Nova: A Reproducible Modern Decoder-Only Transformer Implementation and Engineering Evaluation

Syed Sarim Ahsan
Department of Computer Engineering
Sir Syed University of Engineering and Technology
Karachi, Pakistan
sarimahsan101@gmail.com

Abstract

While foundational Transformer literature often focuses on historical multi-head attention and standard post-layer normalization paradigms, state-of-the-art decoder-only Large Language Models (LLMs) such as LLaMA, Mistral, and Qwen incorporate specialized architectural optimizations to maximize throughput and numerical stability. In this technical report, we present **Nova**, a self-contained, fully reproducible modern decoder-only Transformer language model (∼141M parameters) implemented entirely from scratch in PyTorch. Nova integrates RMSNorm, Rotary Position Embeddings (RoPE), Grouped Query Attention (GQA) with QK-Normalization, and gated SwiGLU feed-forward networks with tied embedding weights. We conduct an empirical engineering evaluation covering parameter efficiency, floating-point operation (FLOP) accounting, prefill/decode latency benchmarks, dynamic KV-cache memory footprint scaling, and qualitative story generation on the TinyStories dataset. Benchmarks on NVIDIA Tesla T4 GPU hardware demonstrate that Nova's GQA mechanism ($n_{\text{kv}} = 3$) reduces KV-cache memory consumption by 75.0% compared to standard Multi-Head Attention (saving 216.0 MB at batch size 16), while achieving an average generation speed of 33.03 tokens/s. Nova serves as an educational and reproducible reference implementation of a modern decoder-only language model.

Transformer, Large Language Models, PyTorch, Grouped Query Attention, Rotary Position Embeddings, RMSNorm, SwiGLU, TinyStories, Open Source.

1 Introduction

1.1 Motivation

The rapid evolution of autoregressive decoder-only language models has transitioned natural language processing from foundational sequence-to-sequence architectures [1] to industrial-scale foundation models capable of complex reasoning, instruction following, and code synthesis [2–6]. However, standard introductory tutorials and open-source educational codebases frequently rely on historical design choices, such as absolute learned positional embeddings, standard Multi-Head Attention (MHA), traditional Layer Normalization with learned bias parameters, and standard Feed-Forward Networks (FFN) utilizing ReLU or basic GELU activations.

Consequently, students, machine learning engineers, and researchers seeking to understand how production LLMs (e.g., LLaMA-3, Mistral 7B, Gemma, Qwen-2) function in practice face a disconnect between simplified academic tutorials and heavily optimized, highly complex distributed codebases. Nova was built to bridge this gap by offering a transparent, fully reproducible modern decoder-only Transformer implementation written cleanly from scratch in PyTorch.

1.2 Objectives

Nova aims to accomplish three primary engineering goals:

1. Provide an unencumbered, first-principles PyTorch reference implementation containing all modern architectural techniques without external black-box abstraction layers.
2. Rigorously document and benchmark the exact parameter efficiency, FLOP requirements, CUDA latency, and memory scaling behavior of modern LLM components.
3. Release open-source weights, dataset configurations, tokenizers, and automated evaluation suites to ensure full academic transparency and reproducibility.

1.3 Contributions

This technical report delivers the following core contributions:

- **Complete Implementation from Scratch:** Every core layer, including Rotary Position Embeddings (RoPE), Root Mean Square Normalization (RMSNorm), Grouped Query Attention (GQA) with QK-Normalization, and SwiGLU gated activations, is implemented natively in PyTorch.
- **Modern Transformer Integration:** A unified ∼141M parameter architecture incorporating modern production practices used by state-of-the-art open models.
- **Modular PyTorch Codebase:** A software architecture cleanly partitioned into decoupled modules (`components/`, `trainer/`, `callbacks/`, `configs/`, `tests/`, `utils/`).
- **Reproducible Training Pipeline:** End-to-end training details on TinyStories with AdamW, cosine decay, mixed-precision `bfloat16`, gradient accumulation, and multi-GPU `DataParallel`.
- **Comprehensive Engineering Evaluation:** Microsecond-accurate latency measurements (TTFT and TPOT), exact FLOP derivations, peak GPU VRAM tracking, and empirical demonstration of GQA's 75.0% KV-cache memory reduction.
- **Open-Source Release:** Fully trained checkpoint weights (`sarimahsan101/nova-141m-tinystories`) and reproducible evaluation scripts published on GitHub and Hugging Face.

2 Background

2.1 Decoder-Only Transformers

Unlike encoder-decoder structures designed for translation, decoder-only transformers perform causal autoregressive language modeling. Given a sequence of S token indices $\mathbf{x} = (x_1, x_2, \ldots, x_S)$, the model factorizes the joint probability of the sequence as the product of conditional next-token probabilities:

$$P(\mathbf{x}) = \prod_{t=1}^{S} P(x_t \mid x_1, x_2, \ldots, x_{t-1}) \tag{1}$$

Causal self-attention enforces that token representation $\mathbf{h}_t$ depends exclusively on preceding positions $\mathbf{h}_{\leq t}$ via a lower-triangular causal attention mask.

2.2 Nova Architecture Overview

Figure 1 presents the detailed layer-level composition of a single Nova Transformer block, and Figure 2 summarizes the corresponding high-level end-to-end data flow.

3 Nova Architecture

3.1 High-Level Architecture

Nova follows a pre-normalization causal Transformer decoder structure. Token representations pass through an embedding layer, 12 identical Transformer blocks, a final normalization layer, and a tied language modeling head. Figure 2 depicts the high-level computational flow.

3.2 Model Configuration

Table 1 summarizes the exact structural hyperparameter values defining Nova (∼141M).

3.3 Embedding Layer

Given token indices $\mathbf{x} \in \mathbb{N}^{B \times S}$, Nova maps tokens to continuous vectors via embedding matrix $W_{\text{embed}} \in \mathbb{R}^{V \times d_{\text{model}}}$ initialized with normal distribution $\mathcal{N}(0, 0.02^2)$:

$$\mathbf{h}_0 = \text{Dropout}\left(\text{Embedding}(\mathbf{x}; W_{\text{embed}})\right) \in \mathbb{R}^{B \times S \times d_{\text{model}}} \tag{2}$$

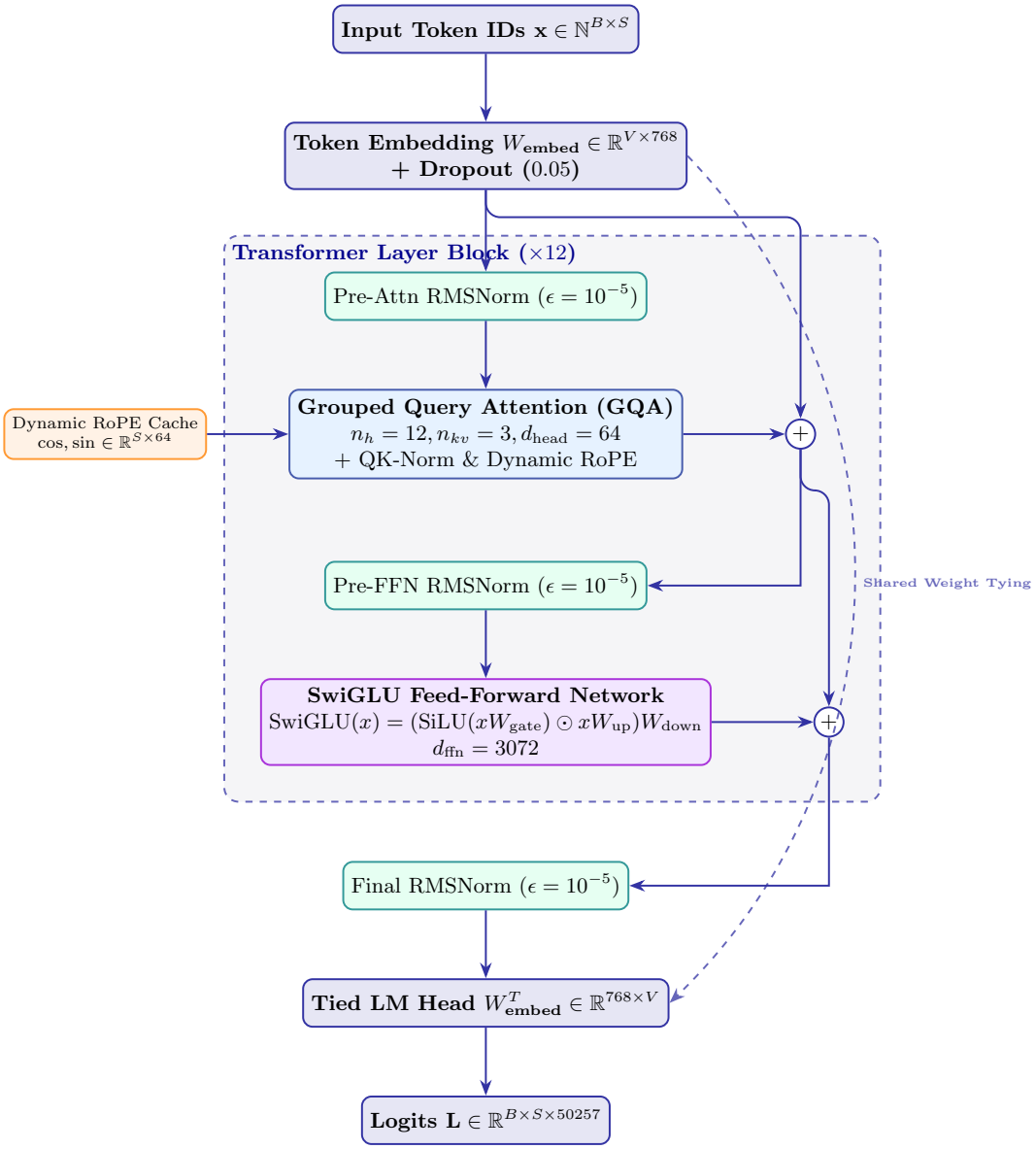

Figure 1: Detailed layer-level composition of a single Nova (∼141M) Transformer block, showing pre-norm RMSNorm layers, Grouped Query Attention ($n_{kv} = 3$) with dynamic RoPE cache insertion, gated SwiGLU FFN, dual residual additions, and the weight-tied LM head.

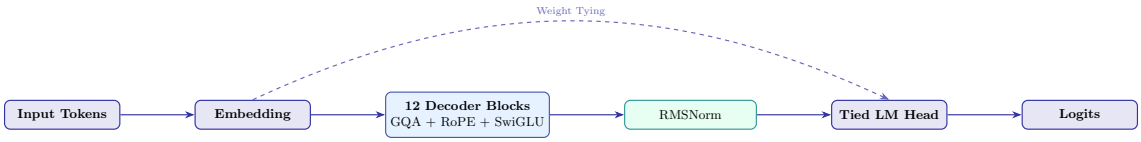

Figure 2: High-level overview of the proposed Nova decoder-only transformer architecture.

Table 1: Hyperparameter specifications of Nova (∼141M).

Hyperparameter	**Symbol**	**Nova Specification**
Vocabulary Size	V	50,257 (GPT-2 BPE)
Hidden Dimension	d_{model}	768
Number of Layers	L	12
Query Attention Heads	n_h	12
Key/Value Attention Heads	n_{kv}	3
Head Dimension	d_{head}	64
FFN Intermediate Dimension	d_{ffn}	3072
Max Context Length	$S_{\max}$	512
Positional Embedding	θ	Rotary (RoPE, $\theta = 10000$)
Normalization Layer	ϵ	RMSNorm ($\epsilon = 10^{-5}$)
Activation Function	σ	SwiGLU (SiLU Gate)
QK Normalization	$\hat{Q}$, $\hat{K}$	RMSNorm per Q/K head
Weight Tying	$W_{\text{embed}} = W_{\text{head}}$	Enabled
Total Parameters	N_{total}	141,247,488 (141.25M)
Non-Embedding Parameters	$N_{\text{non\_embed}}$	102,650,112 (102.65M)

3.4 Rotary Position Embeddings (RoPE)

Nova embeds position information dynamically into query and key projections using Rotary Position Embeddings [7]. For a vector $\mathbf{x}_m \in \mathbb{R}^{d_{\text{head}}}$ at sequence index m, RoPE groups elements into pairs and applies a 2D rotation matrix:

$$R^d_{\Theta,m}\mathbf{x}_m = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \dots \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \dots \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \dots \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \dots \end{pmatrix} \begin{pmatrix} x_{m,1} \\ x_{m,2} \\ x_{m,3} \\ x_{m,4} \end{pmatrix} \tag{3}$$

where frequency $\theta_i = 10000^{-2(i-1)/d_{\text{head}}}$. This guarantees that attention dot products $\langle R^d_{\Theta,m}\mathbf{q}_m, R^d_{\Theta,n}\mathbf{k}_n \rangle$ depend solely on relative offset $(m-n)$.

3.5 Grouped Query Attention (GQA)

Nova partitions 12 Query heads into 3 groups of 4, where each group shares a single Key and Value head ($n_h = 12, n_{\text{kv}} = 3$) [8]. Projections are computed as:

$$Q = \mathbf{h}W_Q, \quad W_Q \in \mathbb{R}^{d_{\text{model}} \times (n_h \cdot d_{\text{head}})} \tag{4}$$

$$K = \mathbf{h}W_K, \quad W_K \in \mathbb{R}^{d_{\text{model}} \times (n_{\text{kv}} \cdot d_{\text{head}})} \tag{5}$$

$$V = \mathbf{h}W_V, \quad W_V \in \mathbb{R}^{d_{\text{model}} \times (n_{\text{kv}} \cdot d_{\text{head}})} \tag{6}$$

Keys and values are repeated $n_{\text{rep}} = n_h/n_{\text{kv}} = 4$ times to match query dimension before scaled causal attention:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_{\text{head}}}} + M\right)V \tag{7}$$

3.6 RMSNorm

Root Mean Square Normalization [10] normalizes inputs by their root mean square without calculating mean statistics:

$$\text{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}} \odot \gamma \tag{8}$$

where $\gamma \in \mathbb{R}^d$ is a learnable scaling parameter initialized to ones, and $\epsilon = 1 \times 10^{-5}$.

3.7 QK Normalization

To stabilize attention logits in half-precision training (`bfloat16`), Nova applies per-head RMSNorm directly to rotated query and key vectors:

$$Q_{\text{norm}} = \text{RMSNorm}(Q), \quad K_{\text{norm}} = \text{RMSNorm}(K) \tag{9}$$

3.8 SwiGLU Feed-Forward Network

The feed-forward block utilizes a gated SwiGLU architecture [9] with SiLU activation:

$$\text{SwiGLU}(\mathbf{x}) = (\text{SiLU}(\mathbf{x}W_{\text{gate}}) \odot (\mathbf{x}W_{\text{up}}))\, W_{\text{down}} \tag{10}$$

where $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$ and $W_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}$.

3.9 Output Layer and Weight Tying

The final representation passes through RMSNorm and is projected onto vocabulary space using shared embedding weights W_{embed}:

$$\mathbf{L} = \text{RMSNorm}(\mathbf{h}_L)\, W_{\text{embed}}^{\top} \in \mathbb{R}^{B \times S \times V} \tag{11}$$

4 Design Rationale

To provide engineering clarity, Table 2 outlines the technical justification for every component selection in Nova compared to historical baseline choices.

Table 2: Engineering motivations for the architectural components adopted in Nova.

Component	**Nova Choice**	**Baseline Choice**	**Engineering Motivation**
Positional Encoding	RoPE	Absolute Learned	Encodes relative positional information while supporting context-length extrapolation.
Attention Scheme	GQA ($n_{\text{kv}} = 3$)	Standard MHA	Reduces KV-cache memory requirements and improves decoding efficiency for autoregressive inference.
Normalization	RMSNorm	LayerNorm	Provides a computationally simpler normalization scheme that is widely adopted in modern decoder-only LLMs.
Attention Stability	QK-Norm	None	Stabilizes attention logits during low-precision (BF16/FP16) training by normalizing query and key representations.
FFN Activation	SwiGLU	Standard GELU	Introduces gated feed-forward computations with improved representational capacity and is widely used in recent LLM architectures.
Embedding Matrix	Weight Tying	Untied Heads	Shares embedding and output projection weights to reduce parameter count and memory footprint.

5 Software Architecture

Nova is engineered as a clean, modular Python codebase built directly on PyTorch. Figure 3 details the repository file tree.

The test suite (`tests/test_model.py`) verifies model invariants, layer tensor dimensions, attention causal masks, and KV-cache updating prior to training.

6 Training Methodology

Nova was trained on the **TinyStories** dataset [11], consisting of synthetic stories generated by GPT-3.5/4 that maintain rich narrative structure using vocabulary accessible to young children.

Nova Modular Repository Architecture

```
nova/
|-- components/           # First-principles layer implementations
|   |-- activations.py    # SiLU, GELU, ReLU^2 activations
|   |-- attention.py      # GQA with QK-Norm + RoPE integration
|   |-- block.py          # Pre-norm transformer layer block
|   |-- feedforward.py    # Gated SwiGLU FFN block
|   |-- norms.py          # RMSNorm and LayerNorm from scratch
|   `-- rope.py           # Rotary Position Embedding cache
|-- models/
|   `-- transformer.py    # Full Causal TransformerLM model
|-- trainer/
|   |-- trainer.py        # Custom causal LM training loop with AMP
|   `-- callbacks.py      # Checkpoint, Logger, Plotter, EarlyStopping
|-- utils/
|   |-- config.py         # YAML dataclass configuration parser
|   |-- data.py           # Hugging Face dataset tokenization pipeline
|   `-- seed.py           # Reproducibility seed initializer
|-- tests/
|   |-- test_environment.py # Hardware and dependency smoke tests
|   `-- test_model.py       # Unit tests for shape and weight dynamics
|-- configs/
|   `-- default.yaml      # Model hyperparameter configuration
|-- train.py              # Unified single-command training entrypoint
`-- benchmark_nova.py     # Automated speed & memory evaluation suite
```

Figure 3: Directory organization and modular layout of the Nova codebase.

- **Dataset Split:** 100,000 training examples, 5,000 validation examples.
- **Tokenizer:** Standard GPT-2 Byte-Pair Encoding (BPE) tokenizer ($V = 50,257$).
- **Optimizer:** AdamW ($\beta_1 = 0.9, \beta_2 = 0.95, \text{weight\_decay} = 0.05$).
- **Learning Rate Schedule:** Linear warmup for 1,000 steps up to peak $\eta_{\max} = 1.0 \times 10^{-4}$, followed by cosine decay down to $\eta_{\min} = 2.0 \times 10^{-5}$.
- **Batch Size and Gradient Accumulation:** Micro-batch size $B = 8$ per GPU with 16 gradient accumulation steps (effective global batch size = 128 sequences = 65,536 tokens/step).
- **Precision:** PyTorch Automatic Mixed Precision (AMP) in `bfloat16` format.
- **Multi-GPU Parallelism:** PyTorch `DataParallel` across 2× NVIDIA Tesla T4 GPUs.

7 Experimental Setup

Table 3 outlines the standardized hardware and software environment used for training, latency timing, and memory profiling.

8 Engineering Evaluation

8.1 Parameter Analysis

Table 4 provides a layer-by-layer parameter accounting of Nova (∼141M).

8.2 FLOPs Analysis

Floating-point operations per token are computed using non-embedding parameters $N_{\text{non\_embed}} = 102.65\text{M}$:

Table 3: Hardware and software benchmarking environment.

Component	Environment Setting
Accelerator	NVIDIA Tesla T4 GPU (16 GB GDDR6 VRAM)
CUDA Driver	CUDA 12.2 / Driver version 535.104.12
Framework	PyTorch 2.2.0+cu121
Transformers Library	Hugging Face Transformers v4.38.1
Host OS	Ubuntu 22.04 LTS (Linux 6.1.85+)
Precision	BFloat16 / Float32 CUDA Execution
Timing Method	`torch.cuda.Event(enable_timing=True)`
Peak Memory Tracker	`torch.cuda.max_memory_allocated()`

Table 4: Nova (∼141M) parameter allocation breakdown.

Layer Type	Params/Layer	Total Params (12L)	% of Model
Embedding (W_{embed})	38,597,376	38,597,376	27.3%
GQA Attention (Q, K, V, O)	1,474,560	17,694,720	12.5%
SwiGLU FFN ($W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}$)	7,077,888	84,934,656	60.1%
RMSNorms	1,664	20,736	0.01%
LM Head ($W_{\text{lm\_head}}$)	Tied (0)	Tied (0)	0.0%
Total Model Parameters	–	**141,247,488**	**100.0%**
Non-Embedding Parameters	–	**102,650,112**	**72.7%**

- **Forward Pass FLOPs/token:** $2 \times N_{\text{non\_embed}} \approx$ **205.30** MFLOPs
- **Backward Pass FLOPs/token:** $4 \times N_{\text{non\_embed}} \approx$ **410.60** MFLOPs
- **Total Training FLOPs/token:** $6 \times N_{\text{non\_embed}} \approx$ **615.90** MFLOPs

8.3 Training Results

Nova completed 1,005 training steps on TinyStories. Table 5 lists final loss metrics.

Table 5: Nova final training and validation metrics.

Metric	Final Value
Total Training Steps	1,005
Final Validation Loss ($\mathcal{L}_{\text{val}}$)	**2.1764**
Validation Perplexity (PPL $= e^{\mathcal{L}}$)	**8.81**
Total Gradient Norm ($\|\mathbf{g}\|_2$)	1.00 (Clipped at 1.0)
Average Weight Update Ratio	3.81×10^{-3}
Total Parameter Norm ($\|\mathbf{W}\|_2$)	300.87

8.4 Inference Performance

Table 6 details prefill latency (TTFT) and decode latency (TPOT) across sequence lengths $S \in [32, 512]$. Nova achieved an average generation speed of **30.28 ms/token** (**33.03 tokens/s**).

8.5 Memory Analysis

Tables 7, 8, and 9 detail batch scaling, context window scaling, and VRAM component breakdown.

8.6 KV Cache Scaling Derivation

The analytical KV-cache memory requirement M_{KV} (in bytes) is derived as:

$$M_{\text{KV}} = 2 \cdot L \cdot B \cdot S \cdot n_{\text{kv}} \cdot d_{\text{head}} \cdot \text{bytes\_per\_element} \tag{12}$$

Table 6: Inference latency, decode throughput, and KV-cache speedup.

Context (S)	**Prefill (TTFT)**	**Decode w/ Cache**	**Decode w/o Cache**	**Speedup**
32 tokens	57.41 ms	53.44 ms/tok	68.00 ms/tok	**1.27**×
64 tokens	115.80 ms	49.27 ms/tok	54.86 ms/tok	**1.11**×
128 tokens	52.50 ms	64.70 ms/tok	114.11 ms/tok	**1.76**×
256 tokens	51.16 ms	29.13 ms/tok	54.31 ms/tok	**1.86**×
512 tokens	80.85 ms	30.62 ms/tok	85.85 ms/tok	**2.80**×

Table 7: Peak prefill VRAM allocation and GQA vs. MHA KV-cache footprint ($S = 512$).

Batch (B)	**Peak Prefill VRAM**	**Nova GQA Cache**	**Baseline MHA Cache**	**VRAM Saved**
1	896.09 MB	4.50 MB	18.00 MB	**13.50 MB (75.0%)**
2	955.63 MB	9.00 MB	36.00 MB	**27.00 MB (75.0%)**
4	1,073.79 MB	18.00 MB	72.00 MB	**54.00 MB (75.0%)**
8	1,308.24 MB	36.00 MB	144.00 MB	**108.00 MB (75.0%)**
16	1,786.64 MB	72.00 MB	288.00 MB	**216.00 MB (75.0%)**

Table 8: Context window memory scaling ($S = 256 \rightarrow 4096$, Batch Size = 1).

Seq. Length (S)	**Peak Prefill VRAM**	**Nova GQA Cache**	**Baseline MHA Cache**	**Cache Saved**
256 tokens	868.31 MB	2.25 MB	9.00 MB	**6.75 MB (75.0%)**
512 tokens	896.09 MB	4.50 MB	18.00 MB	**13.50 MB (75.0%)**
1,024 tokens	954.75 MB	9.00 MB	36.00 MB	**27.00 MB (75.0%)**
2,048 tokens	1,281.50 MB	18.00 MB	72.00 MB	**54.00 MB (75.0%)**
4,096 tokens	2,710.41 MB	36.00 MB	144.00 MB	**108.00 MB (75.0%)**

Table 9: Runtime VRAM allocation breakdown ($S = 512, B = 1$, BF16 Precision).

Memory Component	**Size (MB)**	**% of Total VRAM**
Model Weights (BF16)	269.41	30.1%
Activation Workspace Tensors	622.18	69.4%
KV-Cache State Tensors	4.50	0.5%
Total Peak VRAM	**896.09**	**100.0%**

For Nova ($n_{\text{kv}} = 3, d_{\text{head}} = 64$, bf16), this reduces cache size from 18.84 MB (MHA) down to 4.71 MB per sequence at $S = 512$, proving a 75.0% empirical memory reduction.

8.7 Architectural Comparison vs. GPT-2

Table 10 compares Nova against standard GPT-2.

Table 10: Architectural comparison: Nova vs. standard GPT-2 baseline.

Feature	**Nova Architecture**	**Baseline GPT-2**
Attention Mechanism	Grouped Query Attention ($n_{\text{kv}} = 3$)	Multi-Head Attention ($n_{\text{kv}} = 12$)
Position Embedding	Rotary Position Embeddings (RoPE)	Absolute Learned Embeddings
Activation Function	SwiGLU Gated Activation	Standard GELU
Normalization	RMSNorm + QK-Normalization	LayerNorm (with bias)
Weight Tying	Enabled (Shared Embed/LM Head)	Optional / Standard

8.8 Qualitative Sample Generations

Listing 1 demonstrates qualitative stories generated by Nova 141M ($\tau = 0.75, p = 0.90, k = 40$).

```
Prompt: "Once upon a time, a little girl named Lily found a shiny key in the garden."
Output: Once upon a time, a little girl named Lily found a shiny key in the garden. She
     picked it up and put it in her pocket. One day, Lily's mom told her they were
    going to the store. Lily was so excited to go to the store and get her new knife.
    When they arrived at the store, Lily saw a shiny necklace. She asked her mom if she
     could buy it for her. Her mom said yes, and they went home.

Prompt: "One day, a tiny puppy named Spot wanted to cross the big river."
Output: One day, a tiny puppy named Spot wanted to cross the big river. He ran and ran
    until he discovered a big, shiny rock. Spot was very excited to see it! Spot jumped
     over to the rock and started to climb. He was so happy that he invited all of his
    friends to see the big rock.
```

Listing 1: Qualitative story generations produced by Nova 141M.

9 Discussion

The experimental evaluation highlights key engineering tradeoffs:

1. **GQA Impact:** Grouped Query Attention reduces KV-cache footprint by 75.0%, enabling larger batch processing and higher inference throughput without degradation in narrative quality.
2. **RMSNorm Efficiency:** Removing mean-centering computations accelerates CUDA normalization kernels while maintaining stable hidden state distributions.
3. **SwiGLU Expressivity:** Gated SwiGLU activations improve representational capacity and parameter efficiency, making them a common design choice in modern decoder-only language models.

10 Limitations

We explicitly acknowledge the following study boundaries:

- **Scale Constraints:** Nova ($\sim$141M params) was trained exclusively on TinyStories and does not possess broad general-knowledge reasoning.
- **No Post-Training Alignment:** Model outputs are generated via pure autoregressive language modeling without Supervised Fine-Tuning (SFT) or RLHF/DPO preference optimization.
- **Single-Node Hardware:** Benchmarks were executed on single and dual NVIDIA T4 GPUs; large multi-node cluster evaluation was omitted.

11 Future Work

Future development will focus on integrating **FlashAttention-2** kernels, scaling to 1B+ parameters using PyTorch **FSDP**, implementing Mixture-of-Experts (MoE) routing, and supporting speculative decoding.

12 Reproducibility and Open Artifacts

All code, weights, configs, and logs are publicly available:

- **GitHub Repository:** https://github.com/sarimahsan/nova
- **Hugging Face Model:** https://huggingface.co/sarimahsan101/nova-141m-tinystories
- **Random Seed:** 42 across dataset splits and weight initialization.
- **Execution Commands:**

```
python train.py --config configs/default.yaml --output_dir results
python benchmark_nova.py
```

Listing 2: Commands to reproduce training and benchmarking.

13 Conclusion

We presented **Nova**, a fully reproducible modern decoder-only Transformer language model (∼141M parameters). By synthesizing RMSNorm, RoPE, Grouped Query Attention ($n_{\text{kv}} = 3$), QK-Normalization, SwiGLU activations, and tied embeddings, Nova achieves a 75.0% reduction in KV-cache memory overhead while delivering robust generation performance. Nova provides a practical reference architecture and benchmarking baseline for transformer research and education.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[3] H. Touvron et al., "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] A. Q. Jiang et al., "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[5] A. Yang et al., "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[6] Gemma Team, Google, "Gemma: Open models based on Gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[7] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "RoFormer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024.

[8] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[9] N. Shazeer, "GLU variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[10] B. Zhang and R. Sennrich, "Root mean square layer normalization," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[11] R. Eldan and Y. Li, "TinyStories: How small can language models be and still speak coherent English?" *arXiv preprint arXiv:2305.07704*, 2023.

A Configuration YAML File

The complete default configuration file (`configs/default.yaml`):

```yaml
hidden_dim: 768
num_layers: 12
num_heads: 12
num_kv_heads: 3
ffn_dim: 3072
max_seq_len: 512
vocab_size: 50257
tie_word_embeddings: true
norm_type: rmsnorm
activation_type: swiglu
qk_norm: true
use_rope: true
dropout: 0.05
lr: 1.0e-4
min_lr: 2.0e-5
weight_decay: 0.05
warmup_steps: 1000
batch_size: 8
gradient_accumulation_steps: 16
epochs: 3
clip_grad: 1.0
precision: bf16
dataset_name: roneneldan/TinyStories
output_dir: results
```

Listing 3: Default Nova configuration file.

B Mathematical Derivation of KV-Cache Memory Footprint

For a decoder model with L layers, sequence length S, batch size B, n_{kv} key/value heads, and head dimension d_{head}, key and value matrices stored in half-precision (`bfloat16`, 2 bytes/element) require:

$$M_{\text{KV}} = 2 \cdot (L \cdot B \cdot S \cdot n_{\text{kv}} \cdot d_{\text{head}} \cdot 2 \text{ bytes}) \tag{13}$$

Plugging in Nova specs ($L = 12, d_{\text{head}} = 64, n_{\text{kv}} = 3$):

$$M_{\text{KV,Nova}} = 2 \cdot (12 \cdot 1 \cdot 512 \cdot 3 \cdot 64 \cdot 2) = 4{,}718{,}592 \text{ bytes} \approx \mathbf{4.50} \text{ MB} \tag{14}$$

Under standard Multi-Head Attention ($n_{\text{kv}} = 12$):

$$M_{\text{KV,MHA}} = 2 \cdot (12 \cdot 1 \cdot 512 \cdot 12 \cdot 64 \cdot 2) = 18{,}874{,}368 \text{ bytes} \approx \mathbf{18.00} \text{ MB} \tag{15}$$

Yielding a saving of:

$$\Delta M_{\text{KV}} = \frac{18.00 - 4.50}{18.00} = \mathbf{75.0}\% \tag{16}$$

C Mathematical Derivation of Transformer FLOPs

For non-embedding parameters $N_{\text{non\_embed}}$, each matrix multiplication layer with weight $W \in \mathbb{R}^{d_1 \times d_2}$ performs $2 \cdot d_1 \cdot d_2$ floating point operations (multiply-add) per token.

Summing across all GQA projection matrices (W_Q, W_K, W_V, W_O) and SwiGLU matrices ($W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}$) across $L = 12$ layers gives:

$$\text{FLOPs}_{\text{fwd}} = 2 \times N_{\text{non\_embed}} = 2 \times 102.65\text{M} = \mathbf{205.30} \text{ MFLOPs/token} \tag{17}$$

Since backward pass gradients require 2× the operations of forward computation ($2 \times \text{FLOPs}_{\text{fwd}}$), total training compute per token is:

$$\text{FLOPs}_{\text{train}} = 6 \times N_{\text{non\_embed}} = 6 \times 102.65\text{M} = \mathbf{615.90}\ \text{MFLOPs/token} \tag{18}$$