Training Tiny Manas: A Small Language Model for a Kyrgyz Epic

Nikita Nosov
https://nik1t7n.com
Technical report · September 5, 2026

Abstract

Tiny Manas is a 26.78-million-parameter autoregressive language model trained from scratch on a Kyrgyz edition of the epic *Manas*. The model uses eight pre-normalized Transformer blocks, rotary position embeddings, a tied 32,768-entry byte pair encoding vocabulary, and a 256-token context. Training uses 418,562 source tokens and 12.288 million sampled target positions on an Apple M5 with 16 GB of unified memory. Controlled comparisons support the use of additional model capacity and rotary positions: the latter reduce validation perplexity from 77.15 to 61.30 under the same training recipe. Bfloat16 training reduces measured training-loop time by 20.58% with nearly unchanged validation loss; inference uses FP32 and a request-local key/value cache. The final model achieves test perplexity 92.88 on the original source under random-window evaluation. Additional book-level evaluation reveals weak transfer to differently formatted editions. Training on an expanded corpus substantially improves new-book prediction but slightly degrades familiar-source prediction, with line formatting accounting for a large part of the measured gain. The results characterize the interaction of model design, limited data, and hardware-specific execution. Generated text reproduces local epic vocabulary and phrasing but does not sustain reliable narrative coherence.

1 Introduction

Training a small language model on a single literary source makes the relationship between data, architecture, and generated text directly inspectable. The restricted setting also makes its limitations consequential: repeated exposure to a short corpus can improve next-token prediction without providing broad linguistic coverage or reliable long-form generation. Tiny Manas studies this setting using the Kyrgyz epic *Manas* and a training workload that fits on a personal computer.

The model is a decoder-only Transformer based on established components [1, 2]. Its design combines rotary position embeddings (RoPE), LayerNorm, multi-head causal attention, Gaussian error linear unit (GELU) feed-forward networks, and tied input/output embeddings. A fixed byte-level tokenizer represents Kyrgyz text without an unknown-token class. The implementation includes data preparation, next-token training, evaluation, and cached autoregressive generation; nanoGPT [11] provides an implementation reference for the compact decoder and optimizer organization.

This report describes the resulting model and the evidence supporting its engineering choices. Architecture comparisons measure prediction at matched training budgets, while execution comparisons measure latency or allocation under specified shapes. Separate corpus comparisons examine how additional editions affect familiar-source prediction and transfer. The study does not introduce a new Transformer architecture or establish state-of-the-art Kyrgyz performance. Its empirical scope is a single-seed, small-data implementation on Apple Silicon. Source code, configurations, measurement definitions, and numerical records are available in the Tiny Manas repository.

Preprint.

2 Model architecture

Tiny Manas maps a sequence of token IDs to a distribution over the next token at every position. Eight Transformer blocks transform the input embeddings into context-dependent representations. Each block combines a causal attention sublayer, which mixes information across preceding positions, with a feed-forward sublayer, which transforms features independently at each position. Layer normalization precedes both sublayers, and residual connections add each sublayer's output to its input. Table 1 specifies the model; Figure 1 shows its computation.

Table 1: Tiny Manas architecture. The vocabulary projection shares the token embedding matrix; RoPE has no learned position table.

Component	Configuration
Trainable parameters	26,779,392
Vocabulary / context	32,768 entries / 256 tokens
Decoder blocks / residual width	8 / 384
Attention	8 query heads and 8 key/value heads; 48 features per head
Position representation	RoPE on queries and keys; frequency base 10,000
Feed-forward network	$384 \to 1,536 \to 384$, GELU activation
Normalization	Pre-LayerNorm in each sublayer; final LayerNorm
Regularization	Drop probability 0.2 at embeddings, attention weights, and sublayer outputs
Vocabulary head	Bias-free linear projection tied to input embeddings

2.1 Embeddings and causal attention

Let B denote the microbatch size, $T \le 256$ the sequence length, $C = 384$ the residual width, $L = 8$ the number of blocks, and $H = 8$ the number of attention heads. For vocabulary $\mathcal{V}$ with $|\mathcal{V}| = 32768$, a lookup into $E \in \mathbb{R}^{32768\times 384}$ maps token IDs to $X \in \mathbb{R}^{B\times T\times 384}$. Embedding dropout acts on these vectors before the first block. Position information enters attention through RoPE rather than an additive position embedding.

For the normalized block input $Z = \mathrm{LN}(X)$, the query, key, and value projections are

$$Q = ZW_Q + b_Q, \qquad K = ZW_K + b_K, \qquad V = ZW_V + b_V. \tag{1}$$

The implementation packs the three projections into one linear operation, then reshapes each output into (B, H, T, d) with $d = C/H = 48$. Every head projects all 384 input features into its own 48-dimensional query, key, and value spaces. These vectors are activations derived from the current context, not fixed vocabulary entries.

RoPE [7] rotates adjacent feature pairs in queries and keys. For pair j at position m, the angle is $m\omega_j$, where $\omega_j = 10000^{-2j/d}$ and $j = 0, \dots, d/2 - 1$. Using column notation for an individual vector,

$$\begin{pmatrix} \widetilde{u}_{2j} \\ \widetilde{u}_{2j+1} \end{pmatrix} = \begin{pmatrix} \cos(m\omega_j) & -\sin(m\omega_j) \\ \sin(m\omega_j) & \cos(m\omega_j) \end{pmatrix} \begin{pmatrix} u_{2j} \\ u_{2j+1} \end{pmatrix}, \qquad u \in \{q, k\}. \tag{2}$$

If R_m denotes the full block-diagonal rotation, then $(R_m q)^\top (R_n k) = q^\top R_{n-m} k$. The dot product therefore contains a relative-position transformation. The implementation uses non-trainable sine/cosine buffers and 32-bit floating-point (FP32) rotation arithmetic before restoring the incoming query/key dtype. RoPE removes the learned position table but does not establish usable context beyond the trained 256-token window.

Within each head, attention computes

$$A = \mathrm{softmax}\left(\frac{\widetilde{Q}\widetilde{K}^\top}{\sqrt{48}} + M\right), \qquad U = \mathcal{D}_{0.2}(A)V, \qquad M_{ij} = \begin{cases} 0 & j \le i, \\ -\infty & j > i. \end{cases} \tag{3}$$

Softmax normalizes over key positions. The causal mask prevents a position from accessing later tokens during training, so the predictions satisfy the autoregressive objective. The scale factor controls the magnitude of the dot products before softmax [1]. Concatenating the eight head outputs restores width 384; a learned output projection mixes their features. The native implementation uses PyTorch scaled dot-product attention on the Metal Performance Shaders (MPS) backend.

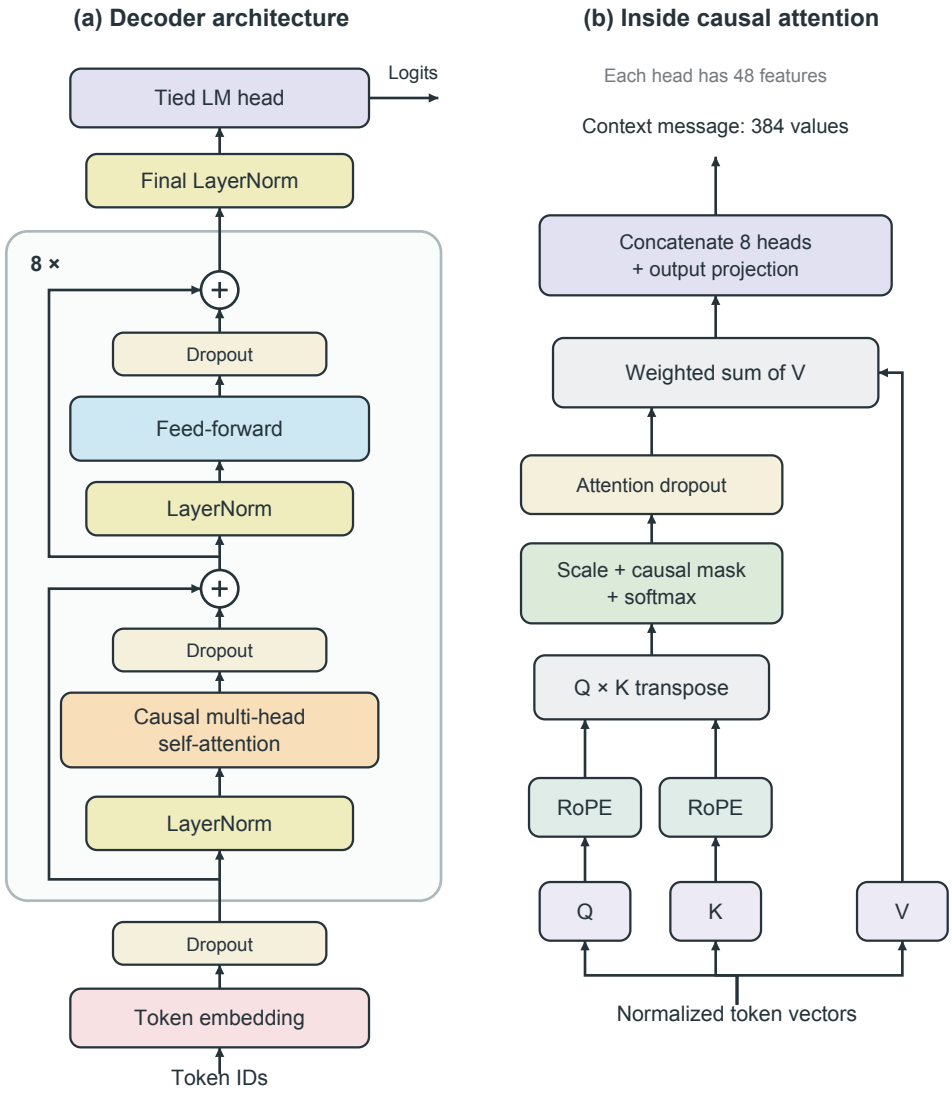

Figure 1: Tiny Manas decoder. (a) Residual connections bypass normalization and the complete sublayer. (b) RoPE transforms query and key vectors before attention; values are not rotated. Attention-weight dropout and attention-output dropout are separate operations. All dropout is disabled for evaluation and generation.

2.2 Feature transformation and regularization

LayerNorm [4] normalizes features independently at each token position:

$$\mathrm{LN}(x) = \gamma \odot \frac{x - \mu(x)}{\sqrt{\sigma^2(x) + 10^{-5}}} + \beta, \tag{4}$$

where $\mu(x)$ and $\sigma^2(x)$ are the mean and variance across the 384 features, and γ, β are learned parameters. The two residual updates in a block are

$$X' = X + \mathcal{D}_{0.2}(\mathrm{Concat}(U_1, \ldots, U_8)W_O + b_O)\,, \tag{5}$$

$$X^{\text{next}} = X' + \mathcal{D}_{0.2}(\mathrm{GELU}(\mathrm{LN}(X')W_1 + b_1)W_2 + b_2)\,. \tag{6}$$

The feed-forward network expands 384 features to 1,536 and projects back to 384. Unlike attention, it has no direct interaction between token positions. Normalizing the input to each sublayer leaves a direct residual path around its transformation.

Dropout [3] regularizes embeddings, attention weights, attention output, and feed-forward output. For drop probability $p = 0.2$, the training operation is $\mathcal{D}_p(z) = m \odot z/(1-p)$ with independent $m_i \sim \text{Bernoulli}(1-p)$. This scaling preserves the expectation; it does not preserve every realized activation or force an attention row to sum to one after dropout. Evaluation uses the identity operation.

A final LayerNorm precedes the language-model head. Its weight matrix is tied to E, giving logits $\text{LN}(X^{(8)})E^\top$, where $X^{(8)}$ is the output of the eighth block. Weight tying avoids a second 12,582,912-parameter vocabulary matrix. The logits cover all 32,768 entries even though only 9,593 vocabulary IDs appear in the original cleaned epic. Training converts these logits to a full-vocabulary cross-entropy loss; generation uses the final position to sample a continuation.

3 Data and training

3.1 Text representation and source corpus

The frozen byte-level byte pair encoding (BPE) tokenizer contains 256 base-byte entries and 32,512 learned entries. It has no special chat tokens and applies no text normalization. Byte representation permits an exact encode/decode round trip, including for text not represented by a single merged token.

Tokenizer training and language-model training use different corpora. The tokenizer corpus contains 188,208 documents and 524,284,895 UTF-8 bytes from FineWeb2 Kyrgyz text, Kyrgyz Wikipedia, the Kyrgyz News Corpus, and Manas-UdS. Its preparation uses normalized-text hashing and verified word-shingle similarity for duplicate filtering. These texts determine token boundaries; they do not train Tiny Manas's prediction weights. Because Manas material occurs in the tokenizer corpus, a passage held out from language-model training is not necessarily unseen during tokenizer construction.

The language-model corpus uses the `Manas01` document in Manas-UdS, identified in its metadata as a 2010 edition attributed to Sayakbai Karalaev. Preparation reads the surface-token column of the VRT source, reconstructs sentences, repairs punctuation spacing, and joins sentences with spaces. Introductory material before the verified epic-start heading is excluded. The resulting corpus contains 1,794,194 characters, 3,249,652 UTF-8 bytes, and 465,069 tokens. This representation does not retain verse-line or page boundaries.

The corpus is divided chronologically in a 90/5/5 ratio (Table 2). Training windows remain inside the training prefix; no window crosses into validation or test text. Token IDs are stored as little-endian unsigned 16-bit integers. Source and tokenizer hashes identify the exact inputs used by the training pipeline.

Table 2: Original language-model corpus. Splits are contiguous in source order rather than randomly interleaved.

Split	Tokens	UTF-8 bytes
Training	418,562	2,924,452
Validation	23,253	164,489
Test	23,254	160,711

3.2 Objective and optimization

Training samples a slice $(s_0, \ldots, s_T)$ and uses $(s_0, \ldots, s_{T-1})$ as input and $(s_1, \ldots, s_T)$ as targets. Every input position predicts its immediate successor. With parameters θ and natural logarithms, the objective is

$$\mathcal{L}(\theta) = -\frac{1}{BT} \sum_{b=1}^{B} \sum_{t=0}^{T-1} \log p_\theta(s_{b,t+1} \mid s_{b,0:t}). \tag{7}$$

Training uses the known preceding tokens rather than sampled model outputs. Backpropagation computes the gradient of the mean cross-entropy, and AdamW [5] updates the weights. Matrix parameters receive decoupled weight decay; one-dimensional bias and normalization parameters do not.

Each update accumulates gradients from two microbatches of eight 256-token sequences. Dividing each microbatch loss by two before backward yields an average over 4,096 target positions per update. A 3,000-update run therefore processes 12,288,000 sampled targets, approximately 29.36 times the training token count. Random windows can overlap, so this is repeated exposure rather than 12.3 million distinct tokens or a fixed number of sequential corpus passes.

Table 3: Training configuration for the final architecture. Parameters and optimizer states remain FP32 under BF16 autocast.

Setting	Value
Hardware / runtime	Apple M5, 16 GB unified memory; PyTorch 2.13.0, MPS
Budget / effective batch	3,000 updates / 4,096 target positions
Initialization / window seeds	1337 / 1338
Optimizer	AdamW; $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1
Learning rate	100-update linear warmup to 3×10^{-4}; cosine decay toward 3×10^{-5}
Gradient clipping	Global norm capped at 1.0 after accumulation
Initialization	Normal $\sigma = 0.02$; zero linear biases; residual projection scale $0.02/\sqrt{2L}$
Precision	BF16 forward autocast; FP32 parameters, optimizer states, and evaluation
Checkpoint selection	Lowest scheduled validation loss, evaluated every 100 updates

The settings in Table 3 apply to the main 27M comparisons unless a comparison explicitly changes the training protocol. The selected RoPE checkpoint occurs at update 2,900 within a completed 3,000-update run. Dropout is disabled during validation, and the optimizer is not involved in evaluation or generation.

3.3 Evaluation methodology

Scheduled validation uses 30 random batches per checkpoint. Additional validation and original-source test reporting use 100 batches with a separate sampler seed, giving 204,800 target positions for $B = 8, T = 256$. These batches resample the same suffix; overlapping windows are not independent observations. Perplexity is $\exp(\mathcal{L})$ and is comparable only when tokenization and scoring definitions match.

Comparisons across vocabulary sizes instead score the same source bytes. Bits per byte (BPB) is

$$\text{BPB} = \frac{\sum_{i \in \mathcal{S}} -\log p_\theta(s_i \mid \text{context}_i)}{\ln(2)\, N_{\text{UTF8}}}, \tag{8}$$

where $\mathcal{S}$ is the set of scored target positions and N_{UTF8} counts their original bytes. The vocabulary comparison scores 164,360 validation bytes after a common context-only prefix, using up to 256 context tokens and 128 target tokens per window. Each target is scored once. The additional book-level evaluation in Section 5.3 also uses fixed target positions and exact byte counts, but its prefix and target selection differ from this vocabulary comparison.

Architecture comparisons share initialization, training windows, and the specified learning-rate budget where possible. Some alternatives receive only a bounded training prefix or checkpoint adaptation; their results are identified accordingly. Measurements use one training seed. The original test suffix was evaluated in multiple phases of the project, so its score is a descriptive held-out result, not a new independent test of all adaptive design choices.

4 Computational implementation

4.1 Training precision and memory

Mixed precision accelerates eligible forward operations without converting the full training state to BF16. In two matched 3,000-update runs of the learned-position 27M architecture, training-loop time decreases from 1,548.04 seconds in FP32 to 1,229.41 seconds with BF16 autocast, a 20.58% reduction. Additional validation losses are 4.345733 and 4.345779, respectively. This comparison supports BF16 for training on the measured MPS workload; it does not imply the same benefit for single-token inference.

Memory savings depend on the measured quantity. After-update live tensor allocation samples decrease by 33.83% in the full training comparison, whereas driver allocation increases by 0.36%. These are allocator observations, not total-process or unified-memory peaks. At $B = 8, T = 256, |\mathcal{V}| = 32768$, a complete logits tensor alone contains 67,108,864 values: 128 MiB in BF16 or 256 MiB in FP32. Full-vocabulary cross-entropy remains practical for this model and batch geometry.

The implementation uses eager execution. A compiled-path comparison on the learned-position model measures 0.33343 seconds per warmed update versus 0.32365 seconds for eager execution. Activation checkpointing [6] is available but disabled: recomputing block activations reduces sampled live allocation from 2.10 to 1.29 billion bytes while increasing median update time from 0.317603 to 0.414878 seconds. Recomputing dropout requires preserving the device random-number state. Since the default workload fits in memory, the measured latency cost does not justify recomputation. These results concern the tested backend and tensor shapes rather than compilation or checkpointing in general.

4.2 Autoregressive decoding

Generation repeatedly evaluates the current context, samples a next token, and appends it. Temperature rescales the final-position logits before softmax; top-k sampling restricts the distribution to its k highest-scoring entries. The qualitative evaluation uses temperature 0.8 and $k = 40$. There is no instruction tuning, preference optimization, retrieval, or external language model.

Only the final position's vocabulary scores are required during generation. Slicing hidden states before the tied projection reduces output storage from $BT|\mathcal{V}|$ to $B|\mathcal{V}|$ values while leaving the training computation unchanged. For FP32 inference at $B = 1, T = 256$, this reduces the output from 32 MiB to 128 KiB. A forward microbenchmark on the learned-position model measures 7.307 milliseconds for all-position logits and 5.894 milliseconds for final-position logits. The saving decreases with shorter sequences and vanishes at $T = 1$.

The key/value (KV) cache avoids recomputing the preceding tokens while the context grows. Prefill processes the prompt and stores keys and values for every block. Subsequent decoding computes the new token's states, appends its keys and values, and attends to the cached history. With one FP32 request at the full context length, persistent storage is

$$2BLH_{\mathrm{KV}}Td \cdot 4 = 2 \cdot 1 \cdot 8 \cdot 8 \cdot 256 \cdot 48 \cdot 4 = 6{,}291{,}456 \text{ bytes.} \tag{9}$$

The factor two counts keys and values, and four is the byte width of FP32. Cache state belongs to one request. Figure 2 shows the transition from prefill to incremental decoding.

At overflow, simply deleting the oldest cache entries would not reproduce a fresh forward pass over the cropped window. Even with RoPE, retained deeper-layer states can contain information from evicted tokens. The implementation therefore rebuilds the cache from the cropped text. On the learned-position checkpoint, caching reduces the time for a 32-token prompt and 64 generated tokens from 0.19675 to 0.17011 seconds ($1.157\times$). For a 248-token prompt and 64 generated tokens, frequent rebuilding limits the measured speedup to $1.0165\times$. These are synchronized local model timings, not HTTP throughput.

The final RoPE model uses FP32 inference. On that checkpoint, BF16 autocast changes familiar-validation loss by only $+0.00002403$ nats; a 10,240-target probability comparison gives mean $D_{\mathrm{KL}}(P_{\mathrm{FP32}} \| P_{\mathrm{BF16}}) = 0.00007131$ and 98.94% top-1 agreement. However, short-prompt generation slows from 0.193243 to 0.229446 seconds (18.73%), while near-full-context generation improves only from 0.383007 to 0.370971 seconds (3.14%). The timing comparison uses three warmup pairs and 20 interleaved measured pairs per condition. BF16 halves persistent cache storage to 3 MiB, but sampled live allocation after 129 cached positions remains approximately 113.64 MB in both precisions. FP32 provides the preferable measured latency tradeoff for this small inference workload. These precision results are from MPS, not the CPU serving environment.

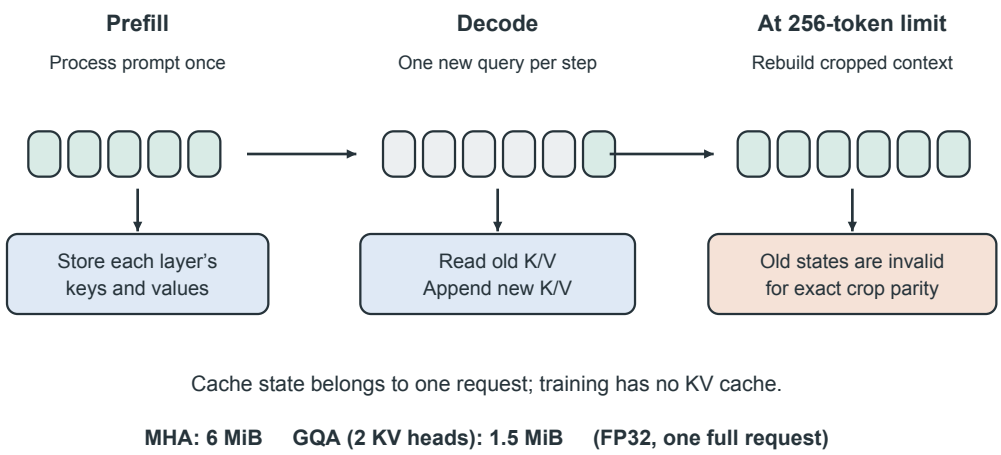

Figure 2: Decoding within a fixed context. Prefill stores each layer's keys and values; incremental decoding processes one new token. At the context limit, the implementation recomputes the cropped 256-token sequence. Sizes for multi-head attention (MHA) and grouped-query attention (GQA) refer only to persistent FP32 key/value storage.

5 Empirical results and design choices

5.1 Capacity, positions, and feature transformations

Increasing capacity improves original-source prediction more than extending context in the tested full-corpus recipe. With learned positions and FP32 training, increasing the model from six blocks of width 256 to eight blocks of width 384 reduces validation perplexity from 103.38 to 77.15 and test perplexity from 151.42 to 116.45. By contrast, extending the smaller model's context from 256 to 512 worsens both scores (Table 4). All three runs process 12,288,000 sampled targets in 3,000 updates. The 512-token configuration halves the microbatch size to retain that target budget, so it samples fewer independent windows per update.

Table 4: Capacity and context with learned position embeddings. All runs use FP32 and the same target-position budget. Validation and test use the additional random-window protocol.

Model / context	Parameters	Val. PPL	Test PPL	Loop min.
13M / 256	13,193,216	103.38	151.42	15.10
13M / 512	13,258,752	112.94	171.85	17.22
27M / 256	26,877,696	77.15	116.45	24.45

At the larger size, RoPE improves prediction without adding trainable parameters. Replacing the 256×384 learned position table removes 98,304 parameters. Under the matched BF16 recipe, validation loss decreases by 0.229944 nats and perplexity by 20.54% (Table 5). The test reduction is similar. A paired update-cost probe measures 0.36506 seconds for RoPE versus 0.33221 seconds for learned positions, so the quality gain comes with a modest per-update cost on this backend. The result supports rotary positions for the 256-token model; it does not establish long-context extrapolation.

LayerNorm and GELU remain the feature-transformation components. A parameter-matched SwiGLU alternative [9] uses hidden width 1,024 with three projection matrices in place of the two GELU matrices at width 1,536. After 3,000 updates, its additional-validation loss is 4.427423 versus 4.345779 for GELU, and its mean repeated-word-trigram ratio rises from 4.10% to 6.95%. RMSNorm [8], which rescales features without subtracting their mean, shows no material advantage over LayerNorm in a 900-update comparison: the mean of the last three validation-loss differences is $+0.003312$ nats. Its full-budget quality was not measured. Both comparisons use the learned-position architecture; they are not a factorial evaluation of these components together with RoPE.

Table 5: Position representation at the 27M scale. Both models use the same tokenizer, 256-token context, and completed 3,000-update BF16 recipe. Exact-byte validation BPB uses a different target protocol from random-window loss and perplexity.

Measurement	Learned positions	RoPE
Parameters	26,877,696	26,779,392
Validation loss	4.345779	4.115836
Validation perplexity	77.152	61.303
Exact-byte validation BPB	0.876442	0.827695
Test loss	4.757561	4.531258
Test perplexity	116.461	92.875

Standard multi-head attention also preserves prediction better than the tested grouped-query adaptation. Grouped-query attention (GQA) [10] reduces eight key/value heads to two by sharing them among query heads. Mean-pooling groups of four pretrained K/V projections reduces persistent FP32 cache storage from 6 to 1.5 MiB. After 150 matched adaptation updates, however, its fixed-window validation loss is 4.497047 versus 4.303030 for the unchanged multi-head control, with no measured decoding advantage. This result concerns short adaptation of a learned-position checkpoint; it does not characterize GQA trained from scratch.

5.2 Vocabulary size and text exposure

A smaller vocabulary reduces embedding storage but usually requires more tokens for the same text. Holding the number of optimizer updates fixed would therefore change source-text exposure. The vocabulary comparison instead uses 32k, 16k, and 8k nested BPE variants trained on the same tokenizer corpus, with identical source-byte split boundaries and 30 complete passes over the same scored training text. Each candidate processes 87,729,780 source bytes; learning rate is scheduled by text exposure rather than update count. Shared Transformer tensors start identically, while vocabulary size necessarily changes the embedding capacity and update count.

Table 6: Vocabulary comparison under equal source-text exposure. The random-window reference is the learned-position BF16 checkpoint, not the final RoPE model; it uses a different training recipe. All BPB values score the same 164,360 validation bytes.

Configuration	Params (M)	Updates	Val. BPB	Bytes/s
32k, random-window reference	26.878	3,000	0.87644	—
32k, equal-text training	26.878	3,090	0.92129	82,253
16k, equal-text training	20.586	3,420	0.91285	95,484
8k, equal-text training	17.441	3,840	0.91012	99,115

Throughput excludes evaluation, checkpoint writes, and generation.

Within the equal-text recipe, the 16k and 8k variants improve byte throughput by 16.09% and 20.50% and slightly improve BPB relative to fresh 32k training (Table 6). Neither reaches the random-window reference quality. The reference differs in sampling and scheduling, so this does not establish that 32k is intrinsically the best vocabulary size. It supports retaining the complete 32k random-window recipe for the reported model while showing that smaller vocabularies offer a useful resource tradeoff under controlled text exposure.

5.3 Additional editions and source transfer

Additional data consist of three Sagymbay Orozbakov volumes for training, a separate Orozbakov book for validation, and a Jusup Mamay edition for test, obtained from the Bizdin collection [12]. The source groups are fixed before model comparison (Table 7). Coordinate-based PDF extraction joins fragmented words, separates verse from smaller-type editorial notes, and removes page counters and watermarks. Unlike the original VRT representation, the resulting text preserves verse newlines. Exact encode/decode checks pass; residual OCR errors and related-edition overlap remain possible. Screening case-folded exact 32-word spans finds no spans requiring removal, but does not rule out shorter formulas or near-duplicates.

Table 7: Additional source groups. The expanded training pool combines the three Orozbakov volumes with the 418,562-token original training prefix.

Source	Role	Tokens
Orozbakov, book 2	Training	110,252
Orozbakov, book 3	Training	91,367
Orozbakov, book 5	Training	141,783
Orozbakov, book 4	Validation	99,630
Jusup Mamay	Test	408,359

The expanded pool contains 761,964 tokens, including 343,402 additional tokens. Half of each expanded-training microbatch comes from the original source and half from the additional volumes, sampled in proportion to eligible window starts without crossing source boundaries. A fresh original-data control and an expanded-data model use the same RoPE architecture, initial weights, BF16 recipe, and 3,000-update budget. This comparison changes content, source weighting, and line formatting jointly.

The new-book comparison scores 256 evenly spaced, nonoverlapping spans of up to 128 target tokens across Orozbakov book 4. A 512-token prefix supplies context only. Familiar-source evaluation scores the original validation remainder after the same prefix exclusion. With context 256, a full target span supplies between 129 and 256 preceding tokens per target. The models score identical targets, but these losses are not the random-window values reported in Table 5.

Table 8: Data composition at a matched 3,000-update budget. Both fresh models use RoPE and context 256. The final Tiny Manas checkpoint provides a separate familiar-source reference.

Model	New-book loss	Familiar-source loss
Tiny Manas	11.84687	4.08818
Original-data control	11.96507	4.09131
Expanded-data model	4.20787	4.13392

Expanded training sharply improves prediction on the new book but slightly worsens prediction on the familiar source (Table 8). Familiar loss exceeds the final Tiny Manas checkpoint by 0.045736 nats, more than the predefined 0.02-nat tolerance for replacing that model. Thus the reported Tiny Manas weights use the original corpus. The result shows a tradeoff under the selected mixture and budget rather than an unqualified benefit from adding books.

A post-hoc diagnostic partitions the 32,686 scored book targets into 4,976 newline-bearing targets and 27,710 other targets. Relative to the fresh original-data control, expanded training reduces mean loss on these groups from 25.1006 to 0.9245 and from 9.6063 to 4.7975. Direct newline-target loss accounts for approximately 47% of the aggregate reduction. Other targets also improve, but their input contexts contain line breaks, so the remaining reduction cannot be attributed solely to broader content or narrator diversity.

Figure 3 also includes a bounded RoPE context-length comparison on the original data. Four 512-token windows replace eight 256-token windows per microbatch, retaining 4,096 targets per update and the same learning-rate horizon. The 512-context run ends at 1,500 updates under a predefined stopping criterion requiring a consistent 0.02-nat improvement in new-book loss. At that point, book loss is 12.004335 using context 512 and 11.954920 using common context 256, versus 11.592824 for the 256-trained control. Familiar-source loss at common context 256 is 4.236962 versus 4.224035. This partial-budget result provides no evidence for extending the reported model's context; full-budget behavior remains unmeasured.

Whole-remainder evaluation of the final Tiny Manas checkpoint further exposes the source-transfer limitation (Table 9). Each split excludes a 512-token context-only prefix, then scores every retained target once with a 128-target stride. The Mamay edition is used only for post-selection reporting, and the expanded-data model is not evaluated on it. The original test gives perplexity 93.21 under this fixed-target protocol, compared with 92.88 under random-window evaluation of the same weights.

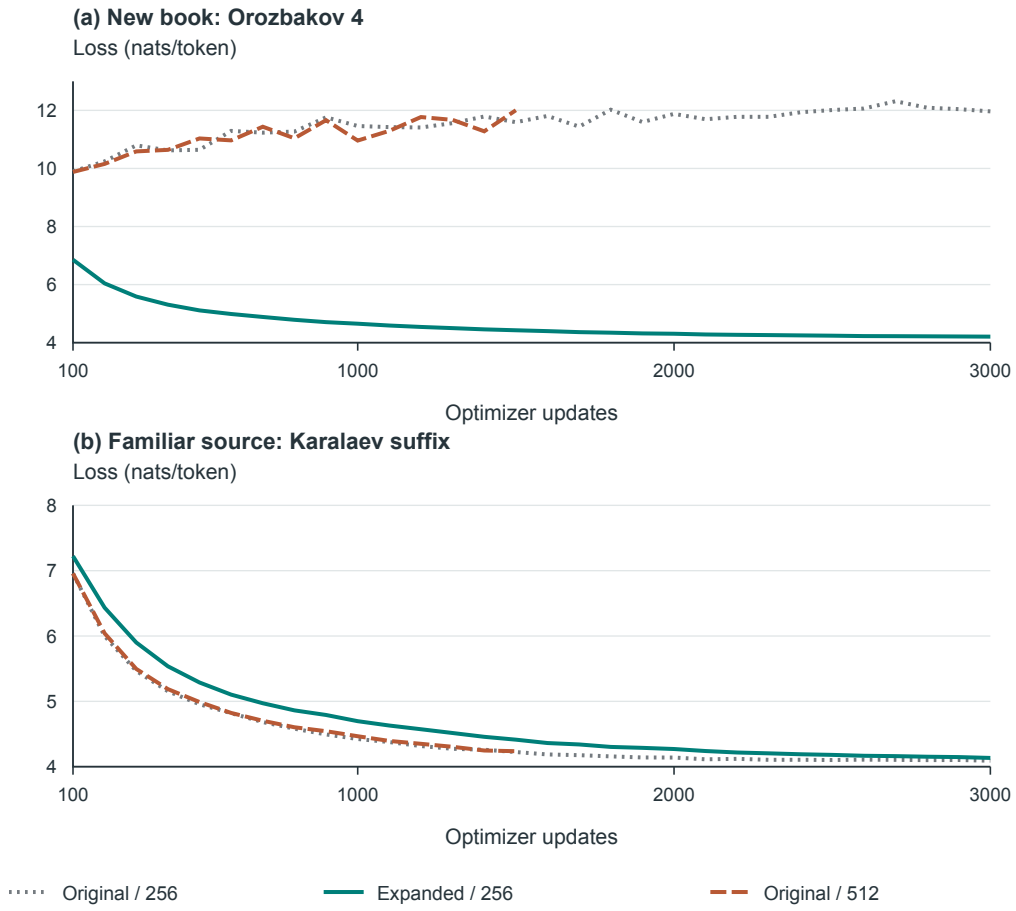

Figure 3: Prediction across source representations. Expanded training improves loss on the line-preserved Orozbakov book, while original-source validation favors the original-data recipe. The 512-context curve ends at its measured 1,500-update budget; the 256-context runs complete 3,000 updates. Curves connect observed scheduled evaluations without extrapolation.

Table 9: Final Tiny Manas model on complete held-out remainders. Exact target and byte counts define BPB. Cross-source differences include content, formatting, and extraction quality.

Source	Targets	Scored bytes	Loss	BPB
Original test suffix	22,742	156,951	4.534839	0.947984
Orozbakov 4, validation	99,118	611,514	11.842740	2.769318
Mamay, test	407,847	2,060,821	12.370524	3.531991

5.4 Generated text

The generation analysis uses five prompts and four sampling seeds, with temperature 0.8, top-k 40, and 256 new tokens per continuation. The 20 RoPE continuations contain epic-associated names, action phrases, and short dialogue fragments. They also exhibit abrupt actor changes, malformed endings, repeated titles, and loss of narrative continuity. These observations are based on the saved outputs rather than a blinded human evaluation.

The repeated-word-trigram ratio counts occurrences beyond each trigram's first occurrence, divided by all word-trigram occurrences. Its mean is 4.134% for RoPE and 4.097% for the learned-position comparison. Matching against the training text uses symmetric case-folding and punctuation-insensitive whole-word normalization. The maximum matching span is 11 words among the 20 RoPE continuations, compared with 9 words for learned positions. These measurements describe

repetition and local source overlap; neither a low ratio nor a short observed match proves coherence or absence of memorization.

The final model's original-test top-1 and top-5 accuracies are 31.70% and 48.53% under random-window evaluation. These are token-prediction accuracies, not reading-comprehension or factuality scores. The model's ability to reproduce local epic style is consistent with learning a narrow source distribution, while the external-book results and qualitative failures limit claims about general Kyrgyz language ability.

6 Limitations

The main training corpus contains one edition of one epic. Chronological held-out text remains close to that source, and the broader tokenizer corpus includes Manas material. Additional books introduce related editions, formulaic overlap, line-format differences, and OCR noise. The cross-source measurements therefore cannot separate linguistic transfer from representation shift. The expanded-data comparison also changes sampling weights; a formatting-matched corpus comparison would be needed to isolate the contribution of additional content.

All training comparisons use one seed, and validation is consulted repeatedly. The original test suffix has been examined across multiple phases. No multi-seed uncertainty intervals, external model benchmark, broad Kyrgyz task suite, or blinded native-speaker evaluation is available. The RMSNorm and RoPE context-512 results cover incomplete training budgets, and the GQA result covers short checkpoint adaptation. These findings support choices within the measured implementation, not a general ranking of architecture components.

Timing and memory measurements come from a shared Apple M5 host. Interleaving short comparisons reduces sensitivity to changing load, but does not remove it; sequential training runs can experience different temperature and background activity. Per-update, training-loop, and complete-generation timings have different measurement boundaries and are not pooled. Allocator samples are not peak system memory. Backend-specific speed results should not be transferred to other hardware without measurement.

The public repository contains code, configurations, hashes, and compact numerical records, but not the licensed corpus payloads or weight files. Manas-UdS source records specify CC BY-NC-SA 4.0; other source terms are recorded separately. Use of the additional editions is authorized for this student research, without a claim of a common open license across all scans. The report does not establish commercial rights to the texts or model weights. Tiny Manas is not a reliable factual reference, general assistant, or production-quality Kyrgyz writing system.

7 Conclusion

Tiny Manas implements a compact autoregressive Transformer for Kyrgyz epic text with 26.78 million parameters, a 32k byte-level BPE vocabulary, and a 256-token context. Its architecture uses RoPE with LayerNorm, GELU, and standard multi-head attention; training uses BF16 while decoding uses FP32 and an exact cropped-context KV cache. Controlled measurements support increased capacity and rotary positions for prediction quality and mixed precision for training efficiency. The model reaches original-source test perplexity 92.88 under random-window evaluation, but broader book-level evaluation reveals weak transfer. Additional training texts improve new-book prediction while introducing a familiar-source tradeoff and a substantial formatting effect. The resulting system demonstrates local next-token learning from limited Kyrgyz data, with narrative coherence and cross-source generalization remaining the principal limitations.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. NeurIPS, 2017. arXiv:1706.03762.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. OpenAI technical report, 2019. Original report.

[3] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research, 15:1929–1958, 2014. JMLR article.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. arXiv preprint, 2016. arXiv:1607.06450.

[5] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. ICLR, 2019. arXiv:1711.05101.

[6] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. *Training Deep Nets with Sublinear Memory Cost*. arXiv preprint, 2016. arXiv:1604.06174.

[7] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv preprint, 2023 (version 5; first posted 2021). arXiv:2104.09864v5.

[8] Biao Zhang and Rico Sennrich. *Root Mean Square Layer Normalization*. NeurIPS, 2019. arXiv:1910.07467.

[9] Noam Shazeer. *GLU Variants Improve Transformer*. arXiv preprint, 2020. arXiv:2002.05202.

[10] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. EMNLP, 2023. arXiv:2305.13245.

[11] Andrej Karpathy. *nanoGPT*. Software repository, accessed September 4, 2026. github.com/karpathy/nanoGPT.

[12] Bizdin. *Manas collection: editions attributed to Sagymbay Orozbakov and Jusup Mamay*. Digital source catalogue, accessed September 5, 2026. new.bizdin.kg/knigi/category/manas.