



AXLearn

Modular, Hardware-Agnostic Large Model Training

Xiyou Zhou

Apple Foundation Model

MLSys 2026 github.com/apple/axlearn

■ The Problem

At Apple, we train AI models for products serving **billions of users** — on a fleet that spans **GPU, TPU, and AWS Trainium**, with **hundreds** of architecture variants.

Existing training systems force a costly trade-off:

- No system covers **all three** of GPU, TPU, and Trainium (Megatron-LM → GPU only; MaxText → GPU + TPU, no Trainium).
- Most rely on **subtyping**, which forces cascading edits whenever a feature lands.
- No system delivers both first-class **modularity** *and* **hardware-agnostic** training.

We need both, at once.

Minimal-code architecture changes *and* the same model running across every accelerator we own.

■ Related Work: No System Has All Three

Existing systems each give up something — backend coverage, architectural agility, or both.

System	Model Agnostic	Modular	3D Parallelism	GPU	TPU	Trainium
Megatron-LM	–	–	✓	✓	–	–
DeepSpeed	✓	–	✓	✓	–	–
PyTorch FSDP	✓	–	–	✓	–	–
PyTorch XLA FSDP	✓	–	–	✓	✓	–
TorchTitan	–	partial	✓	✓	–	–
Flax / Haiku / Pax	✓	partial	✓	✓	✓	–
MaxText	–	partial	✓	✓	✓	–
AXLearn (Ours)	✓	✓	✓	✓	✓	✓

The gap we close

First system that is **model-agnostic**, **modular**, and runs across **GPU**, **TPU**, and **Trainium**.

■ Why Subtyping Hurts

Replacing an FFN with MoE *looks* like 4 lines (DeepSpeed):

```
- self.fc3 = nn.Linear(84, 10)
+ self.fc3 = nn.Linear(84, 84)
+ self.fc3 = deepspeed.moe.layer.MoE(...)
+ self.fc4 = nn.Linear(84, 10)
```

Every **parent** layer instantiates the child — so a child change forces a new parent subtype. By induction, the change propagates to *every ancestor module*.

4 LoC

the diff in isolation

200+ LoC

actual QwenV2→MoE diff

4,000 LoC

at Apple's variant scale

■ LoC-Complexity: the Right Metric

Plain LoC is a snapshot. We need to measure how a system evolves.

LoC-Complexity

The *asymptotic* LoC required to add a new feature variant, as a function of N and M .

N = modules in the system (e.g., attention, FFN, transformer layer);

M = variants of the new feature (e.g., MoE routing / gating schemes).

System	LoC-Comp(RoPE)	LoC-Comp(MoE)	RoPE LoC	MoE LoC
Megatron-LM	$O(NM)$	$O(N)$	400	20
DeepSpeed	$O(NM)$	$O(NM)$	320	4,000
TorchTitan	$O(NM)$	$O(NM)$	240	400
Praxis (Pax)	$O(NM)$	$O(M)$	300	5
MaxText	$O(NM)$	$O(NM)$	200	300
AXLearn (Ours)	$O(1)$	$O(1)$	0	0

■ Key Idea: Strict Encapsulation

- **Every** module is replaceable: input pipeline, model, optimizer, checkpointer, trainer loop.
- **Configs compose hierarchically as a tree.** A parent never inspects its child's hyperparameters — it only agrees on input/output shapes.
- A **config modifier** traverses the tree and rewrites targeted subtrees *without touching parents*.

Drop-in replacement, by construction.

Swap an FFN for an MoE; switch the attention kernel; change the optimizer — the surrounding architecture is untouched.

■ Config Modifier: 10 LoC, 1,000+ Experiments

```
def replace_config(cfg, tgt, new_cfg):          # the modifier (10 LoC)
    def enter_fn(child):
        for k, v in child.items():
            if isinstance(v, tgt.Config):
                new_cfg.set(**v.items())
                child.set(k, new_cfg)
    cfg.visit(enter_fn=enter_fn)
```

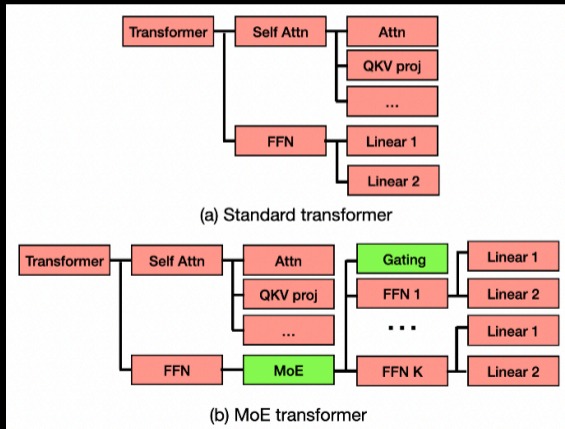
Apply MoE to every experiment config in production:

```
replace_config(trainer_cfg, target=FeedForwardLayer,
               new_cfg=MoELayer.default_config().set(...))
```

Reused 1,000+ times in production

No edits to TransformerLayer, Llama, or any other ancestor module.

■ Modularity in One Picture

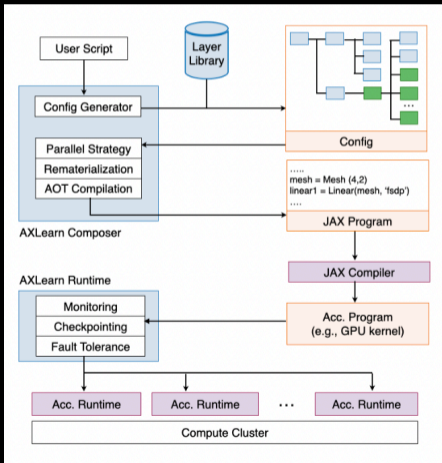


The **green** nodes are the *only* changes from the standard Transformer to the MoE Transformer.

DeepSpeed: 4,000+ LoC in existing files.
AXLearn: **0** — only the new module is added.

Counts only changes to existing files; the MoE implementation itself is comparable across frameworks.

■ System Architecture



Composer (*config* → *JAX program*)

- Mesh selection & sharding annotations
- Rematerialization, AOT compilation
- Per-backend kernel selection

Runtime (*orchestration on the cluster*)

- Async checkpointing (S3 / GCS)
- Watchdog & fault tolerance
- Slice-level hot-swap on Kubernetes

Built on **JAX + XLA + GSPMD** — one program, every backend.

■ Hardware-Agnostic via XLA + GSPMD

One JAX program, four backends. But XLA alone isn't enough.

- **Native config-based parallelism** — FSDP, pipeline, expert, sequence, tensor parallelism, no model code edits.
- **Mesh rules** apply per-backend modifiers automatically: FSDP within TPU v5e slices + DP across; 8-way TP within H100 nodes + FSDP across.
- **Rematerialization tags** let users save/recompute specific activations per hardware (HBM-bound vs compute-bound).
- **Custom kernels** (FlashAttention, etc.) drop in as black-box nodes — without breaking encapsulation.
- **First training system** to support AWS Trainium2 *at scale*.

Same Llama2-70B config trains on H100, TPU-v5p, *and* Trainium2 — only the mesh shape changes.

■ Performance: Matches or Exceeds SOTA on Every Backend

Identical hardware, batch size 1024. **The only system on all four backends.**

Model	Hardware	System	Iter (s)	MFU	Tok/s
Llama2-7B	32 × H100-8	MaxText	1.4	54.7%	3.0M
	32 × H100-8	AXLearn	1.4	54.2%	3.0M
	tpu-v5p-512	MaxText	2.7	61.6%	1.6M
	tpu-v5p-512	AXLearn	2.5	66.2%	1.7M
	64 × Trainium2-16 [†]	AXLearn	1.2	24.2%	3.5M
Llama2-70B	64 × H100-8	MaxText	9.4	39.1%	446K
	64 × H100-8	AXLearn	9.2	40.0%	456K
	tpu-v5p-1024	MaxText	12.3	64.4%	341K
	tpu-v5p-1024	AXLearn	11.6	68.0%	360K
	64 × Trainium2-16 [†]	AXLearn	11.2	25.0%	374K
Qwen-3 30B-A3B	tpu-v5p-1024	MaxText	13.0	31.3%	1.3M
	tpu-v5p-1024	AXLearn	12.9	31.6%	1.3M
	64 × B200-8	Megatron-LM	4.1	20.2%	4.1M
	64 × B200-8	AXLearn	4.3	19.2%	3.9M

[†] No baseline system supports Trainium2 at this scale — AXLearn is the only entrant.

Near-linear scaling on a 150B model: MFU 40.6% at 8,192 chips → 37.6% at 32,768 chips.

■ Runtime: Built for Failures

At 32,768 chips, hardware breaks every few hours — recovery is everything.

- **Async checkpointing** to S3 / GCS with bandwidth- and memory-aware sharding.
- **Watchdog** on step time + utilization triggers restart, on-call alert, or stack-trace dump.
- **Multi-tier checkpoints** on node-local memory/disk; weights restored in-replica then broadcast over fast interconnect.
- **Slice-level hot-swap** on Kubernetes: over-provisioned spares replace failed nodes in seconds.

4 min

slice hot-swap

9 min

checkpoint restore

21 min

total time lost*

* downtime + lost progress since last checkpoint, on a 32,768-chip job

■ Bonus: Modular Design Enables Efficient Inference

Encapsulated KV-cache layouts let us drop in continuous batching, disaggregated prefill/decode, and paged caches *without* re-implementing models or layers.

Llama2 inference on TPU (v5p-8 / v6e-8). vLLM's TPU support was *experimental* at the time of testing — this is a feasibility comparison, not a head-to-head against vLLM's production GPU stack.

Model	System	TTFT	TPOT	Throughput
Llama2-7B (TPU v5p-8)	vLLM	538.6 ms	22.4 ms	1,117 tok/s
	AXLearn	40.1 ms	9.1 ms	3,125 tok/s
Llama2-70B (TPU v6e-8)	vLLM	80 s	189.8 ms	705 tok/s
	AXLearn	150.5 ms	28.1 ms	1,139 tok/s

The takeaway

A modular training framework can deliver a production-grade inference stack *with minimal additional effort* — not a claim that AXLearn is a specialized inference engine.

■ Deployed at Apple

- One codebase across **AWS, GCP, Azure, on-premises** — GPU, TPU, Trainium2.
- **Thousands of models** trained by **hundreds of engineers** over several years.
- Trains models powering **Apple Intelligence** — features used by **1B+ users**.
- Open source under Apache 2.0; built on **JAX + XLA + GSPMD**.

Open source — github.com/apple/axlearn

Apache 2.0 · JAX · XLA · GSPMD

■ Takeaways

1. **Strict encapsulation** unlocks modularity — configs as trees, modules as drop-ins.
2. **LoC-Complexity** captures what plain LoC misses: how a system *evolves*.
3. **One codebase, every backend** — GPU, TPU, B200, Trainium2 — via XLA + custom kernels + mesh rules.
4. **Bet on a compiler-first design** (XLA + GSPMD) paid off: layers stay modular as new hardware arrives.

Open source — github.com/apple/axlearn

Apache 2.0 · JAX · XLA · GSPMD

