BwETAFv3: Overview

1. Introduction

BwETAFv3 represents the latest iteration in the BwETAF model series, building upon its predecessors to achieve improved stability, scalability, and training efficiency. Designed as a causal language model (CLM) for text generation, it leverages incremental refinements in architecture choices, normalization strategies, and dataset utilization. BwETAFv3 demonstrates how iterative evolution across model generations can produce more robust and adaptable language models.

In addition to its strong performance across a wide range of downstream tasks, the instruct-tuned variant of BwETAFv3 enhances alignment, responsiveness, and the ability to follow user intents, making it more suitable for interactive applications. Furthermore, the underlying module incorporates KV caching, enabling more efficient inference and support for longer contexts with reduced computational overhead.

The motivation behind the BwETAF series is to explore efficient language model design at small-to-medium scales, where stability and training practices play a critical role in performance. Unlike large industrial-scale LLMs, this work emphasizes practical experimentation, focusing on how design choices such as smarter tokenizers, efficient optimizers, and dataset composition can shape model behavior.

2. Architecture Overview

At its foundation, BwETAFv3 follows the **standard Transformer design**, illustrated in Figure 1. Each Transformer block consists of two key components: a self-attention layer and a feed-forward network (FFN). The self-attention mechanism allows the model to compute context-aware representations by weighting relationships between tokens across the input sequence. This is followed by the FFN, a position-wise network that applies non-linear transformations to expand the representational capacity of the model.

To ensure stability during training, each sub-layer is wrapped with residual connections and preceded by normalization (a pre-normalization scheme). These design elements are crucial for scaling Transformer models effectively, preventing issues such as vanishing gradients and enabling deeper architectures.

While BwETAFv3 retains this backbone, it integrates a set of targeted refinements to improve efficiency and stability at the small-to-medium model scale. These include grouped-query attention (GQA) for reduced memory and compute cost, SwishGLU activations in the FFN for improved expressivity, rotary positional embeddings (RoPE) for longer context generalization, and RMS normalization in place of LayerNorm for more stable training.

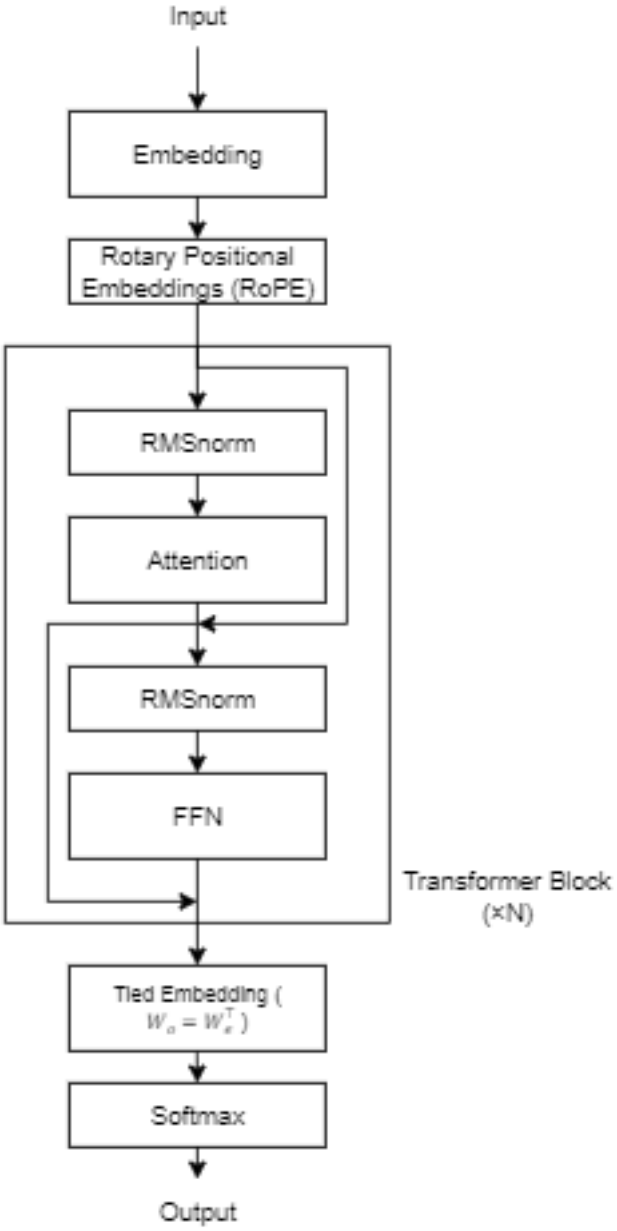


Figure 1: High-level schematic of BwETAFv3 architecture

3. Training Dynamics

3.1 Optimizer Selection

BwETAFv3 is trained using the **AdamW optimizer**, which extends the standard Adam algorithm by decoupling weight decay from the gradient update. This approach provides effective regularization, preventing overfitting while maintaining stable convergence. AdamW is particularly well-suited for Transformer-based architectures, handling sparse gradients efficiently and adapting learning rates for individual parameters, which stabilizes training across the network's deep layers.

Among several optimizers tested — including Lion and Adafactor — AdamW was preferred due to its remarkable stability and consistent performance gains, even though it comes with heavier resource usage. The choice of optimizer directly impacts the convergence behavior and overall quality of learned representations, and in BwETAFv3, AdamW combined with careful learning rate scheduling and gradient clipping ensures robust and reliable training.

3.2 Learning Rate Schedule

BwETAFv3 employs a **cosine decay learning rate schedule with a warmup period of approximately 250–300 steps**. This approach allows the initial learning rate to remain relatively high while the model transitions from easier datasets to more challenging ones, facilitating faster convergence and more effective adaptation. Compared to linear decay, which maintains a lower learning rate throughout much of training, cosine decay ensures that the model can fully adjust to the complexity of the actual dataset, improving overall performance and stability. This schedule, combined with AdamW and gradient clipping, contributes significantly to the smooth and reliable training dynamics observed in BwETAFv3.

3.3 Gradient Clipping & Regularization

To maintain stable training in BwETAFv3, **gradient norm clipping** was applied with a threshold of 7, preventing gradient explosions while allowing meaningful updates. An alternative strategy — absolute mean gradient-wise clipping — was considered but not utilized in this training run.

Regularization was further supported through optimizer hyperparameters, with AdamW settings of `b1 = 0.95`, `b2 = 0.98`, and `eps = 1.5e-8`, helping stabilize updates and control variance in parameter adjustments. Additionally, a modest **dropout rate of 0.05** was applied initially, balancing model robustness against overfitting while preserving most of the representational capacity of the network.

3.4 Precision & Stability

BwETAFv3 was trained with **bfloat16 (bf16) precision** for both model weights and optimizer state, allowing for efficient use of TPU memory and

faster computation without significant loss of numerical fidelity. Despite this, the **loss function was computed in full precision (fp32)** to maintain stability during backpropagation, preventing issues such as underflow or vanishing gradients that can occur in lower-precision arithmetic.

This setup provides flexibility: while training benefits from bf16 efficiency, the model can later be loaded in any precision (bf16, fp32, or mixed) depending on the target hardware and inference requirements. Combining bf16 storage with fp32 loss calculation ensures both computational efficiency and stable, high-quality training dynamics.

3.5 Batching

BwETAFv3 used **variable batch sizes** optimized per model to fully utilize TPU memory while ensuring stable convergence. Per-device batch size was chosen based on memory constraints, while the global effective batch size was computed as: global batch size = per-device batch $\times$ number of devices **Gradient accumulation was not used** in these experiments. Specific batch sizes for each model are documented in their respective model cards. This approach balanced hardware efficiency and training stability, enabling large-context sequence modeling without running into out-of-memory issues.

3.6 Dataset Scale

BwETAFv3 was trained using a staged dataset strategy to balance **early convergence** with **final model robustness**. Initially, the model was trained on **TinyStories**, a smaller and simpler corpus, which allowed it to quickly acquire basic syntax, semantic patterns, and reasoning abilities. This stage was especially beneficial for smaller models in the family, enabling efficient learning without overloading limited capacity.

Subsequently, the model was trained on a larger, more diverse dataset, **FineWeb**, sampled to provide a wide range of linguistic and conversational patterns. The dataset size for each model was chosen proportionally to its parameter count, ensuring effective utilization of capacity while avoiding overtraining.

Training was guided by a **tokens-per-parameter ratio of approximately 20–30**, adapted to each model size. Smaller models received fewer tokens for efficient learning, while larger models were exposed to more tokens to fully leverage their parameters and improve generalization.

This staged approach allowed the family of models to maintain stable training, achieve strong convergence on simpler data, and progressively acquire the capacity to handle more complex, varied inputs.

3.7 Context Length

All models in the BwETAFv3 family were trained with a **base context length of 2048 tokens**, providing a balance between computational efficiency and the ability to capture long-range dependencies in text. This context length was chosen to accommodate typical conversational and narrative inputs while keeping memory usage manageable on a single TPU.

To support relative positional information, **Rotary Positional Embeddings (RoPE)** were employed. For this family of models, the RoPE **frequency was set to 10000.0**, which defines the maximum effective sequence length the model can handle. Exceeding this frequency leads to breakdowns in attention computation and can cause the model to fail on longer sequences, effectively setting a hard upper limit on context length.

However, the effective context length can be **extrapolated beyond 2048 tokens** with targeted fine-tuning on smaller datasets. This allows the model to handle longer sequences without retraining from scratch, providing flexibility for tasks requiring extended context while maintaining efficiency and stability.

3.8 Stability Techniques

Several techniques were employed to ensure **stable and efficient training** of the BwETAFv3 family:

- **Initialization:** All weights in the model were initialized with a consistent value across layers, chosen experimentally to ensure that the initial loss roughly corresponds to `ln(vocab_size)`. Biases were initialized to **0**. This careful initialization helped prevent early-stage gradient explosions or vanishing and stabilized training from the first step.
- **Normalization:** The model uses **RMSNorm**, which has been found to be both faster and more stable than traditional LayerNorm. RMSNorm normalizes the hidden states based on their root-mean-square magnitude, ensuring consistent scaling across layers. Additionally, **pre-layer normalization** was applied, placing normalization before the attention and feed-forward blocks, further improving gradient flow in deep stacks.
- **Learning Rate Scaling:** The learning rate was carefully tuned relative to model size and batch size to prevent instability, especially during early training steps.
- **Gradient Smoothing:** Techniques such as **gradient clipping** were applied to cap extreme gradient values, reducing the risk of spikes that can destabilize training. This was combined with RMSNorm to maintain smooth updates across all layers.

Together, these stability techniques ensured that models across the family—regardless of size—trained reliably and efficiently, achieving consistent convergence even with long-context sequences and varying dataset scales.

4. Architectural Choices

4.1 GQA (Grouped Query Attention)

Grouped Query Attention (GQA) was employed to improve memory efficiency and training speed without significantly compromising model performance. In standard multi-head attention, each query independently attends to all key-value pairs, which can lead to large memory usage in the KV cache, especially for long sequences and larger models.

GQA addresses this by grouping multiple queries together allowing them to share key-value projections. This reduces the size of the KV cache and decreases the number of computations required, resulting in **faster training and lower memory consumption**.

Importantly, this optimization introduces **minimal performance loss**, meaning the model retains almost the same attention quality while benefiting from much greater hardware efficiency. This makes GQA particularly valuable when training on TPUs with memory constraints, as it allows the model to scale in size and context length without hitting hardware limits.

4.2 SwishGLU / Activation Functions

BwETAFv3 uses **SwishGLU** as the primary activation function in its feed-forward blocks, chosen for **efficiency and improved gradient flow** compared to traditional activations like GeLU. SwishGLU combines the benefits of the Swish function with a gating mechanism, allowing the network to model more complex interactions without significantly increasing computation.

The **SwishGLU activation** can be expressed as:

```
FF(x) = (x * sigmoid(x)) x W2(x)
```

Where:
- `x` is the input to the activation,
- `sigmoid(x)` provides the smooth Swish gating,
- `x` represents element-wise multiplication,
- `W_2(x)` is the linear projection in the feed-forward block.

This formulation allows the network to retain stronger gradients during back-propagation and improves convergence speed. It is also computationally efficient on TPU architectures, which is important for models of this family, as it reduces per-step training time while maintaining expressiveness.

4.3 RoPE Positional Embeddings

BwETAFv3 uses **Rotary Positional Embeddings (RoPE)** to encode token positions in a way that naturally supports **longer context extrapolation**. Unlike absolute positional embeddings, RoPE applies a **rotation to the query**

and key vectors based on their position, allowing the model to generalize to sequence lengths beyond the training context.

For a vector of dimension `d`, RoPE rotates each pair of dimensions `(i, i+1)` as follows:

```
q_rotated[i]   = q[i] * cos(0) - q[i+1] * sin(0)
q_rotated[i+1] = q[i] * sin(0) + q[i+1] * cos(0)
```

Where `0 = position / (10000^(2i/d))`.

- For this family of models, the **RoPE frequency was set to 10000.0**, which effectively determines the maximum sequence length the model can handle. Exceeding this frequency can cause attention computations to break.

- This approach allows the model to handle the **base context of 2048 tokens** while also providing flexibility for extrapolation with targeted fine-tuning.

RoPE was chosen for its scaling properties stability, and the ability to extend context length without additional parameters or retraining.

4.4 Tokenizer

BwETAFv3 uses a **custom Byte-Pair Encoding (BPE) tokenizer** with a **vocabulary size of 16,384**, trained specifically on the same dataset used for the smaller 33M model in the family. This ensures that tokenization is closely aligned with the data distribution the model sees during training.

Key characteristics of this tokenizer include:
- **Byte-level BPE:** Operates at the byte level rather than being whitespace-based, allowing it to handle arbitrary text, including code snippets or unusual symbols.
- **Accent removal:** Preprocessing removes diacritics to simplify the token space and improve model generalization across similar words.
- **Emoji support:** While basic emojis are supported, coverage is limited due to the vocabulary size and byte-level approach. Full emoji coverage is planned for future models, such as an instruct-tuned variant.
- **Training framework:** The tokenizer was built using **Hugging Face's Trainer**, ensuring reproducibility and compatibility with the rest of the training pipeline.

This custom tokenizer balances efficiency, data alignment, and vocabulary coverage providing a robust foundation for the family of models while leaving room for expansion in future iterations.

4.5 Tied Embedding

To improve parameter efficiency and maintain consistent representations between input and output layers, BwETAFv3 employs **tied embeddings**. In this approach, the same embedding matrix is shared between the token embedding layer (used to map input tokens to continuous vectors) and the final output projection layer (used to map hidden states back to vocabulary logits).

This technique reduces the total number of trainable parameters, which is particularly beneficial at small-to-medium model scales, where parameter count and memory usage are critical constraints. Beyond efficiency, tied embeddings also provide representational consistency: the same vector space used to encode tokens is reused to decode them, reinforcing alignment between how the model "understands" and how it "expresses" tokens.

In BwETAFv3, this design choice balances performance with compactness, ensuring that parameters are allocated to core components such as attention and feed-forward layers rather than duplicated embeddings.

5. Training Setup & Infrastructure

5.1 Hardware

BwETAFv3 models were trained on two main TPU setups:
- **TPU v2-8** via Google Colab
- **TPU v3-8** via Kaggle

Each setup utilized **8 TPU cores**, with memory limits and compute throughput optimized per model size. Specific hardware configurations, including memory per core and per-device batch size for each model, are detailed in the respective **model cards**. These setups provided sufficient parallelism for training the family of models while keeping training times manageable.

5.2 Software Stack

The training pipeline was built primarily with:
- **JAX / Flax** for model definition, forward/backward passes, and autograd operations
- **Optax** for the AdamW optimizer and learning rate scheduling
- **Hugging Face Tokenizers** for building and training the custom BPE tokenizer

This stack was chosen for its **flexibility, performance on TPUs, and seamless integration** with existing Hugging Face tools.

5.3 Distributed Training Strategies

Training was performed using **data parallelism across all 8 TPU cores**. Inputs were sharded evenly across devices, and gradients were synchronized

after each step to maintain a consistent global model update. Techniques such as `pmap` were used for parallel execution, leveraging TPU cores efficiently. This approach provided scalable throughput without requiring model sharding or more complex MoE-style routing.

Overall, this combination of hardware and software allowed the models to be trained efficiently, with stable convergence and reproducible results across both TPU setups.

6. Inference Considerations

6.1 KV Caching

BwETAFv3 integrates **KV caching** to speed up autoregressive generation by storing key-value pairs for past tokens. Currently, KV caching is supported only for **single-token streaming**, and **batch processing is not yet implemented**. Usage instructions are provided in the corresponding **model card**.

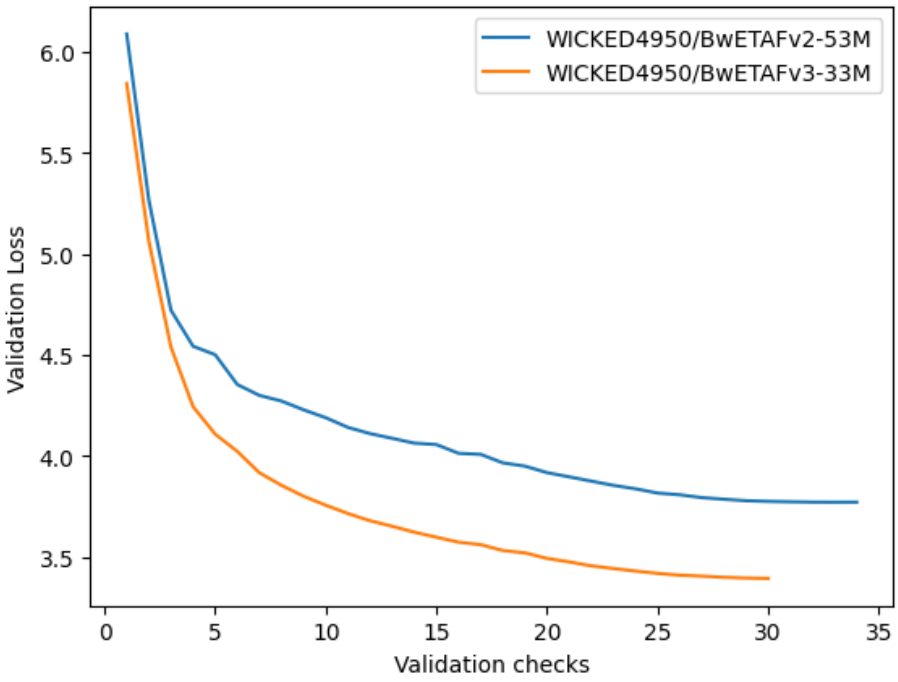
6.2 Context Window Extrapolation

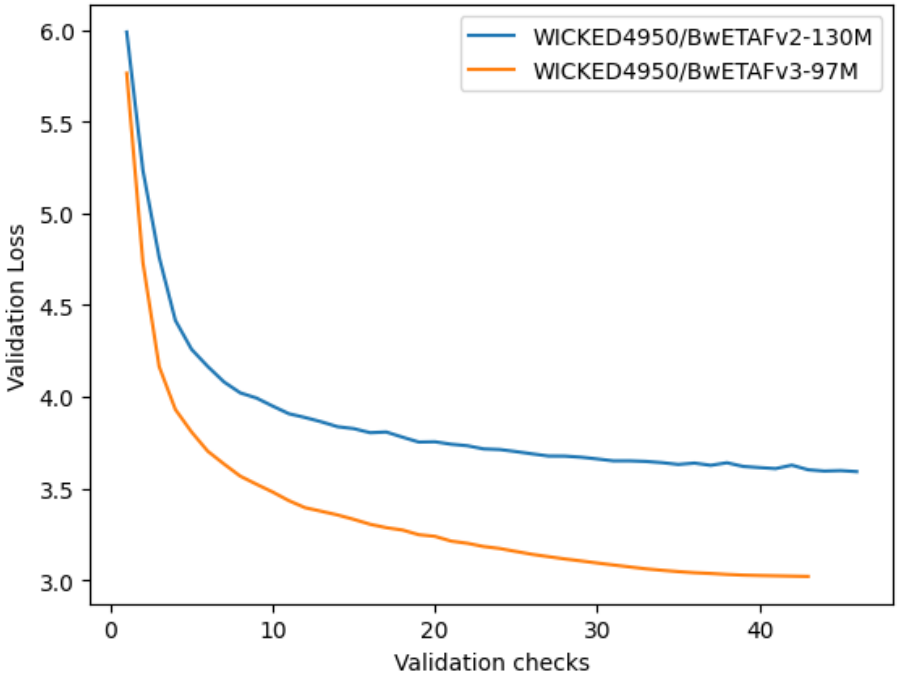
Although the models were trained with a **base context length of 2048 tokens**, they can handle slightly longer sequences for short extensions. Beyond this, performance gradually degrades, especially in attention accuracy and long-range dependencies. Users can extend the context modestly for generation tasks, but significant increases beyond 2048 tokens may lead to unstable outputs.

7. Evaluation

BwETAFv3 is evaluated primarily in comparison to its predecessor, BwETAFv2, and other small-scale transformer models. While exact benchmarks are available in the **model cards for each model**, several trends are notable:

- **Perplexity:** BwETAFv3 shows modest improvements over v2 on both TinyStories and FineWeb datasets, indicating better predictive capacity.
- **Efficiency:** Architectural refinements and RMSNorm usage reduce training time and memory footprint compared to v2.
- **Qualitative behavior:** Generated outputs exhibit improved coherence and stability, reflecting the design focus on "doing more with less."





This evaluation emphasizes architectural and training advancements rather than

exhaustive model-specific metrics, highlighting how v3 builds on and improves v2's foundation.

8. Related Work

BwETAFv3 draws inspiration from several small-scale transformer efforts that prioritize architectural elegance and training efficiency:

- **GPT-2**: Inspired our hyperparameter choices and overall training setup.
- **OPT**: Served as a benchmark for small-scale model performance.
- **TinyLlama**: Provided the RoPE positional embeddings we adopted in BwETAFv3.

While BwETAFv3 remains slightly behind these models in raw performance, it shares their ethos: achieving more with less, with style.

8. Limitations

While the BwETAFv3 family demonstrates strong training and architectural design, several limitations should be noted:

- **Training Instability:** Despite stability techniques such as RMSNorm, pre-layer normalization, and careful initialization, gradient spikes and sensitivity to learning rate schedules can still occur, especially during early training steps or with larger models.
- **Dataset Biases and Content Limitations:** Models were primarily trained on TinyStories and sampled portions of FineWeb. This can lead to biases in language style, content, and reasoning patterns. Additionally, FineWeb contains unfiltered internet text, which may include uncensored or inappropriate content.
- **Context Limitations:** Although trained with a base context of 2048 tokens, performance gradually degrades for longer sequences. Short extensions beyond this limit are possible, but very long inputs may reduce attention quality and coherence.
- **Hallucinations:** Like most LLMs, the models can produce plausible but incorrect or nonsensical outputs, particularly on topics not well represented in the training data.
- **Compute Constraints:** Training was performed on a single TPU v2-8 or v3-8 setup. This limits the total number of tokens processed per model and restricts experimentation with larger batch sizes or longer context windows.
- **Intended Use:** While the models can be fine-tuned for downstream tasks such as instruct tuning or chatbot applications, this paper focuses on the research and architectural design of the family. Users should consider task-specific evaluation before deployment.

9. Conclusion

This paper presents the BwETAFv3 family, developed by **Boring._.wicked**, paper written with the help of **ChatGPT**.

The models demonstrate strong performance compared to their predecessors, achieving stable convergence and generalization across multiple datasets. They also show **competence with other models in a similar parameter range**, thanks to design choices such as Grouped Query Attention, SwishGLU activations, RMSNorm with pre-layer normalization, and scalable RoPE positional embeddings.

Overall, the BwETAFv3 family highlights how careful design and optimization can produce **efficient and versatile language models**, capable of downstream adaptation such as instruct tuning or chatbot usage.

10. Contact

For further questions, collaborations, or updates related to the BwETAFv3 family, you can reach me through the following platforms:

- **GitHub:**WICKED4950
- **Hugging Face:** WICKED4950
- **Instagram:** Boring._.wicked
- **Discord:** fused_computation.1

11. Acknowledgements

Special thanks to the open-source ecosystem that made this work possible:
- **JAX** and **Flax** for enabling flexible and efficient training
- **Hugging Face** for tokenizer tools
- **Google Colab** and **Kaggle** for TPU access
- This paper specifically is written with the help of **ChatGPT**
- Inspiration drawn from the broader research community behind models like **DeepSeek**, **LLaMA** and **ChatGPT**

12. Citations

- Attention Is All You Need — https://arxiv.org/abs/1706.03762
- AdamW — https://arxiv.org/abs/1711.05101
- Lion Optimizer — https://arxiv.org/abs/2302.06675
- TinyStories Dataset — https://arxiv.org/abs/2305.07759
- FineWeb Dataset — https://huggingface.co/datasets/HuggingFaceFW/fineweb
- RMSNorm — https://arxiv.org/abs/1910.07467
- Grouped Query Attention (GQA) — https://arxiv.org/abs/2305.13245
- SwiGLU — https://arxiv.org/pdf/2002.05202

- Rotary Position Embeddings (RoPE) — https://arxiv.org/abs/2104.09864
- TPU Architecture — https://arxiv.org/abs/1704.04760