Lunaris MoC: Mixture-of-Collaboration with Iterative Reasoning Loops

Francisco Antonio

Abstract

We present **Lunaris MoC**, a decoder-only Transformer that replaces standard feed-forward sublayers with *Mixture-of-Collaboration* (MoC) blocks and augments expert computation with *Iterative Reasoning Loops* (IRL). For each token, a learned router selects a small Top-K set of experts. Unlike classic mixture-of-experts (MoE) designs that process experts independently, MoC explicitly enables communication among the selected expert states through a lightweight message-passing module (multi-head attention + MLP) before a learned fusion produces the block output. IRL increases effective depth inside each expert by performing multiple refinement micro-steps without increasing parameter count. We describe the architecture, routing/capacity control, auxiliary router losses, and an efficient training implementation with distributed data-parallelism, bf16 autocast, fused AdamW, checkpointing, and detailed expert utilization logging.

1 Introduction

Mixture-of-experts (MoE) Transformers scale model capacity by activating only a subset of parameters per token, improving compute efficiency relative to dense scaling [2, 1]. However, standard MoE can suffer from *expert isolation*: independently processed experts may under-utilize complementary information and can make credit assignment across experts more difficult.

This work proposes **Mixture-of-Collaboration (MoC)**, which keeps sparse Top-K routing but introduces explicit interaction between the selected experts prior to fusion. Additionally, we propose **Iterative Reasoning Loops (IRL)**, a simple refinement mechanism inside each expert that increases effective computation depth without adding parameters.

Contributions.

- A collaborative sparse feed-forward block (MoC) that performs message passing across the Top-K expert states (optionally via a mediator token) before fusion.
- A lightweight IRL expert design that performs S refinement micro-steps with stable residual scaling.
- A practical training system (`train_moc.py`) supporting DDP, bf16 autocast, fused AdamW, checkpointing, and expert routing diagnostics.

2 Background

2.1 Decoder-only Transformers

We use a standard decoder-only Transformer backbone [3], with Pre-LN normalization, rotary positional embeddings, and grouped-query attention (GQA).

2.2 Mixture-of-Experts

In MoE Transformers, a router selects experts for each token and combines their outputs. Typically, selected experts are evaluated independently, and outputs are combined using router probabilities. Auxiliary losses are often used to encourage balanced utilization and stable routing.

3 Related Work

Sparse expert models. MoE layers scale parameter counts while keeping per-token compute bounded by activating only a small subset of experts. This line of work explores router designs, auxiliary losses to promote balanced utilization, and systems techniques for efficient dispatch and all-to-all communication.

Mitigating expert isolation. While MoE improves scaling efficiency, independently processed experts can become specialized in ways that limit cross-expert information sharing. Lunaris MoC directly targets this issue by adding explicit collaboration (message passing) among the selected expert states prior to fusion.

Iterative computation without parameter growth. IRL is inspired by the general idea of increasing effective depth via iterative refinement or recurrent computation while keeping parameter count fixed. In Lunaris MoC, IRL is localized to each expert FFN and uses a simple residual scaling for stability.

4 Method

4.1 Model overview

The full model is implemented in `model_moc.py` as `LunarisCodex`. Each Transformer block contains (i) causal self-attention, and (ii) a sparse MoC feed-forward sublayer.

4.2 Iterative Reasoning Loops (IRL)

Inside each expert, we apply an IRL feed-forward network. Given an input token state x, we define:

$$h^{(0)} = x, \tag{1}$$

$$h^{(s+1)} = h^{(s)} + \alpha\,\mathrm{FFN}\big(h^{(s)} + x\big), \quad s = 0, \ldots, S-1, \tag{2}$$

$$\mathrm{IRL}(x) = h^{(S)}, \quad \alpha = 1/\sqrt{\max(1, S)}. \tag{3}$$

This increases effective depth while keeping parameters fixed.

4.3 MoC: Mixture-of-Collaboration

For each token, a router produces logits over experts:

$$\ell = W_r x, \quad p = \mathrm{softmax}(\ell). \tag{4}$$

We select Top-K experts using `torch.topk` and compute normalized Top-K probabilities. Each selected expert applies IRL to produce expert states $e_1, \ldots, e_K$.

Capacity control. To bound per-expert workload, we impose a capacity C based on a capacity factor and drop excess token–expert assignments. Dropped assignments contribute zero output and can be penalized via an optional drop-rate term.

Collaboration. Instead of treating e_k as independent, MoC performs R rounds of message passing across the selected expert states (optionally appending a learnable mediator token). In our implementation, each round applies multi-head attention followed by an MLP with residual connections and RMSNorm.

Fusion. Let $\tilde{e}_k$ denote refined expert states after collaboration. We compute a weighted expert aggregate $\bar{e} = \sum_k w_k \tilde{e}_k$ and a mediator representation m (either the learned mediator output or the mean). A learned gate mixes them:

$$\gamma = \sigma(W_g x), \qquad y = \gamma\, m + (1 - \gamma)\, \bar{e}. \tag{5}$$

4.4 Auxiliary router losses

We use standard router regularizers, including a load-balancing term and a logit scale regularizer ("z-loss"). The total training loss is:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_{\text{aux}}\, \mathcal{L}_{\text{balance}} + \lambda_{\text{z}}\, \mathcal{L}_{\text{z}} + \lambda_{\text{drop}}\, \mathcal{L}_{\text{drop}}. \tag{6}$$

5 Training system

The training pipeline is implemented in `train_moc.py`.

5.1 Data format

Training expects pre-tokenized shards stored as `.npy` arrays of token IDs, memory-mapped on CPU. The dataset yields $(x, y, \text{valid\_len\_y})$, and targets beyond `valid_len_y` are set to `ignore_index=-1` *on GPU.*

5.2 Optimization and precision

We use AdamW with cosine decay and warmup. On supported GPUs, training runs with bf16 autocast and enables TF32 for matmuls. Router parameters are trained with reduced learning rate and no weight decay.

5.3 Distributed training and checkpointing

Training uses `torchrun` DDP, with `no_sync()` during gradient accumulation. Checkpoints are saved periodically as a numbered file (`ckpt_<step>.pt`) and as `latest_checkpoint.pt`.

5.4 Diagnostics for expert routing

We log expert utilization (fraction of routed pairs per expert) and, when $K \geq 2$, a co-occurrence heatmap over expert pairs. We also report an estimate of *active parameters per token* given Top-K routing.

6 Experimental Protocol (work in progress)

At the time of writing, Lunaris MoC is a research-and-engineering project under active development, and we do not yet report a full benchmark suite. We include this section to document *how* the model should be evaluated and to make future results directly comparable.

6.1 Reproducibility checklist

- **Code**: `model_moc.py` (architecture) and `train_moc.py` (training).
- **Config**: a YAML file specifying model/training hyperparameters (see repository README).
- **Data**: pre-tokenized `.npy` shards of token IDs, memory-mapped by the dataset.
- **Seeds**: set per-rank seeds for PyTorch/NumPy.
- **Precision**: bf16 autocast when available; TF32 enabled for matmuls on Ampere+.
- **Checkpointing**: numbered checkpoints plus a rolling `latest_checkpoint.pt`.

6.2 What to report (recommended)

When results are available, we recommend reporting:

- Training and validation cross-entropy and perplexity.
- Throughput (tokens/sec global and per GPU) and peak memory.
- Router statistics: expert utilization, drop rate (if enabled), and expert co-occurrence heatmaps.
- The estimate of *active parameters per token* given Top-K routing.

6.3 Ablations (recommended)

The design exposes several natural ablations:

- Collaboration steps $R \in \{0, 1, 2, 4\}$.
- Mediator token on/off.
- IRL steps $S \in \{1, 2, 4\}$.
- Top-K routing $K \in \{1, 2, 4\}$ and number of experts E.
- Capacity factor and drop penalty weight.

6.4 Baselines without extra resources

If compute is limited, a minimal baseline suite can still be informative:

- A **dense** Transformer using the same backbone but with a standard (dense) FFN.
- A **non-collaborative MoE** baseline where experts do not communicate (i.e., replace collaboration with a probability-weighted sum).

These baselines can often be trained at smaller scale (fewer layers or shorter runs) to validate the modeling hypothesis.

7 Limitations

- **No full benchmark yet**: we do not claim state-of-the-art performance without standardized evaluations.
- **Engineering overhead**: collaboration adds extra compute compared to a plain MoE layer (especially for larger K or more collaboration steps).
- **Routing sensitivity**: expert utilization and stability depend on auxiliary loss weights, capacity, and data regime.
- **Interpretability**: while utilization/co-occurrence logs provide signals, understanding what each expert learns remains an open problem.

8 Ethical considerations

Lunaris MoC is a general-purpose language modeling architecture. Any deployment should consider dataset licensing, privacy, and potential downstream misuse. We encourage users to follow best practices for data governance and to evaluate safety characteristics appropriate to their application.

9 Funding statement

This work was developed without external funding.

10 Conclusion

Lunaris MoC introduces collaboration among routed experts and iterative refinement inside experts. The implementation is designed to be research-grade yet production-minded, with detailed routing diagnostics and a practical DDP training pipeline.

Open-source release

This project is released under the Apache License 2.0. Code is organized as `model_moc.py` (architecture) and `train_moc.py` (training system).

References

[1] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

[2] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, and Geoffrey Hinton. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.