AReaL-DTA: Dynamic Tree Attention for Efficient Reinforcement Learning of Large Language Models

Up to **8.31**× training throughput by *not* computing the same prefix twice

Jiarui Zhang

June 4, 2026

AReaL Team, Ant Group
jiarui-z23@mails.tsinghua.edu.cn

Outline

Motivation: long shared prefixes recomputed in every forward & backward

RL Post-training Is Powerful — and Painfully Expensive

RL has become the engine driving the next wave of LLM capabilities:

- Math reasoning, code generation
- Multi-turn agentic tasks
- Long chain-of-thought training

But on the systems side, the bill is huge:

- **Compute heavy (training)**: long context's activations
- **Memory heavy (rollout)**: long context's KV cache
- **Throughput balance**: need to balance training and rollout in asynchronous scenarios

The hidden waste

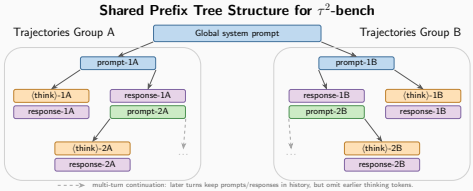
Rollouts for the same prompt share **long prefixes** — yet today's frameworks can recompute them in *every forward and every backward pass*.

Can we compute the shared prefix only once?

Where Does the Sharing Come From?

In multi-turn agentic RL (τ^2-Bench)

- Long **global system prompt** ($\sim$ 4.8K tokens, identical for all rollouts)
- Multi-turn dialogue history is shared across continuations, except at points where earlier-turn *thinking* tokens are dropped—these are the divergence points



Multi-turn rollouts naturally form a **prefix tree**.

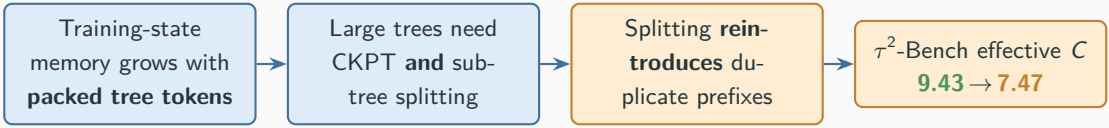
Compression rate on raw τ^2-Bench rollouts:

9.43$\times$ $\iff$ **89.4**% prefix sharing

Why Sparse Tree-Mask Training Still Falls Short

Sparse Tree-Mask Training Still Scales Poorly

Packing the rollout prefix tree with a tree attention mask avoids duplicate prefixes. In *training*, the chain of consequences still limits the gain:



Why CKPT? For 4B/8B/14B models, even a single 16K sequence has large training states, so Sparse needs activation recomputation before it can fit sequence-level backpropagation.

Weakness of activation recomputation: it is layer-wise, not token-wise — so it cannot exploit tree-structured, shared-prefix rollouts for KV reuse.

Hardware-side penalty: Sparse tree-mask execution has irregular access patterns and lower MFU than optimized dense-attention kernels.

*We need memory bounded by the **longest path**, not the whole tree.*

AReaL-DTA: Dynamic Tree Attention

Key Idea: Walk the Tree — Don't Materialize It

Traditional training: one full forward → one full backward, per microbatch.

AReaL-DTA breaks this contract:

- Build a **prefix tree** of all rollouts.
- Traverse it with **depth-first search (DFS)**.
- Interleave *forwards* with *backwards*.

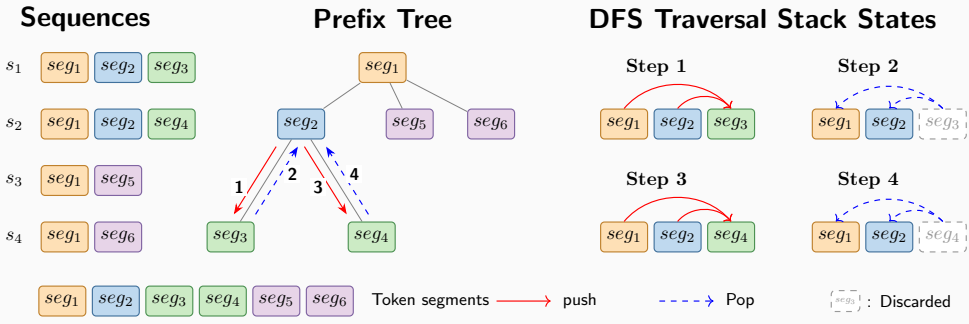
Memory is bounded by the **longest root-to-leaf path**, not the total token count of the tree.

Push Walk down a branch; reuse cached prefix KV; forward only the new token segment and push it onto the stack.

Visit At each leaf: compute the loss and trigger backward immediately.

Pop Backpropagation accumulates gradients in shared prefix nodes; release activations from the last stack token segment, but keep the prefix state for the next push (i.e., it's sibling branch).

DFS Traversal: One Path at a Time



Shared prefix is computed **once** and reused across siblings.

Why This Saves Memory — and Compute

Space complexity

Naive tree-mask $\mathcal{O}$(total tree tokens)

AReaL-DTA $\mathcal{O}$(longest path)

$\Rightarrow$ no need to fully materialize attention masks or activations of the whole tree.

Compute reuse

- Each shared-prefix node is forwarded **once**.
- Each leaf's loss gradient is backpropagated immediately.
- Prefix gradients are correctly *accumulated* from all descendants.

Live computation graph = only the path we are currently descending.

Vanilla DTA Backward: Push–Pop DFS

Algorithm: Vanilla Dynamic Tree Attention

initialize active stack S
procedure $\mathrm{DFS}(u)$
  if u is a leaf: attach $\mathcal{L}(u)$
  for child v of u:
    $\mathrm{FWD}(\mathrm{depth}(u) \to \mathrm{depth}(v))$
    push segment $[\mathrm{depth}(u)+1, \mathrm{depth}(v)]$
    $\mathrm{DFS}(v)$
    $\mathrm{BWD}(\mathrm{depth}(v) \to \mathrm{depth}(u))$
    pop segment $[\mathrm{depth}(u)+1, \mathrm{depth}(v)]$
$\mathrm{DFS}(\mathrm{root}(\mathcal{T}))$

Vanilla DTA already reuses shared prefixes, but it couples every tree edge with one forward and one backward.

This creates many short $\mathrm{FWD}/\mathrm{BWD}$ rounds when moving between leaf trajectories.

Backward DFS Order

Algorithm: BackwardDFSOrder

for node v in postorder($\mathcal{T}$):
  if v is a leaf: $\mathrm{tail}(v) \leftarrow \mathrm{depth}(v)$
  else:
    order children u increasingly by $(\mathbf{1}[u \text{ is internal}], \mathrm{tail}(u))$
    leaf child first; among the same type, smaller tail first
    reversed DFS order places short/leaf-only branches late
    so they can be popped directly without extra
    FWD-NoGrad *cache anchors*
    $\mathrm{tail}(v) \leftarrow \mathrm{tail}$(first child under this order)
$\rho \leftarrow$ leaf order from DFS using this child order
return reverse(ρ)

Greedy heuristic from the paper:

- Minimize the number of extra forward passes by maximizing reuse of shared prefixes.
- Balance the backward pass segment lengths to avoid frequent short backward steps, which can become memory-bound.

Prioritize branches that lead to fewer extra forward passes and balance gradient accumulation more evenly.

Optimized DTA Backward: Leaf Order + Chunked Pop

Algorithm: Dynamic Tree Attention Backward

Require: prefix tree $\mathcal{T}$, losses $\{\mathcal{L}(\mathbf{s})\}$, block size B
procedure POP_BYCHUNK($\ell_{\text{backward}} \to \ell_{\text{lcp}}$)
  choose boundaries $t_0 = \ell_{\text{lcp}} < t_1 < \cdots < t_m = \ell_{\text{backward}}$ with $t_{j+1} - t_j \le B$
  for $k = m-1, \ldots, 0$:
    FWD($t_k \to t_{k+1}$) *construct graph for this segment*
    BWD($t_{k+1} \to t_k$) *propagate incoming gradients and attached losses*
    pop suffix within $[t_k + 1, t_{k+1}]$
end procedure
$\pi \leftarrow$ BACKWARDDFSORDER($\mathcal{T}$); initialize active stack S and $\ell_{\text{backward}} \leftarrow 0$
for $i = 0, \ldots, |\pi| - 1$:
  $\mathbf{s} \leftarrow \pi_i$; $n \leftarrow |\mathbf{s}|$
  if $i > 0$: $\ell_{\text{lcp}} \leftarrow \text{LCP}(\pi_{i-1}, \mathbf{s})$; POP_BYCHUNK($\ell_{\text{backward}} \to \ell_{\text{lcp}}$)
  else: $\ell_{\text{lcp}} \leftarrow 0$
  FWD($\ell_{\text{lcp}} \to n$); push suffix $\mathbf{s}[\ell_{\text{lcp}} + 1 : n]$ onto S
  attach $\mathcal{L}(\mathbf{s})$ at position n; $\ell_{\text{backward}} \leftarrow n$
  if $i + 1 < |\pi|$: $\ell_{\text{next\_anchor}} \leftarrow \text{LCP}(\mathbf{s}, \pi_{i+1})$; **else**: $\ell_{\text{next\_anchor}} \leftarrow 0$
  $p_{\text{pop}} \leftarrow n - \ell_{\text{next\_anchor}}$
  if $p_{\text{pop}} \ge B$: $m \leftarrow \lceil p_{\text{pop}} / B \rceil$; $\Delta \leftarrow \lceil p_{\text{pop}} / m \rceil$; $\ell_{\text{next\_anchor}} \leftarrow n - \Delta$
  if $\ell_{\text{lcp}} < \ell_{\text{next\_anchor}}$: FWD-NOGRAD($\ell_{\text{lcp}} \to \ell_{\text{next\_anchor}}$) *materialize cache anchor*
end for
POP_BYCHUNK($\ell_{\text{backward}} \to 0$)

Extra FWD-NOGRAD work is bounded by anchor construction and costs around **7%** in profiled runs.

Scaling Out: Load-balanced Dispatch

Load-balanced Multi-GPU Dispatch

Problem. Split N rollouts into K trees, one per trainer GPU,
minimize the slowest GPU's cost

$$\mathcal{C} = \min \max_{j=1\ldots K} \mathcal{C}(\mathcal{T}_j)$$

Two competing goals:

- Balance per-GPU work
- Preserve prefix sharing maximally

AReaL-DTA's recipe

1. Sort rollouts in DFS order, i.e., lexicographic order $\Rightarrow$ similar prefixes are adjacent.
2. Cut the ordered list into K *contiguous* segments.
3. Binary-search the smallest threshold τ that admits a K-cut with $\mathcal{C}(\mathcal{T}_j) \leq \tau$.

Disabling this DP balancer costs **11.93%** system performance.

Evaluation: up to $8.31\times$ training throughput

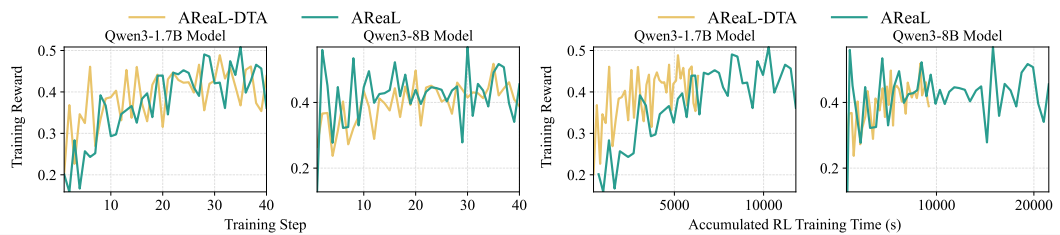
Setup

- **Workload:** τ^2-Bench multi-turn agentic RL.
- **Algorithm:** PPO; reference AReaL (state-of-the-art async RL system).
- **Models:** Qwen-3 family (1.7B, 4B, 8B, 14B).
- **Hardware:** 8× NVIDIA H800 GPUs.
- **Context length:** 16K (activation checkpointing on for AReaL and chunked-backpropagation on for AReaL-DTA).

Model	System	Parallel Strategy (rollout + train)
1.7B	AReaL-DTA	dp5-pp1-tp1 + dp3-pp1-tp1
	AReaL	dp2-pp1-tp1 + dp6-pp1-tp1
8B	AReaL-DTA	dp5-pp1-tp1 + dp3-pp1-tp1
	AReaL	dp2-pp1-tp1 + dp6-pp1-tp1

Optimal asynchronous RL training configuration for the parallel strategy.

End-to-End: Similar Reward, Much Faster



Reward vs. **training step**

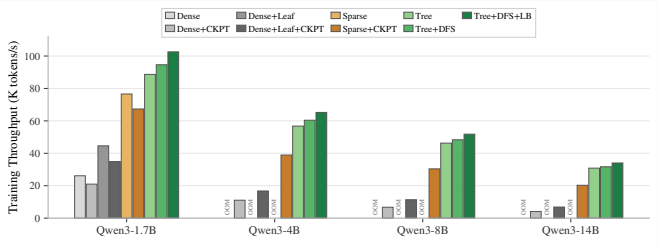
Reward vs. **wall-clock training time**

End-to-end throughput: Qwen3-1.7B: **1.28**× / Qwen3-8B: **2.28**×

Stability is preserved — AReaL-DTA does not degrade async RL training.

Under asynchronous settings, with the improved model training performance, AReaL-DTA will allocate more GPUs to the rollout generation phase for the optimal end-to-end throughput.

Training-Side Throughput: Up to 8.31×



Compared to dense baseline with activation checkpointing:

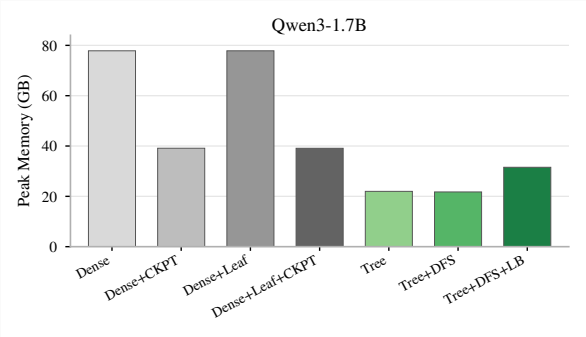
- Tree: 7.53×
- Tree + DFS order: 7.74×
- Tree + DFS order + LB: 8.31×

AReaL-DTA vs. Sparse+CKPT

1.52×–1.70× higher throughput across model sizes.

Extra forward overhead accounts for around **7%** of total training time.

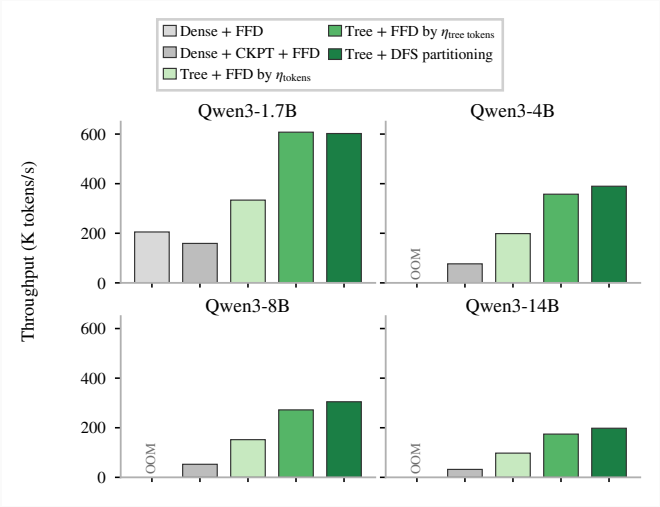
Memory: Qwen3-1.7B Peak Training Memory



- Main-paper memory ablation: Qwen3-1.7B on τ^2-Bench.
- Dense fits in this setting, so peak memory comparison is direct.
- AReaL-DTA reduces peak memory by over **50%** compared with Dense.

How? AReaL-DTA adopts *chunked backpropagation*: it recomputes chunk activations from cached prefix KV, which acts as *token-wise* gradient checkpointing and leverages tree-structured rollouts for efficient KV reuse.

Multi-GPU Scaling: Throughput Holds Up to 8 GPUs



Backward throughput on 8 GPUs (Qwen3 1.7B / 4B / 8B / 14B).

- Validated at 2 / 4 / 8 GPUs — same trend.
- DFS-aware partition keeps per-GPU work even *and* prefix reuse high.

Removing the load balancer: —**11.93%** training performance.

Takeaways

Why AReaL-DTA Matters

- RL post-training has hidden **tree-shaped** structure — treating it as a flat list of sequences wastes compute and memory.
- Sparse tree-mask training struggles to take full advantage of prefix sharing: packed training-state memory can force activation recomputation and subtree splitting.
- AReaL-DTA **dynamically** carries out tree-structured attention for shared-prefix rollouts: **depth-first traversal** of the prefix tree walks one path at a time, interleaving forwards and backwards (bounded memory by longest path).

Rethinking *how attention is executed* for RL workloads is a promising direction for the next generation of LLM training systems.

AReaL-DTA: Open-Source Branch & Configuration

Paper: `arxiv.org/abs/2602.00482`

Code (development branch with DTA, to be merged):

`github.com/areal-project/AReaL` (`feat/dta`)

DTA + DFS-aware packing: For both `actor` **and** `critic`, set:

- `tree_training_mode: dta`
- `packing_algorithm: dta`

Example YAML (τ^2-Bench):

`examples/tau2/dta`

Thank you!

AReaL-DTA

Questions?