraid because he was happy. He saw his friends smiling and said, "Thank you, Bob! You are very nice and kind." The dragon made a fun day at the park.
>
> **Prompt:** A small robot wanted to make a friend
>
> **Output:** A small robot wanted to make a friend of a nice place to each other, but he didn't give up. He was a bit selfish because he was playing outside to play. Tim went to his friend Sam and his friends asked him to wait. Sam was so happy to play with his toys and stayed in the playground. He played with his toys and had a lot of fun on the ground.
>
> **Prompt:** Lucy learned an important lesson when
>
> **Output:** Lucy learned an important lesson when she needed help. She was a playful day for the clumsy clumsy clumsy boy.

All four samples reproduce the short-story register and all four reach the learned boundary within the sampling budget. The dragon example is locally fluent but logically contradicts its prompt; the robot example loses its initial subject; and the Lucy example repeats a word and assigns the wrong semantic type to a person. These deliberately retained weak cases show that surface fluency exceeds narrative and semantic consistency. Because sampling was stochastic and only four prompts were examined, this remains a small qualitative suite rather than a statistically reliable benchmark.

8 Limitations

The current data pipeline reads the entire corpus into memory and converts tokens to 64-bit integers. This is convenient for a 250 MB experiment but inefficient for larger corpora; streaming or memory-mapped compact token storage would substantially reduce peak system-memory use. Random batch sampling does not guarantee that each example is observed exactly once per epoch. The contiguous validation split can also differ systematically from earlier portions of the source file. Evaluation on only 20 random batches produces visibly noisy validation estimates. During the reported run, checkpointing retained only the most recent model rather than the best validation model; the implementation was subsequently updated to save both `latest.pt` and `best.pt` and to log future evaluations to CSV.

The byte vocabulary trades simplicity for longer sequences. The model uses learned absolute positional embeddings and is limited to its configured 256-token context. No instruction tuning,

preference optimization, retrieval system, safety tuning, or factuality evaluation is included. Finally, the run processed only about 9% as many sampled tokens as the size of the training split, and qualitative evidence comes from only four stochastic examples; broad claims about generation quality are therefore not warranted.

9 Reproducibility

The repository separates configuration, tokenizer, data pipeline, model, training, inspection, and generation into small Python modules. The principal commands are:

```
python inspect_model.py
python train.py --smoke-test --device cuda
python train.py --device cuda ‘
  --data D:\GPT-50M\data\tinystories-250mb.txt ‘
  --batch-size 1 --max-steps 10000
python generate.py --checkpoint checkpoints\latest.pt ‘
  --prompt "Once upon a time" --tokens 200 --seed 1337 --device cuda
```

The random seed defaults to 1337. Exact GPU results may nevertheless vary because hardware kernels and mixed-precision execution are not guaranteed to be bitwise deterministic. At the time of writing, the code exists in the local project directory but has not been assigned a public repository URL; long-term independent reproducibility would benefit from a versioned public archive.

10 Conclusion

This project demonstrates an end-to-end decoder-only language-model implementation with 50,784,720 parameters on consumer Windows hardware. In 10,000 steps, validation loss fell from 2.3879 at step 100 to 0.6886 at the final evaluation, and the completed run took approximately 64 minutes. Four generated samples showed recognizable narrative structure and learned the corpus boundary marker, while also exposing semantic limitations expected from the model size, byte-level representation, and limited training exposure. Post-experiment improvements now preserve structured metric logs, save both latest and best-validation checkpoints, and stop displayed generation at the story marker. The principal next steps are to evaluate on a fixed and larger validation sample, compare the fixed prompt suite across future checkpoints, publish a versioned code archive, and replace the in-memory loader before scaling the corpus.

References

[1] Ronen Eldan and Yuanzhi Li. Tinystories dataset card. Hugging Face Datasets, 2023. Revision f54c09fd23315a6f9c86f9dc80f725de7d8f9c64; CDLA-Sharing-1.0.

[2] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2019.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.