Table 1: The details of model architecture

Hidden size	Intermediate Hidden Size	Context Len	Heads	Layers	Vocab size
2,048	5,632	2,048	16	22	32,000

Positional embedding We use RoPE (Rotary Positional Embedding) (Su et al., 2021) to inject positional information into our model. RoPE is a widely adopted method recently used by many mainstream large language models, such as PaLM (Anil et al., 2023), Llama (Touvron et al., 2023a), and Qwen (Bai et al., 2023).

RMSNorm In pre-normalization, to attain a more stable training, we normalize the input before each transformer sub-layer. In addition, we apply RMSNorm (Zhang and Sennrich, 2019) as our normalization technique, which can improve training efficiency.

SwiGLU Instead of using the traditional ReLU non-linearity, we follow Llama 2 and combine Swish and Gated Linear Unit together, which is referred to as SwiGLU (Shazeer, 2020), as our activation function in TinyLlama.

Grouped-query Attention To reduce memory bandwidth overhead and speed up inference, we use grouped-query attention (Ainslie et al., 2023) in our model. We have 32 heads for query attention and use 4 groups of key-value heads. With this technique, the model can share key and value representations across multiple heads without sacrificing much performance.

2.3 Speed Optimizations

Fully Sharded Data Parallel (FSDP) During training, our codebase has integrated FSDP[1] to leverage multi-GPU and multi-node setups efficiently. This integration is crucial in scaling the training process across multiple computing nodes, which significantly improves the training speed and efficiency.

Flash Attention Another critical improvement is the integration of Flash Attention 2 (Dao, 2023), an optimized attention mechanism. The repository also provides fused layernorm, fused cross entropy loss, and fused rotary positional embedding, which together play a pivotal role in boosting computational throughput.

xFormers We have replaced the fused SwiGLU module from the xFormers (Lefaudeux et al., 2022) repository with the original SwiGLU module, further enhancing the efficiency of our codebase. With these features, we can reduce the memory footprint, enabling the 1.1B model to fit within 40GB of GPU RAM.

Performance Analysis and Comparison with Other Models The incorporation of these elements has propelled our training throughput to 24,000 tokens per second per A100-40G GPU. When compared with other models like Pythia-1.0B (Biderman et al., 2023) and MPT-1.3B [2], our codebase demonstrates superior training speed. For instance, the TinyLlama-1.1B model requires only 3,456 A100 GPU hours for 300B tokens, in contrast to Pythia's 4,830 and MPT's 7,920 hours. This shows the effectiveness of our optimizations and the potential for substantial time and resource savings in large-scale model training.

2.4 Training

We build our framework based on lit-gpt.[3] In adhering to Llama 2 (Touvron et al., 2023b), we employ an autoregressive language modeling objective during the pretraining phase. Consistent with Llama 2's settings, we utilize the AdamW optimizer (Loshchilov and Hutter, 2019), setting β_1 at 0.9 and

[1] https://huggingface.co/docs/accelerate/usage_guides/fsdp
[2] https://huggingface.co/mosaicml/mpt-1b-redpajama-200b
[3] https://github.com/Lightning-AI/lit-gpt