EQUILIBRIUM LANGUAGE MODELS

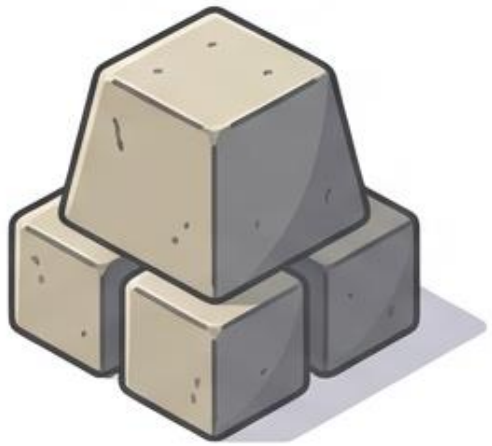
Yikun Jiang[1], Huanyu Wang[1], Tianhong Ding[1], Wenhu Zhang[2], Yiming Wu[3], Hanbin Zhao[4,*], John C.S. Lui[5]

[1]Huawei Technologies Co., Ltd., [2]The Hong Kong University of Science and Technology, [3]The University of Hong Kong, [4]Zhejiang University, [5]Chinese University of Hong Kong

Email: jiang_yikun@pku.edu.cn

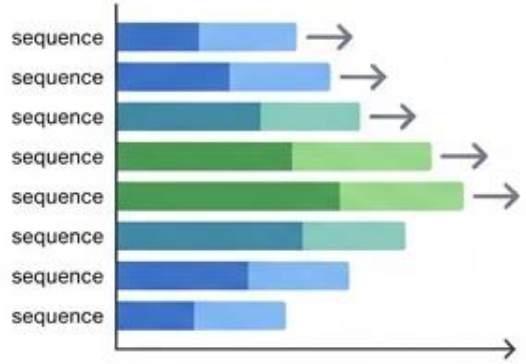
Open Source: github.com/Jyk-122/ELM

Edge deployment is fundamentally constrained by memory limits.



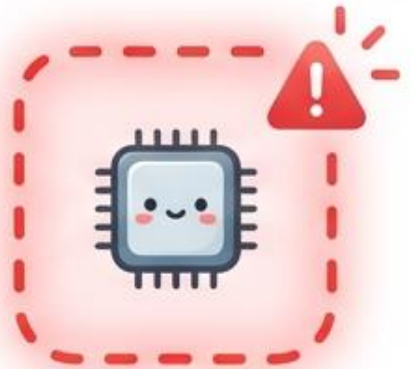
Parameter Weights
(Static RAM overhead)

+



KV Cache
(Dynamic, growing RAM overhead)

=



Edge Device Memory Budget
Capacity Exceeded

Traditional compression methods force a choice: lose critical reasoning accuracy (via pruning) or save zero parameter memory (via dynamic skipping).

The Flawed Compromises

Statistical Pruning

+ Reduces overall model size.

- Severe brain drain on challenging reasoning tasks (Math/Code) due to capacity loss.

Dynamic Skipping

+ Reduces computational cost during inference.

- Almost zero parameter memory savings. Ineffective for strict edge memory budgets.

Our Goal: The Ultimate Balance

High parameter savings AND high accuracy retention simultaneously.

Reinterpreting Deep Computation

We shift from explicit forward passes to solving for a dynamic equilibrium state.

Dense Model

Explicit Transformer Layers

Replaced by

Equilibrium Language Model

Implicit Fixed-Point Layer

Instead of recklessly pruning individual components, ELMs replace entire groups of Transformer layers with a single, lightweight fixed-point network.

The Math Behind the Magic

Infinite depth achieved with finite parameters.

The single, lightweight Fixed-Point Layer.

$$z^* = f(z^*, x)$$

The Equilibrium State (Final Output).

The conditional input token.

By defining the output as an equilibrium solution, the network analytically backpropagates through z^*. It achieves the representational capacity of an infinitely deep explicit stack—using a fraction of the parameters.

Making ELMs Real: Two Critical Challenges

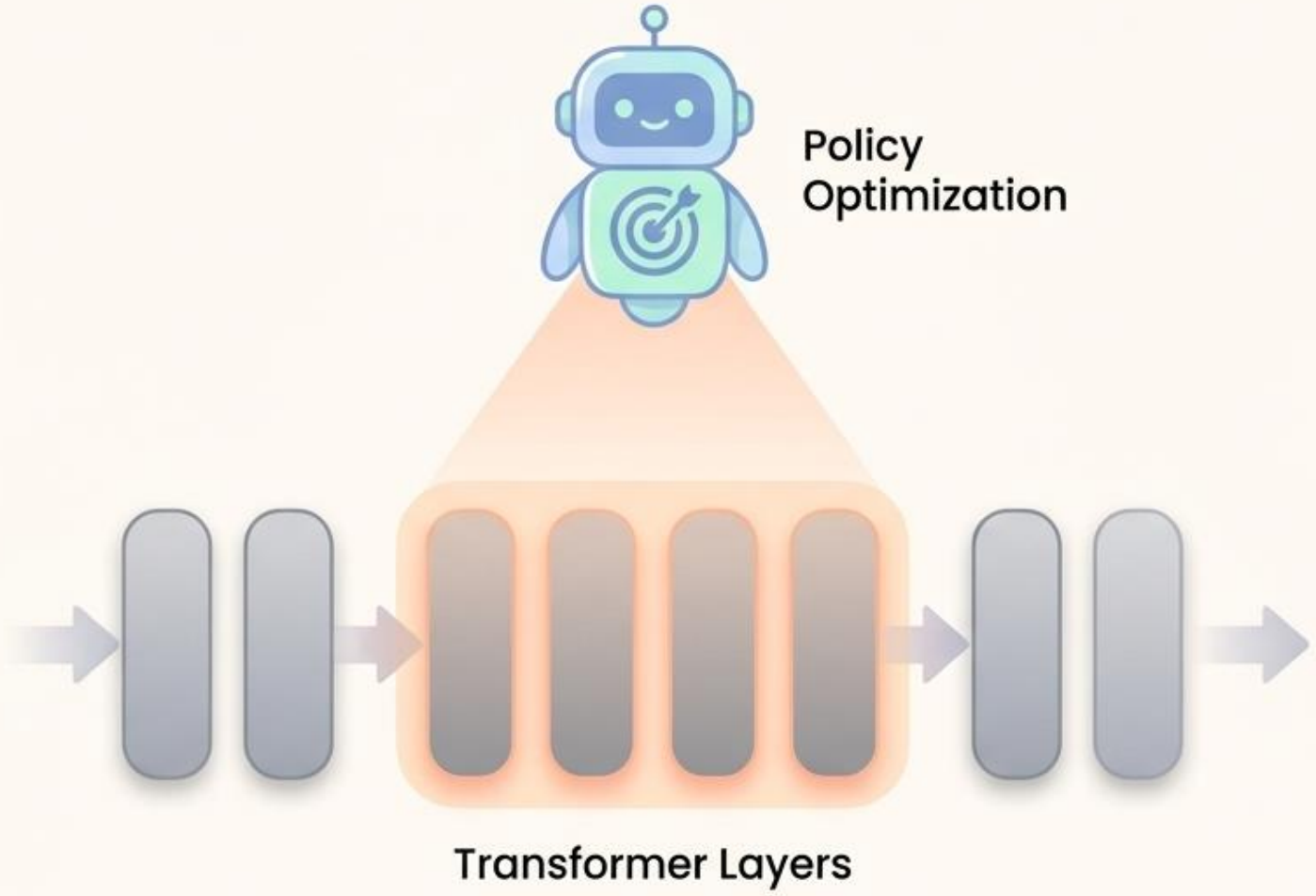
Challenge 1: Layer Selection

Which specific layer intervals should be compressed into the fixed-point network? Standard heuristics fail to capture the approximation error.

Challenge 2: KV Cache Explosion

How do we handle the massive KV cache memory accumulation caused by looping the fixed-point layer multiple times during inference?

Challenge 1: Group Pruning Policy Optimization (GPPO)



Mechanism:

A Reinforcement Learning framework that treats optimal layer selection as an actionable policy.

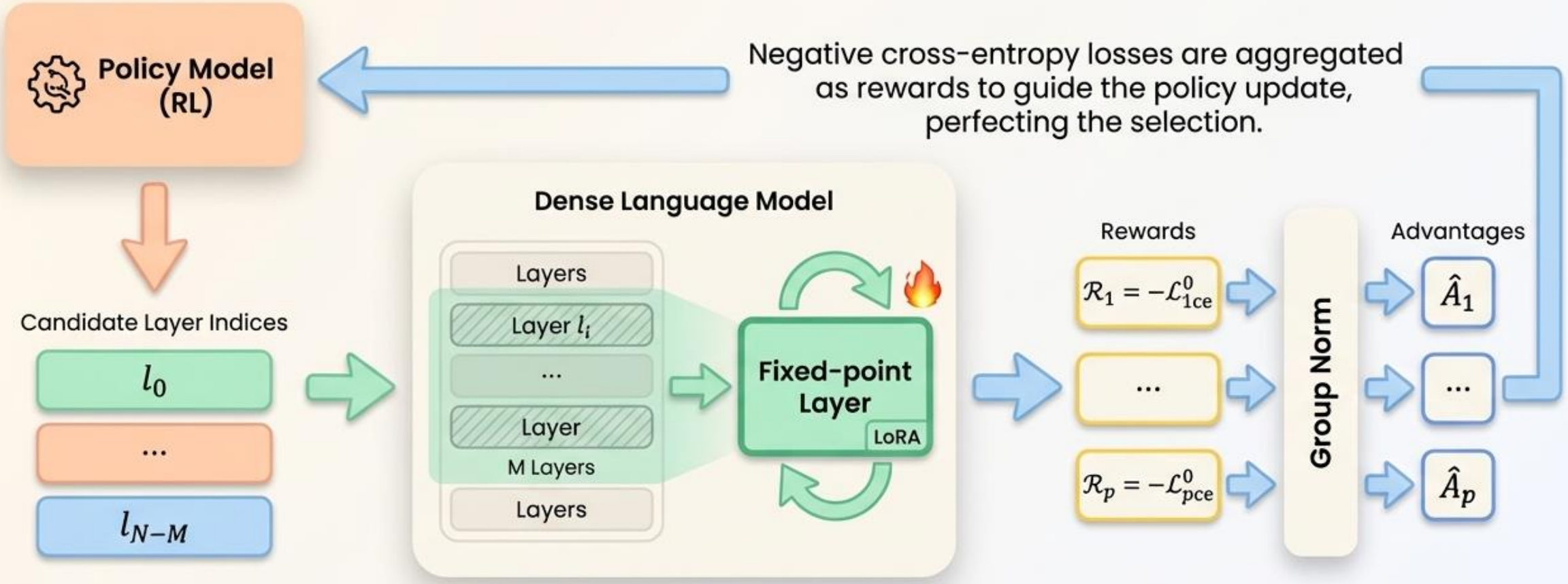
Exploration:

It autonomously samples layer intervals from a learnable policy distribution, hunting for the perfect compression target.

Reward Signal:

It constantly adapts to maximize downstream performance by using a supervised fine-tuning loss as the defining reward.

The Autonomous GPPO Workflow



Sampling

Policy model autonomously selects candidate layers.

Reward Model

The ELM is constructed, and the Fixed-point layer is trained efficiently with LoRA.

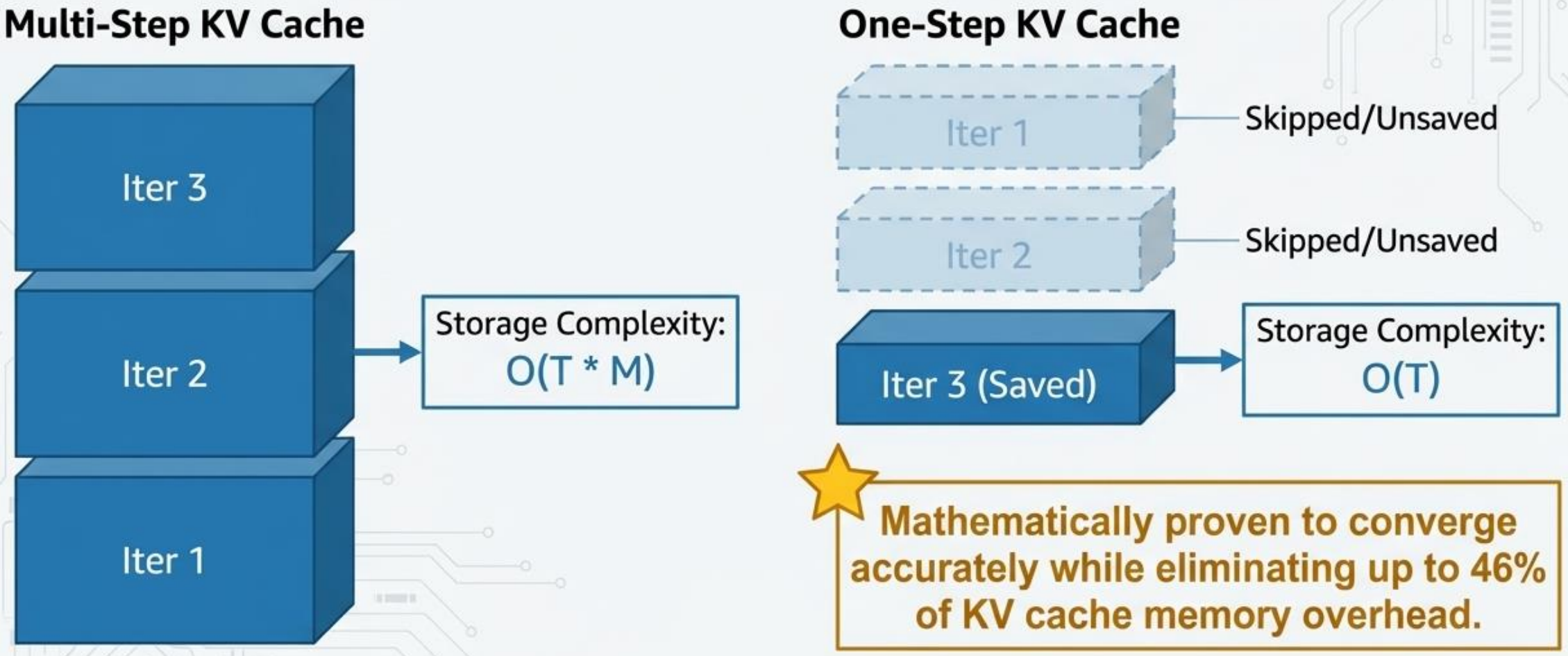
Update

⚠️ Challenge 2: The KV Cache Explosion

- **The Problem:** Fixed-point solvers iterate multiple times per token to reach equilibrium.
- **The Threat:** Standard inference would store the KV cache for every intermediate iteration step (O(T x iters)). This entirely defeats our memory-saving goals and crushes the edge device's RAM.

Innovation 2: One-Step KV Cache

Standard fixed-point loops store cache for every step, destroying memory efficiency. One-Step KV retains only the final iteration cache.



Accelerating the Equilibrium

Faster edge inference with adaptive termination

1

Adaptive Termination

Thanks to One-Step KV, we implement token-adaptive iterations. Fast-converging tokens exit the loop early, dynamically saving compute.

2

Advanced Solvers

Utilizing Anderson Acceleration further reduces computational cost by ~38% with negligible accuracy loss. Edge inference becomes incredibly fast.

Accelerating the Training (SJFB)

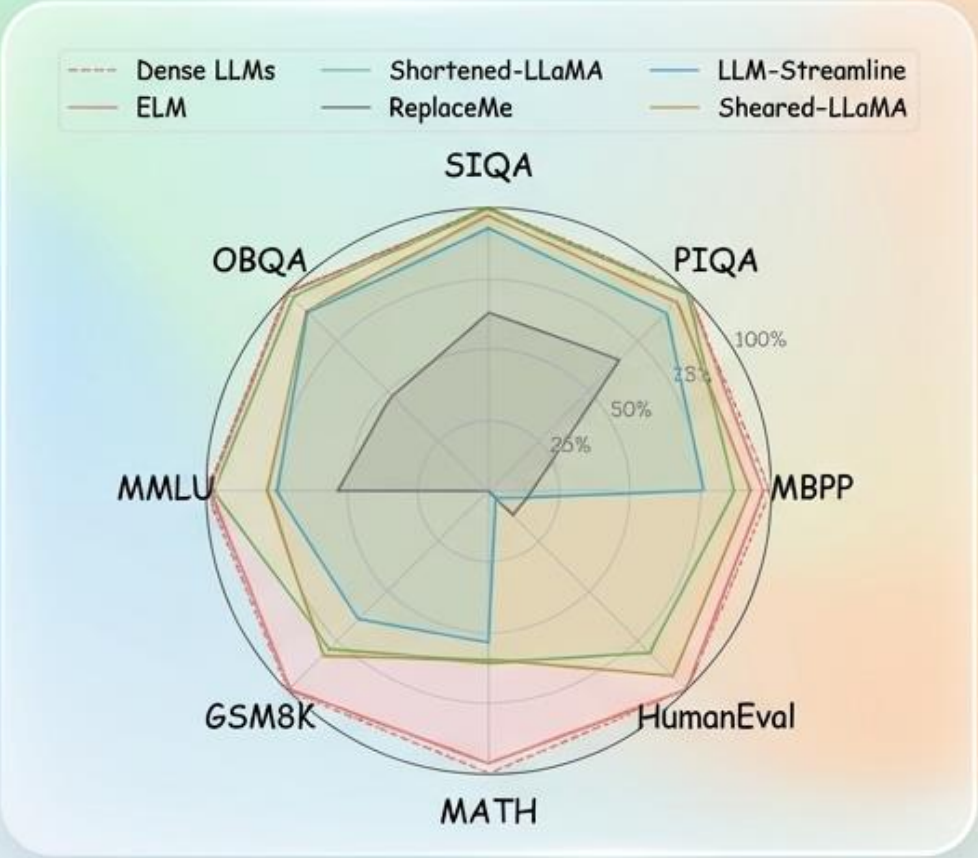
Stochastic Jacobian-Free Backpropagation (SJFB)

- **How it Works:**
  Training implicit gradients usually requires expensive matrix inversion. SJFB ingeniously drops the inverse Jacobian, computing gradients only at the final step.
- **The Impact:**
  - ↘ Slashes training time by 14%
  - ↘ Reduces memory cost by 13%

The Ultimate Test: 99% Accuracy Retained

28%

Fewer Parameters.

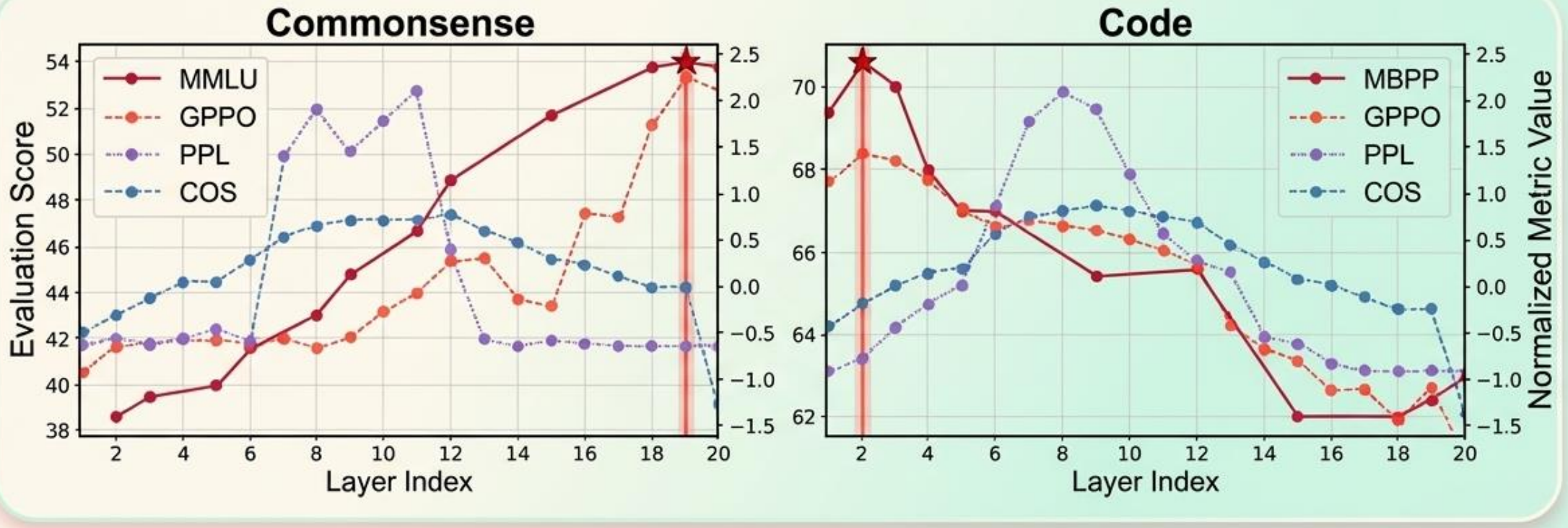


99%

Average Accuracy Maintained.

Across Common Sense, Math, and Code benchmarks, ELMs drastically outperform traditional pruning baselines (Shortened-LLaMA, LLM-Streamline, Sheared-LLaMA), nearly matching the dense model's full footprint.

The Efficacy of GPPO



The Evidence: GPPO (red line) consistently outperforms heuristic metrics like Cosine Similarity and Perplexity.

The Insight: It accurately identifies the optimal layer choices (l^*) for entirely different downstream tasks, perfectly correlating with empirical enumeration.

Unlocking True Edge Efficiency

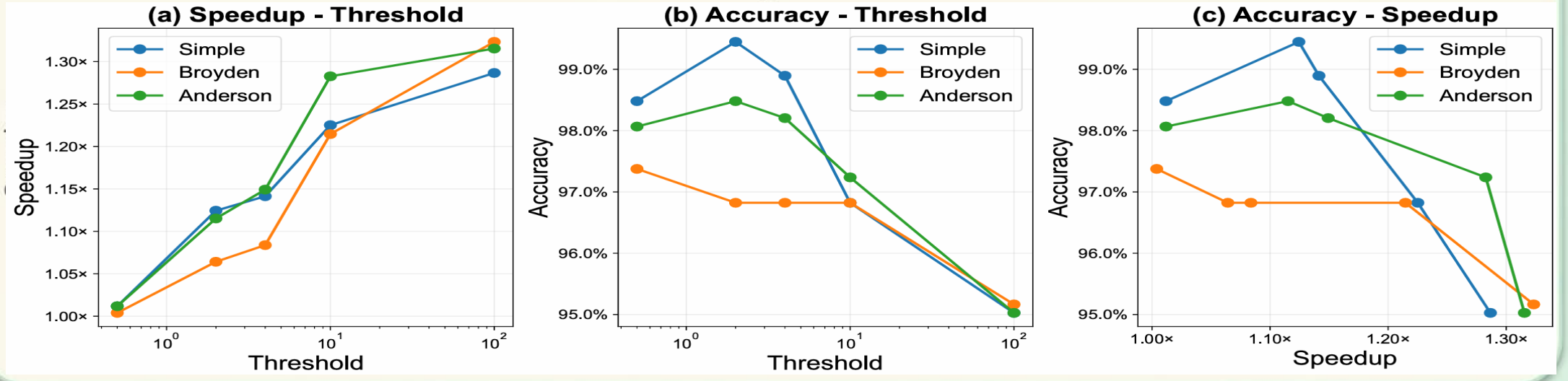
⬇️ 25% KV Cache RAM

One-Step KV reduces cache overhead by up to 25% at a 4K context length.

⚡ 28.3% Speedup

The Anderson solver achieves massive decoding speedups while retaining 97% accuracy.

Pareto Efficiency and Trade-offs



Conclusion & Future Horizons

- **Redefining Compression:** ELMs successfully compress LLMs via fixed-point theory without sacrificing deep reasoning.
- **Edge Ready:** GPPO + One-Step KV Cache completely resolve the training and memory bottlenecks of iterative models.
- **The Future:** Generalizing to pre-training and exploring 100% fixed-point LLM architectures.

Explore the Code: https://github.com/Jyk-122/ELM