Performance Foundations of Parallel & Distributed Reasoning Language Models

Maciej Besta†*, Leonard Schmidt*, Lara Nonino, Robert Gerstenberger, Pierre Pang, Patrik Okanovic, Ales Kubicek, Tiancheng Chen, Baraq Lipshitz, Torsten Hoefler

ETH Zurich †*Corresponding author* **Core contributions*

Abstract—Reinforcement Learning with Verifiable Rewards (RLVR) and other RL-style post-training paradigms have been used for aligning large language models (LLMs) with reasoning standards. The resulting recent *Reasoning Language Models* (RLMs) such as DeepSeek-R1, o3, and Kimi k1.5 show that such RL-style post-training ("RL-for-LLMs") can substantially improve chain-of-thought reasoning, long-horizon planning, and self-correction. However, the computational footprint of these systems is massive: state-of-the-art RLM training requires millions of GPU-hours and tightly coupled multi-model pipelines that stress modern hardware far beyond classical supervised LLM training. This makes RLM training as much a parallel and distributed systems problem as an algorithmic one. In this work, to facilitate developing RLMs that are simultaneously high-performance, scalable, and cost-effective, we first systematize the RL-for-LLM paradigm and provide a compute-centric analysis of prominent post-training algorithmic frameworks: Proximal Policy Optimization (PPO), Group Relative Policy Optimization (GRPO), as well as their variants. Second, we develop a taxonomy of *intra-* and *inter-model* parallelism strategies for RL-for-LLMs, covering both traditional techniques (data, tensor, pipeline, sequence, context, and expert parallelism) as well as novel forms of parallelism and optimization techniques for multi-model RLM training, for example disaggregated placement, stage fusion, hybrid parallelism, and asynchronous execution. We harness the work–depth model of parallel computing to make our taxonomy and its insights rigorous and portable. Finally, we analyze existing RLM frameworks and we distill practical guidelines and outline open research directions for building scalable, fast, and cost-effective RLMs.

Index Terms—Parallel Reasoning Language Models, Distributed Reasoning Language Models, Parallel RLVR, Distributed RLVR, Asynchronous Reasoning Language Models, Asynchronous RLVR.

1 Introduction

Reinforcement learning (RL) is now the computationally dominant post-training paradigm for aligning large language models (LLMs) with reasoning standards. After pre-training and supervised fine-tuning (SFT), RL optimizes a policy against learned reward models or other feedback signals to improve task-specific performance beyond next-token prediction [20, 192]. The resulting frontier *Reasoning Language Models* (RLMs), such as DeepSeek-R1 [107], OpenAI's o3 [188], and Kimi k1.5 [86], demonstrate that carefully designed preference learning and policy optimization over Chain-of-Thought trajectories can produce substantially stronger reasoning, long-horizon planning, and self-correction than SFT alone. A key paradigm behind this trend is *RL with Verifiable Rewards* (RLVR), in which reward signals are obtained from automatically checkable outcomes, such as exact-answer verification in mathematics, unit tests in code, or other task-specific verifiers [118, 173, 243].

The computational footprint of RLMs is enormous [156]. For example, OpenAI reports that in developing o3 it scaled RL by an *additional order of magnitude* in both train-time RL compute and inference-time reasoning compute, while still observing clear gains from further scaling [187]. More broadly, cost analyses of frontier model training indicate that end-to-end training runs for leading systems already cost tens to hundreds of millions of US dollars in compute alone, and they may reach more than $1B by 2027 [73].

In contrast to classical supervised LLM training, RLM pipelines [26] couple several large models and stages. A standard Proximal Policy Optimization (PPO)-style setup maintains four distinct LLMs (actor (policy), reward, critic (value), and a reference model) and repeatedly executes them across three stages (Generation, Assessment[1], Training) with complex data dependencies. Recent reasoning-oriented systems often use critic-free Group Relative Policy Optimization (GRPO)-style training in practice, largely because removing the critic reduces memory pressure and simplifies large-scale rollout training [107, 213]. Nevertheless, PPO-style actor–critic training remains a key and relevant design point as its learned value baseline reduces gradient variance and stabilizes policy updates, providing more faithful token-level credit assignment than critic-free alternatives such as GRPO [79, 239]. PPO has been used or revived in reasoning-focused RL systems such as VAPO [268]. Overall, both PPO and GRPO as well as their variants are crucial RLM training frameworks.

Systems used for RL post-training, for each ReaL [169], RLHFuse [282], and OpenRLHF [117], have shown that naively reusing supervised-training parallelization strategies for RLMs yields suboptimal GPU utilization. Instead, the RLM workflow demands joint reasoning about model placement, inter-model scheduling, and heterogeneous workload characteristics. This combination of massive scale and multi-model RL pipelines makes RLMs fundamentally a *parallel and distributed systems* problem.

[1]While the Assessment stage is often called "Inference" in the literature, we use the term "Assessment" because it more precisely describes its functional role. Indeed, inference is not specific to this stage: Generation also performs inference, albeit autoregressively.

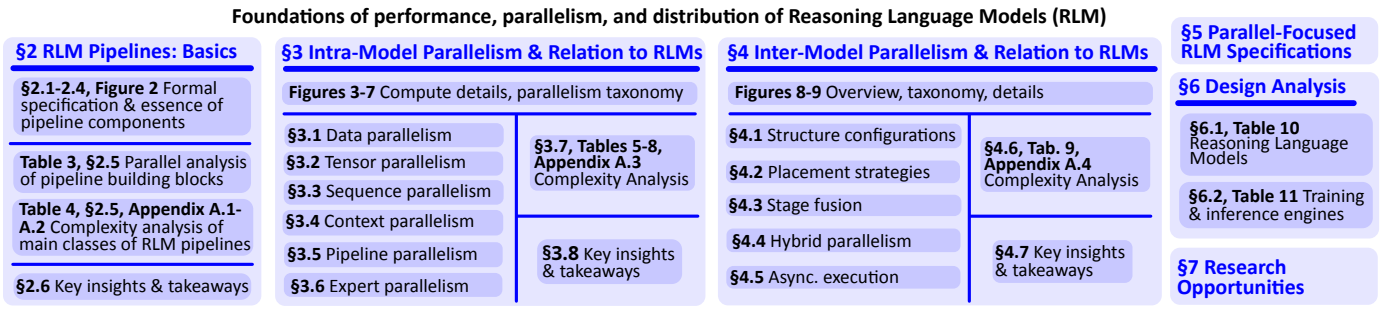


Fig. 1: **Overview of the contributions and analyses in this work.**

To facilitate the development of next-generation scalable and cost-effective RLMs, we conduct an in-depth investigation into the foundations of their parallel and distributed computational characteristics; a roadmap of the paper can be found in Figure 1. First, we systematize the RL-for-LLM paradigm and provide a compute-centric analysis of several prominent post-training frameworks (**contribution 1**), including PPO-style [279] and GRPO-like methods [213]. We also consider preference-gradient approaches such as Direct Preference Optimization (DPO) [250] – while these methods do not use RL, they have been shown to also enhance reasoning capabilities [134, 238, 242, 252].

Next, we develop a taxonomy of parallelism for RLMs (**contribution 2**). The first part focuses on **intra-model** parallelism, instantiated by LLM-based policies, critics, rewards, and references. For this, we analyze traditional forms of parallelism and how they interact with the RL pipeline, for instance which RL stage benefits most from which form of intra-model parallelism. Here, we consider data parallelism (DP) [146], tensor parallelism (TP, also referred to as operator parallelism [25, 219], pipeline parallelism (PP) [120, 176], sequence parallelism (SP) [132], context parallelism (CP) [160], and expert parallelism (EP) [141]).

The second part of our taxonomy targets **inter-model parallelism**: opportunities to run different models and RL stages concurrently, and system-level optimizations that reduce end-to-end iteration time by reshaping how these components interact. We show that the available forms of inter-model parallelism are largely determined by a small set of concrete design choices: *model structure configurations*, which can eliminate redundant model executions (e.g., critic–reward merging), *model placement strategies*, which enable component-level concurrency by co-locating models on the same devices or disaggregating them onto separate device groups/services; *stage and sub-task fusion*, which introduces stage-level overlap (e.g., overlapping long-tailed generation with reward/value evaluation, or pipelining actor and critic updates) to remove serialization between stages; *hybrid intra-model configurations across components*, where different models and stages use different mixtures of intra-model parallelism; and *asynchronous execution*, which decouples rollout generation from learning via bounded staleness. Our taxonomy provides a common performance-oriented language for formulating, comparing, and implementing new optimizations for large-scale RLM training.

To ensure that all the insights are portable across different architectures, we use work/depth/memory complexities for all the parts of the taxonomy (**contribution 4**).

We then use the taxonomy to conduct a systematic analysis of existing RLM models and frameworks and analyze them through the lens of the parallelism and optimization strategies they implement (**contribution 5**). Concretely, we examine general-purpose libraries such as TRL [231], ColossalChat [267], DeepSpeed-Chat [261], OpenRLHF [117], HybridFlow [217], and NeMo RL [184, 214], as well as more specialized systems including ReaL [169], RLHFuse [282], FlexRLHF [251], StreamRL [281], Asynchronous RLHF [180], AReaL [92], and Pipe-RLHF [262]. For each framework, we catalogue which intra- and inter-model techniques are supported. This enables practitioners to make informed choices about which framework best matches their needs, and it reveals systematic gaps where our taxonomy suggests additional optimizations (for example, underexplored stage-fusion and placement policies for multi-model PPO-style pipelines). In this way, our study not only describes the current ecosystem but also points to concrete directions for enhancing existing RL frameworks and designing new ones.

1.1 Complementary Analyses & Related Work

There exist general studies on the blueprint for RLMs [26] and broad surveys on LLM-enhanced RL [61, 237]. Other works provide comprehensive analyses of specific components of the RL pipeline [232, 250, 279].

Some surveys detail techniques for resource-efficient LLMs, covering algorithms, model compression, and systems [17, 83, 253]. More broadly, [25] offers a foundational concurrency analysis of parallel and distributed deep learning. However, these works do not focus on the unique demands of RLM pipelines.

Closer to our work, Liu et al. explore acceleration techniques for deep RL [162] and Chen et al. survey the effects of scaling LLM's reasoning capabilities [68]. However, these works do not explain the underlying performance foundations of the parallel and distributed systems that enable RLMs to scale. Specifically, the latter focuses on how reasoning capabilities emerge with scale, not on the concurrency and system bottlenecks of the RLM training loop. Similarly, the former analyzes acceleration for general deep RL, not the specific parallel architectures required by modern RLM pipelines.

We complement these studies by providing the first in-depth analysis to demystify the performance foundations of parallel and distributed RLMs.

1.2 Analysis of Parallel Algorithms

We use formal models for reasoning about parallelism. Specifically, we use *the work-depth (WD) analysis*, an established approach for bounding runtimes of parallel algorithms. The **work** (W) of an algorithm is the total number of operations and the **depth** (D) is defined as the longest sequential chain of execution in the algorithm, and it forms the lower bound on the algorithm execution time [53, 56]. One usually wants to minimize depth while preventing work from increasing too much. Finally, **Memory** accounts for model parameters, gradients, optimizer state (Adam first and second order moments), and stored activation tensors during a single training iteration.

2 Overview & Foundations of RLMs

We first overview basic RLM post-training concepts.

2.1 RL Post-Training Pipeline: Overview

The essence of RL for LLM can be captured as a simple three-stage loop that iterates continuously: *Generation* $\rightarrow$ *Assessment* $\rightarrow$ *Training* $\rightarrow$ *Generation* $\rightarrow$ $\ldots$. These stages operate over a *batch* of input prompts, denoted by $X = \{x_1, x_2, \ldots, x_B\}$, where B is the batch size. Starting from a language model that has typically been pre-trained and then fine-tuned via supervised learning (SFT), this loop further improves this model (referred to as the *policy model* π_θ) through RL-based optimization. In each iteration, the *Generation* stage produces candidate responses to input prompts using the current policy π_θ. Then, *Assessment* evaluates those responses and produces feedback (which can be both scalar and vector). Finally, *Training* uses the resulting feedback to update the policy π_θ. Both Assessment and Training stages may invoke one or more auxiliary models that provide the feedback signals used to improve the policy π_θ, i.e., the *reward model* R_φ, the *critic model* V_ψ, and the *reference model* π_{ref}. This iterative process continues until convergence criteria are met or a predetermined number of training loops are completed, with the updated π_θ model from each training stage serving as the starting point for the next cycle. We overview this process in Figure 2; additional mathematical details are in Appendix A and in Table 1.

2.2 Terminology: RLVR, RLHF, RLAIF & Others

We use ***RL-style reasoning post-training*** as the umbrella term for reasoning-oriented post-training pipelines that use feedback beyond next-token prediction. This term is broader than strict online RL: it includes **(1)** RLVR, where online RL algorithms such as PPO and GRPO-style methods use verifiers as rewards; **(2)** online RL-based post-training with model-based assessment, including RL from Human Feedback (RLHF), RL from AI Feedback (RLAIF), direct RLAIF (d-RLAIF), and learned reward methods; and **(3)** offline preference-optimization methods such as DPO, which are not RL in their basic form but have also been used to enhance reasoning. This broader umbrella is also reflected in systems practice: frameworks originally named after RLHF are now used for broader reasoning post-training. For example, OpenRLHF [117] supports PPO- and GRPO-style training, while RLHFuse [282] optimizes the same staged workflow used by both RLHF and RLVR systems.

Strict RLVR denotes online RL with rewards produced by an automatic verifier rather than a learned reward model. Examples include exact-answer matching in mathematics, unit-test execution or compilation in code, theorem proving, and other programmatic correctness checks [107, 213].

A second class uses **model-based assessment**. In classical RLHF, rewards are derived from human preference data, typically through a learned reward model [192]. In canonical RLAIF, human preference labels are replaced by AI-generated preferences or critiques, which are then used to train a reward model; Constitutional AI is a canonical example [21, 140]. A closely related variant is *direct RLAIF* (d-RLAIF), where an LLM directly evaluates policy outputs and provides rewards during RL, without training a separate reward model [140]. Thus, LLM-as-a-judge reward schemes can naturally be viewed as instances of d-RLAIF rather than a separate feedback class. Neither RLAIF nor d-RLAIF is strict RLVR because their Assessment signal is an AI-generated judgment rather than an externally verifiable correctness check.

Learned assessment models are nevertheless important for reasoning. Process-supervised reward models improve mathematical reasoning [155]; Math-Shepherd trains a process reward model and uses step-level PPO to improve GSM8K and MATH performance [235]; there are further similar examples [76, 103, 138]. These examples motivate retaining reward-model terms in our complexity analysis. When Assessment is implemented by a learned outcome or process reward model, it requires LLM forward passes; when process-level feedback is used, the evaluator may be invoked at many reasoning steps. Such learned-assessment pipelines can therefore be more compute-heavy than lightweight exact-match RLVR, even though both instantiate the same Assessment role.

A third class consists of **offline preference-optimization methods**. DPO is not RL in its basic form: it inherits the preference-learning objective of RLHF but optimizes static preference pairs without online rollout generation or explicit reward-model inference [200]. Yet, DPO-style methods have also been adapted to reasoning, for example through iterative reasoning preference optimization and step-wise DPO [134, 238, 242, 252]. We include them because they share parts of the same post-training staged dataflow with policy/reference forward passes and policy backpropagation.

Overall, all these methods build upon all or part of the same systems abstraction: *Generation* $\rightarrow$ *Assessment* $\rightarrow$ *Training*. Online RL methods such as RLHF, RLAIF/d-RLAIF, and RLVR execute the full loop: the policy samples trajectories, Assessment maps them to rewards, model judgments, verifier outcomes, advantages, or other learning signals, and Training updates the policy and potentially critic. Offline preference methods such as DPO remove online Generation and explicit reward Assessment from the inner loop, but retain the policy/reference comparison and Training components. Our analysis therefore targets the broad class of RL-style reasoning post-training systems, where the Assessment stage may be instantiated by a learned reward model, an LLM evaluator, an automatic verifier, or an offline preference signal.

Models & training signals used in the RL pipeline	
π_θ	Policy (Actor) model: the trainable LLM being optimized.
π_{ref}	Reference model: the frozen baseline policy (usually a checkpoint right after SFT).
R_φ	Reward model: an assessment tool that maps prompt–response pairs to scalar rewards.
V_ψ	Critic model (Value function): an assessment tool that estimates expected future returns from the reward model.
$r_b^{(i)}, A_{b,t}^{(i)}, \hat{G}_{b,t}^{(i)}$	Reward for the i-th candidate of prompt b, advantage at token step t, and discounted return estimated at token step t.
Data & Dimensions	
$\mathcal{X}, \mathcal{Y}$	Space of input prompts and token sequences, respectively
$X = \{x_1, \ldots, x_B\}$	Batch of B input prompts; $X \subset \mathcal{X}$.
$y = (y_1, \ldots, y_T)$	A response sequence consisting of T tokens ($y \in \mathcal{Y}$).
K	Number of candidate responses generated per prompt (in the Generation stage).
S, T	The average prompt length and the average response length.
LLM & system design	
N, L, V, B	Number of parallel devices, number of layers, vocabulary size, batch size, respectively.
$d, d_{\text{ff}}, h, d_k = d/h$	Hidden dimension, the intermediate FFN dimension, number of parallel heads, and per-head dimension, respectively.
E, E_a, d_e	The number of experts per layer, the number of active experts per token, and the expert hidden dimension, respectively.

TABLE 1: **Overview of basic mathematical notation and concepts used in the paper.** Notation specific to the complexity analyses is detailed separately in Table 3.

2.3 Auxiliary Models in the RL-LLM Pipeline

We now investigate the functional and execution characteristics of models used in the RL pipeline, see Table 2 and Figure 2 (top). While conceptually distinct, in practical settings, they share similar LLM-based architectures.

Model	State	Used in	System design remarks
Policy (Actor) π_θ	Trainable	Generation, Training	**Bottleneck:** Sequential decoding dominates wall-clock time.
Reward R_φ (Outcome)	Frozen	Assessment	Efficient parallel evaluation; no backward pass. Easy to offload.
Reward R_φ^{pr} (Process)	Frozen	Assessment	High overhead; sequential decoding if using sequential verification (e.g., MCTS).
Critic (Value) V_ψ	Trainable	Assessment, Training	Doubles training compute (fwd+bwd). High memory pressure (optimizer states).
Reference π_{ref}	Frozen	Assessment	Required for KL/DPO. Forward-only; often quantized to save memory.

TABLE 2: **Overview of models used in the RLM pipeline.**

2.3.1 Reward Model

The reward model is a function $R_\varphi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that maps a prompt–response pair (x, y) to a scalar reward r, i.e., $R_\varphi(x, y) = r$, representing overall response quality, helpfulness, or alignment with desired behavior. These rewards are typically learned from human preference data and serve as the principal alignment signal for the policy. During the Assessment stage, the reward model performs a *single forward pass per full sequence* to produce $r_b^{(i)}$. It is a frozen model (no gradients computed). Moreover, as scalar rewards are usually single-token, its execution is usually non-autoregressive, which makes its evaluation considerably cheaper than autoregressive policy generation.

There are two types of reward models: **outcome-based** and **process-based** reward models (R_φ^{proc}). The former is a standard mechanism that assigns a single scalar score r to a *completed sequence*. Process-based reward models provide feedback at *intermediate steps* of reasoning. Formally, it maps a sequence to a vector of rewards. While this provides a richer supervision signal, it significantly increases computational cost (as it may require multiple forward passes or Monte Carlo Tree Search (MCTS) integration) and is difficult to train [26, 107].

2.3.2 Critic Model

The critic model is a function defined as $V_\psi : \mathcal{X} \times \mathcal{Y}_{\leq t} \to \mathbb{R}$. It estimates the expected future reward from a partial sequence, i.e., $V_\psi(x, y_{<t}) = v_t$. Given a prompt x and prefix $y_{<t}$, the critic predicts v_t as an estimate of the return that the policy can expect from that point onward, i.e., $V_\psi(x, y_{<t}) \approx \mathbb{E}_{\pi_\theta}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau \mid x, y_{<t}\right]$. When r comes from a learned reward model, we have $r_T = R_\varphi(x, y_{1:T})$ and $r_t = 0$ for $t < T, \gamma = 1$. Unlike the reward model, the critic provides *token-level* information, producing value estimates for all positions t in a single forward pass over the full sequence. During the Assessment stage, these predictions are combined with observed rewards to compute advantage estimates A_t. These advantage estimates are then used in Training to guide gradient updates of the policy model. The critic runs a full forward pass per sampled sequence and a backward pass for its parameter update, often in parallel with the policy gradient computation.

Unlike reward models, the critic is *trainable*. The reward model R_φ acts as a fixed proxy for task quality; keeping it fixed during policy optimization provides a stationary reward objective. The critic V_ψ, however, estimates the expected return under the *current* policy. As π_θ changes, the distribution of rollouts and their expected returns changes as well. Therefore, PPO-style methods update V_ψ after each rollout batch by minimizing a value-regression loss $\frac{1}{|\mathcal{B}|}\sum_{(x,y)\in\mathcal{B}}\sum_t \left(V_\psi(x, y_{<t}) - \hat{G}_t\right)^2$, where $\hat{G}_t$ is a Monte-Carlo or GAE-style return target computed from the rewards collected on that batch. In this way, the critic tracks the return function induced by the evolving policy.

2.3.3 Reference Model

The reference model π_{ref} is a frozen policy $\pi_{\text{ref}} : \mathcal{X} \times \mathcal{Y} \to [0, 1]$ that serves as a fixed baseline to stabilize optimization. It is typically a snapshot of the policy after SFT and before

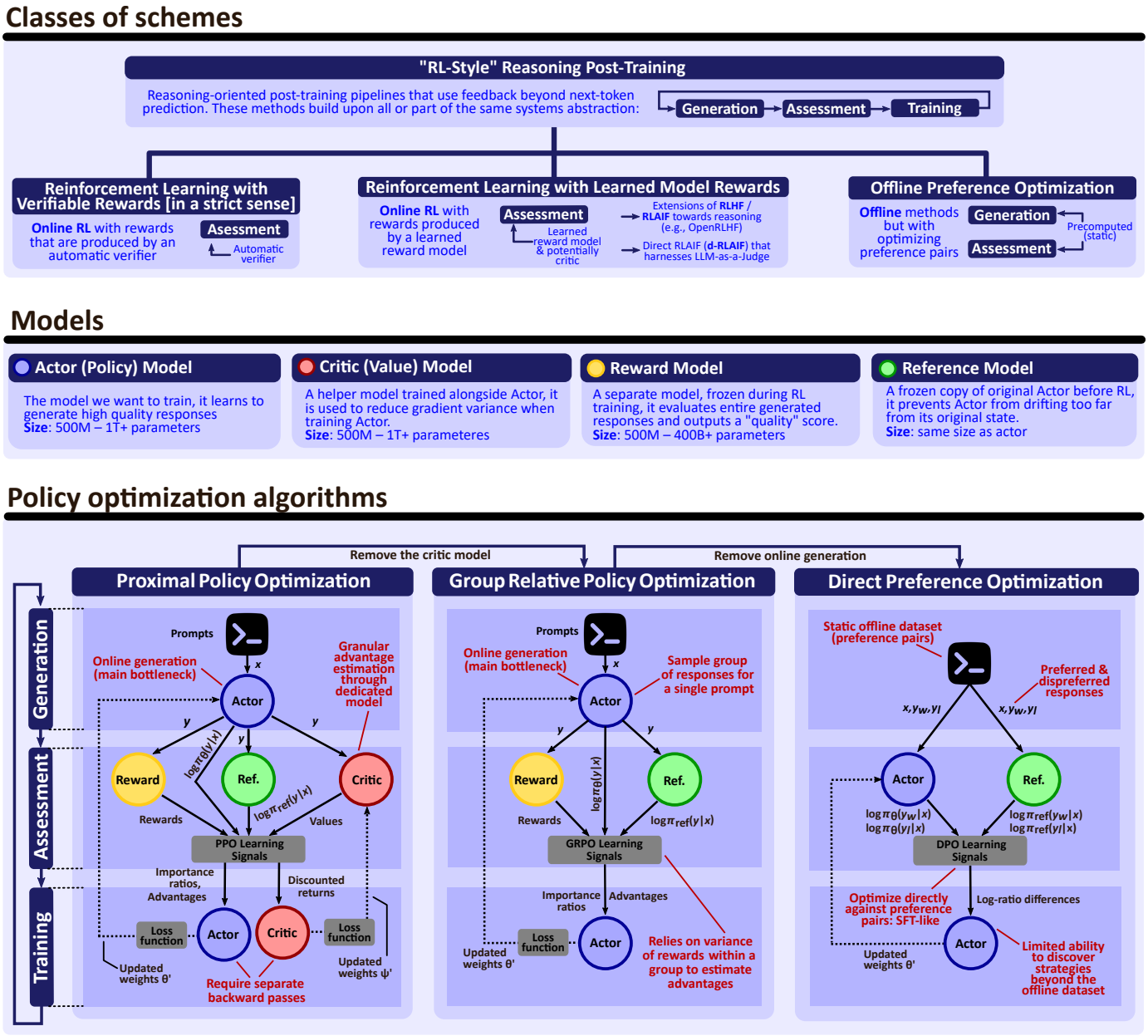


Fig. 2: **Overview of policy optimization algorithms and used models.** *No AI was used to conceive or to draw the figure.*

any RL. During the Assessment stage, it provides per-token log-probabilities $\log \pi_{\text{ref}}(y_t \mid x, y_{<t})$ used to measure deviation from the current policy. During the Training stage, these outputs are reused for regularization, e.g., through a KL-divergence penalty $D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$ or preference deltas in DPO. Since it remains frozen, π_{ref} requires only forward passes, but it can still be expensive to evaluate at scale because it performs a full-sequence teacher-forced forward pass over every rollout.

2.4 Auxiliary Models vs. Algorithmic RL Frameworks

Depending on what auxiliary models are used, there are **PPO-like methods**, **GRPO-like methods**, and **DPO-like methods**, with distinct computational trade-offs. PPO [208] uses all four models, requiring both forward and backward passes for π_θ and V_ψ, and additional forward-only evaluation for R_φ and π_{ref}. GRPO [107, 156, 213] omits the critic, saving backward passes but still requiring reward inference. DPO [200] relies solely on two models (π_θ and π_{ref}), resulting in an offline teacher-forced forward/backward pipeline (without online Generation or explicit reward-model inference) that is computationally lightweight and easier to scale; however, it also takes as input the pre-computed preference dataset. Thus, the combination of these models not only defines the algorithmic behavior of the RL method but also determines its computational profile, parallelism potential, and system-level design.

2.4.1 *Variants and Emerging Frameworks*

There are numerous variants of PPO, GRPO, and DPO-like methods, we now summarize most important ones, focusing on their computational characteristics.

Within the PPO family, VAPO [268] retains the value model but improves value-based training efficiency and

stability through value pretraining and decoupled GAE, so its compute remains recognizably actor–critic rather than critic-free. RTO [280] shifts supervision toward token-level rewards and advantages, increasing per-token bookkeeping during training. Safe-RLHF [75] extends PPO-like RLHF with a separate cost model and a Lagrangian safety constraint, thereby adding another assessment model while making helpfulness–harmlessness trade-offs explicit. Other PPO-style trust-region variants include divergence-based DPPO [198], which replaces ratio clipping with direct divergence estimates while preserving the basic online actor–critic profile, and mirror-descent formulations such as MDPO [228], which largely preserve the actor–critic compute structure while changing the policy update from PPO's clipped-ratio surrogate to a Bregman- or KL-regularized mirror-descent step around the previous policy.

Within the GRPO family, DAPO [265] improves sample and FLOP efficiency through asymmetric clipping, dynamic sampling, token-level losses, and better handling of over-long trajectories, while pruning-based extensions such as DPPO [286] reduce wasted decoding and training work by pruning unpromising trajectories with unbiased correction. Other GRPO-like extensions such as Graph-GRPO [60], RiskPO [203], GBMPO [266], MicroCoder-GRPO [154], and PIPO [233] mostly preserve the same critic-free online compute profile while changing the regularizer, feedback signal, or application setting. Related critic-free online methods such as RLOO [6] and ReMax [153] also avoid value-model training, but replace learned critics with leave-one-out, sample-based, or greedy-response baselines rather than GRPO's group-relative normalization.

DPO-style variants typically preserve the offline, SFT-like compute profile in which the policy is updated by batched teacher-forced passes over preference data. IPO [96] changes the pairwise objective without materially changing this profile; ORPO [114] combines the SFT term with an odds-ratio preference penalty and removes the explicit reference model; SimPO [170] also removes the reference model and thus reduces memory and forward cost; KTO [87] uses binary desirability signals instead of paired preferences; and AlphaDPO [246] introduces adaptive margins while keeping the basic offline scaling behavior. Reward-aware preference objectives [223] partially reintroduce reward information into this family: they add quality-aware signals to the preference loss without returning to the full online actor–critic loop.

Finally, **emerging likelihood-oriented methods** such as MaxRL [224] target reasoning-heavy settings with compute-indexed sampling objectives that interpolate between standard RL and maximum-likelihood optimization as additional sampling compute is allocated. Overall, these variants do not change the main systems distinction: online methods are dominated by rollout generation, critic-free methods remove V_ψ-dependent costs, and offline preference methods trade exploration for cheaper batched training.

2.5 Computational Analysis of RL-LLM

We now analyze computational complexities of RL-LLMs. Mathematical derivations are detailed in Appendix B.

We first derive the computational costs of **building blocks**, see Table 3. Namely, each harnessed model comes with forward and backward pass costs (work, depth, and memory consumption). These costs are similar or identical (in the asymptotic sense) across different models, because all models are based on the transformer architecture.

Next, we obtain the **complexities for PPO, GRPO, and DPO**, see Table 4. Overall, the work–depth expressions make the ordering precise. Among the online methods, PPO is the most expensive because its per-iteration work contains all major terms, and its training memory also includes both trainable models. GRPO removes the critic V_ψ, so it eliminates the additional $\Theta(BK(S+T)[C_f^{\text{tok}}(V_\psi)+C_b^{\text{tok}}(V_\psi)])$ work and the corresponding $\Theta(|V_\psi|)$ model-state memory, but it retains the same dominant online bottleneck as PPO, namely the autoregressive generation depth $D_{\text{gen}} = \Theta(T \cdot D_f(\pi_\theta))$. By contrast, DPO removes online rollout generation and explicit reward/critic evaluation from the inner loop, leaving only batched forward/backward passes over preference pairs. Thus, unlike PPO and GRPO, DPO has no T-step autoregressive term in its inner-loop depth, which explains why its systems profile is much closer to standard SFT. The trade-off is algorithmic: because DPO optimizes over a fixed preference dataset rather than fresh online rollouts, it cannot directly explore behavior outside that dataset.

Later in Section 5, we also present detailed algorithmic **parallelism-focused specifications** (Algorithms 1, 2, 3, and 4) that explicitly annotate most relevant parallel execution.

2.6 Key Insights & Takeaways

We summarize key takeaways.

▲ **Autoregressive generation imposes an irreducible per-rollout dependency.** For PPO and GRPO, each rollout is generated token by token, yielding a per-trajectory depth of $O(TD_f(\pi_\theta; S+T))$. Parallelizing Assessment or Training cannot remove this sequential dependency within an individual trajectory. Consequently, efficient decoding kernels remain central to reducing rollout latency and increasing generation throughput.

▲ **Generation is not necessarily the end-to-end system bottleneck.** The sequential depth of an individual rollout does not imply that the entire Generation stage must dominate iteration time. Through inter-stage fusion, completed samples or rollout subbatches can be streamed into Assessment while longer trajectories or later rollout waves are still being generated. This overlap is useful when completion lengths are skewed or when finite generation capacity forces the rollout batch to be processed in multiple waves. It trades additional memory (e.g., through live KV-cache and communication buffers) for reduced pipeline idle time.

▲ **Auxiliary models define the systems profile.** PPO uses $\{\pi_{\text{ref}}, R_\varphi, V_\psi\}$, GRPO removes V_ψ, and DPO removes both R_φ and V_ψ from the inner loop. Hence the PPO–GRPO gap is precisely the critic cost $\Theta\Big(BK(S+T)[C_f^{\text{tok}}(V_\psi)+C_b^{\text{tok}}(V_\psi)]\Big)$ plus critic model-state memory. The GRPO–DPO gap is more structural: DPO also removes online rollout generation from the optimization loop, eliminating the T-step generation depth term.

▲ **Frozen models are placement opportunities.** The reward and reference models are forward-only. They require no

Cat.	Symbol	Meaning and remarks	Mathematical expression
Work	$C_f^{\text{tok}}(M)$	**Forward-pass FLOPs per token for model M on a context of length $S+T$ (input prompt length + response length).** The reference policy π_{ref} is architecturally identical to the actor π_θ. Unembedding FLOPs per token are $2Vd_\pi$ (policy only). The reward model R_φ adds a projection head applied to the final token only, contributing $\frac{2d_R}{S+T}$ FLOPs per token, whereas the value model V_ψ applies a projection head to every token, contributing $2d_V$ FLOPs per token.	$C_f^{\text{tok}}(\pi_\theta) = 2L_\pi\left[4d_\pi^2 + 2d_\pi d_{\pi\text{ff}} + (S+T)d_\pi\right] + 2Vd_\pi$ $C_f^{\text{tok}}(\pi_{\text{ref}}) = C_f^{\text{tok}}(\pi_\theta)$ $C_f^{\text{tok}}(R_\varphi) = 2L_R\left[4d_R^2 + 2d_R d_{R\text{ff}} + (S+T)d_R\right] + \frac{2d_R}{S+T}$ $C_f^{\text{tok}}(V_\psi) = 2L_V\left[4d_V^2 + 2d_V d_{V\text{ff}} + (S+T)d_V\right] + 2d_V$
	$C_b^{\text{tok}}(M)$	**Backward-pass FLOPs per token for model M on a context of length $S+T$ (input prompt length + response length).**	$C_b^{\text{tok}}(M) \approx 2 \cdot C_f^{\text{tok}}(M)$
	$C_{\text{gen}}^{\text{roll}}(\pi_\theta)$	**Autoregressive generation FLOPs per rollout** for generating T tokens (response length) from a context of length S (input prompt length) with KV caching (policy only). It is decomposed into prefill and decode. Prefill FLOPs are for processing the prompt of length S once and initializing the KV cache before autoregressive decoding begins. Prefill attention is assumed to be dense; as it is causal only half of the full MM product are needed, giving $\approx S^2 d_\pi$ FLOPs for both QK^T and for PV; $O(S^2)$ and $O(Sd)$ account for lower-order contributions (respectively – softmax and scaling by $1/\sqrt{d_h}$ as well as RoPE and layer norms). Decode FLOPs are for generating the remaining $T-1$ tokens autoregressively with KV caching. At decode step t, the model attends to context length $S+t$.	$C_{\text{gen}}^{\text{roll}}(\pi_\theta) = C_{\text{pf}}^{\text{roll}}(\pi_\theta) + C_{\text{dec}}^{\text{roll}}(\pi_\theta)$ $C_{\text{pf}}^{\text{roll}}(\pi_\theta) = 2L_\pi S\left(4d_\pi^2 + 2d_\pi d_{\pi\text{ff}} + Sd_\pi\right) + L_\pi\left(O(S^2) + O(Sd_\pi)\right) + 2Vd_\pi$ $C_{\text{dec}}^{\text{roll}}(\pi_\theta) = \sum_{t=1}^{T-1}\left[2L_\pi\left(4d_\pi^2 + 2d_\pi d_{\pi\text{ff}} + 2(S+t)d_\pi\right) + 2Vd_\pi\right] + L_\pi\left(O(S+t) + O(d_\pi)\right)$
Depth	$D_f(M)$	**Forward-pass depth** of model M on a context of length $S+T$, defined as the length of the critical path assuming unbounded parallelism. The policy π_θ includes an additional unembedding (output projection) step, which contributes a $\log d_\pi$ term to the depth. For the reward model R_φ and the critic V_ψ, the additional projection head contributes an extra $\log d$ term to the depth.	$D_f(\pi_\theta) = O\left(L_\pi\left[\log d_\pi + \log(S+T)\right] + \log d_\pi\right)$ $D_f(\pi_{\text{ref}}) = D_f(\pi_\theta)$ $D_f(R_\varphi) = O\left(L_R\left[\log d_R + \log(S+T)\right] + \log d_R\right)$ $D_f(V_\psi) = O\left(L_V\left[\log d_V + \log(S+T)\right] + \log d_V\right)$
	$D_f^{\text{TP}}(M)$	**Forward-pass depth** of model M **with tensor parallelism** degree P_t on a context of length $S+T$.	$D_f^{\text{TP}}(M) = O\left(L_M\left[\log\frac{d_M}{P_t} + \log(S+T)\right] + \log\frac{d_M}{P_t}\right)$
	$D_b(M)$	**Backward-pass depth** of model M on a context of length $S+T$. It equals the depth of the forward pass for the corresponding model M (which entails computing the gradients with respect to activations), plus the additional term $O(\log(B(S+T)))$ coming from computing the gradients with respect to model weights, where one has to accumulate gradients across batch (as these two branches can overlap, the expression given is a conservative bound).	$D_b(M) = D_f(M) + O(\log(B(S+T)))$
	$D_{\text{gen}}(\pi_\theta)$	**Autoregressive generation depth** for generating T tokens given a prompt of length S. Prefill contributes one forward-pass depth (over context of length S); decode contributes one sequential forward-pass depth per generated token (i.e., over context of length $S+t$ for the t-th token).	$D_{\text{gen}}(\pi_\theta) = D_f(\pi_\theta; S) + \sum_{t=1}^{T-1} D_f(\pi_\theta; S+t)$ $= O\big(T \cdot D_f(\pi_\theta; S+T)\big)$
Memory	$\lvert M\rvert$	**Model parameter count** of M (weights and embeddings). It always includes the parameters from query, key, value and output projections ($4d^2$), two FFN projections ($2dd_{ff}$), and the final logit computation (Vd). For reward and critic models, there are also d parameters in the final head that outputs the score(s).	$\lvert\pi_\theta\rvert = L_\pi(4d_\pi^2 + 2d_\pi d_{\pi\text{ff}}) + Vd_\pi$, $\lvert\pi_{\text{ref}}\rvert = \lvert\pi_\theta\rvert$ $\lvert\pi_{\text{ac}}\rvert = L_{\text{s}}(4d_{\text{ac}}^2 + 2d_{\text{ac}}d_{\text{ac,ff}}) + Vd_{\text{ac}} + d_{\text{ac}}$ $\lvert R_\varphi\rvert = L_R(4d_R^2 + 2d_R d_{R\text{ff}}) + Vd_R + d_R$ $\lvert V_\psi\rvert = L_V(4d_V^2 + 2d_V d_{V\text{ff}}) + Vd_V + d_V$
	M_{Act}	**Training activation memory** per processed token for model M. Under the leading-order activation model used throughout the paper, activations required for backpropagation scale with the number of layers and hidden width. Constant factors from Q/K/V tensors, FFN intermediates, normalization, residual, and other temporary tensors are suppressed. This assumes no activation checkpointing.	$M_{\text{Act}} = \Theta(L_M d_M)$
	M_{Inf}	**Forward-only inference-buffer memory** per processed token for model M. Because intermediate layer buffers can be reused across layers and the attention matrix is assumed not to be fully materialized, the leading-order buffer scales with hidden width rather than layer count.	$M_{\text{Inf}}(M) = \Theta(d_M)$
	M_{KV}	**KV-cache memory** required for one rollout of prompt length S and response length T during generation. If all BK rollouts are generated concurrently, the peak KV-cache memory is $BK \cdot M_{\text{KV}}$; if rollouts are generated sequentially across the K samples per prompt, the peak reduces accordingly but the generation depth increases.	$M_{\text{KV}} = 2(S+T)L_\pi d_\pi$

TABLE 3: **Computational building blocks used across the paper.** We assume canonical multi-head attention and two-projection FFN. For computing FLOP costs, an addition and a multiplication count as two separate FLOPs (i.e., a dot product of vectors of dimensionalities d results in $d + (d-1) \approx 2d$ FLOPs under this model). C_f^{tok}, C_b^{tok}, $C_{\text{gen}}^{\text{roll}}$ are the costs of (respectively) forward pass, backward pass, and autoregressive generation; defining them per token and per rollout effectively clarifies the notation for the subsequent analyses. The explicit depth expressions count the selected GEMM reduction chains and parameter-gradient accumulation; they suppress the additional logarithmic reductions from normalization, attention/vocabulary softmax, sampling, and distributed collectives. For generation, we explicitly distinguish prefill and decode. Prefill processes the prompt once, while decode generates tokens sequentially with KV caching. This decomposition is important because rollout generation is the dominant online bottleneck and is not equivalent to a single teacher-forced forward pass over a sequence of length $S+T$.

Framework	PPO (Online)	GRPO (Online)	DPO (Offline)
Work (Total FLOPs)			
Generation	$O(BK \cdot C^{\text{roll}}_{\text{gen}}(\pi_\theta))$	$O(BK \cdot C^{\text{roll}}_{\text{gen}}(\pi_\theta))$	– †
Assessment	$O(BK(S+T) \cdot [C^{\text{tok}}_f(\pi_{\text{ref}}) + C^{\text{tok}}_f(R_\varphi) + C^{\text{tok}}_f(V_\psi)])$	$O(BK(S+T) \cdot [C^{\text{tok}}_f(\pi_{\text{ref}}) + C^{\text{tok}}_f(R_\varphi)])$	$O(B_{\text{pairs}}(S+T) \cdot [C^{\text{tok}}_f(\pi_{\text{ref}}) + C^{\text{tok}}_f(\pi_\theta)])$
Training	$O(BK(S+T) \cdot [C^{\text{tok}}_{\text{tr}}(\pi_\theta) + C^{\text{tok}}_{\text{tr}}(V_\psi)])$	$O(BK(S+T) \cdot C^{\text{tok}}_{\text{tr}}(\pi_\theta))$	$O(B_{\text{pairs}}(S+T) \cdot C^{\text{tok}}_{\text{tr}}(\pi_\theta))$
Depth (Critical Path Latency)			
Generation	$O(D_{\text{gen}}(\pi_\theta))$	$O(D_{\text{gen}}(\pi_\theta))$	–
Assessment	$O(\max\{D_f(\pi_{\text{ref}}),\ D_f(R_\varphi),\ D_f(V_\psi)\})$ ‡	$O(\max\{D_f(\pi_{\text{ref}}),\ D_f(R_\varphi)\})$ ‡	$O(\max\{D_f(\pi_\theta),\ D_f(\pi_{\text{ref}})\})$
Training	$O(\max\{D_{\text{tr}}(\pi_\theta),\ D_{\text{tr}}(V_\psi)\})$	$O(D_{\text{tr}}(\pi_\theta))$	$O(D_{\text{tr}}(\pi_\theta))$
Memory Cost			
Generation	$O(\lvert\pi_\theta\rvert + BK \cdot M_{\text{KV}})$	$O(\lvert\pi_\theta\rvert + BK \cdot M_{\text{KV}})$	–
Assessment	$O(\lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + \lvert V_\psi\rvert + BK(S+T)M_{\text{Inf}})$	$O(\lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + BK(S+T)M_{\text{Inf}})$	$O(\lvert\pi_\theta\rvert + \lvert\pi_{\text{ref}}\rvert + B_{\text{pairs}}(S+T)M_{\text{Act}})$
Training	$O(\lvert\pi_\theta\rvert + \lvert V_\psi\rvert + BK(S+T) \cdot M_{\text{Act}})$	$O(\lvert\pi_\theta\rvert + BK(S+T) \cdot M_{\text{Act}})$	$O(\lvert\pi_\theta\rvert + B_{\text{pairs}}(S+T) \cdot M_{\text{Act}})$

TABLE 4: **Asymptotic Work, Depth, and Memory analysis of PPO, GRPO, and DPO.** S is prompt length, T is response length. We denote parameter counts by $|\cdot|$. **"tr" subscript:** For conciseness, for training-stage model invocations, we define $C^{\text{tok}}_{\text{tr}}(M) := C^{\text{tok}}_f(M) + C^{\text{tok}}_b(M) \approx 3C^{\text{tok}}_f(M)$ and $D_{\text{tr}}(M) := D_f(M) + D_b(M)$, because a parameter update requires a forward pass followed by backpropagation. **Note on Work:** C^{tok}_f and C^{tok}_b are the cost of forward and backward passes per token and $C^{\text{roll}}_{\text{gen}}$ is the autoregressive generation cost per rollout; these are derived in Table 3. Work scales with total tokens $(S+T)$. **Note on Depth:** Generation is depth-bound by T (sequential), while in Assessment/Training there is no outer linear T-step autoregressive chain. † DPO is offline; generation occurs prior to the training loop. ‡ Max depth assumes parallel (disaggregated) execution; sequential co-located execution is additive. M_{KV} is Key-Value cache memory; M_{Act} is activation memory (training); M_{Inf} is inference buffer (assessment). Note that π_{ref} never requires M_{Act}.

gradients or optimizer state, so they can be replicated, quantized, offloaded, or served as independent inference services. By contrast, trainable models π_θ and V_ψ dominate memory through activations, gradients, and optimizer state.

▲ **Reward granularity is a compute–credit-assignment trade-off.** Outcome rewards are cheap: one scalar per completion. Process rewards can improve reasoning supervision but may require step-level annotations, verifier calls, or search, turning Assessment from one batched forward pass into a much heavier verification workload.

▲ **DPO is SFT-like, but not free.** DPO still performs $W_{\text{DPO}} = \Theta\Big(B_{\text{pairs}}(S+T)[C^{\text{tok}}_f(\pi_\theta) + C^{\text{tok}}_f(\pi_{\text{ref}}) + C^{\text{tok}}_b(\pi_\theta)]\Big)$, but these are teacher-forced passes over static preference pairs. Its systems advantage is therefore the removal of online generation and reward/critic inference; its algorithmic limitation is that it cannot directly explore beyond the support of the preference dataset.

3 Intra-Model Parallelism

Efficient training and inference for each individual model in the RL-LLM pipeline is essential. *Intra-model parallelism* denotes established techniques that allow a single model to be trained or served across multiple devices by partitioning its parameters, activations, or computation graph. The main families are: (i) data parallelism, (ii) tensor/operator parallelism, (iii) sequence and context parallelism, (iv) pipeline parallelism, (v) expert parallelism, and (vi) memory-centric optimizations such as optimizer sharding and activation checkpointing. These techniques are typically combined into hybrid schemes (e.g., ZeRO-style sharded data parallelism plus tensor parallelism) to balance memory footprint, communication cost, and compute utilization. As these techniques are well-known [120, 132, 141, 146, 160, 176, 219], we summarize them and instead focus on implications for RL–LLM pipelines.

We detail the computational aspects of the intra-model execution Figure 3 (mathematical details and flow diagrams), Figure 4 (forward pass), Figure 5 (backward pass), and in Figures 6-7 (details on intra-model parallelism and taxonomy).

3.1 Data Parallelism

Data parallelism (DP) replicates the model across N devices and shards the input batch across replicas. Each replica performs a local forward and backward pass on its shard, followed by a gradient synchronization step (typically an all-reduce [63, 227]) to keep parameters identical across devices. This strategy is conceptually simple and scales throughput almost linearly when the model fits into a single device and communication is not a bottleneck.

Naive DP quickly becomes memory-limited for very large models because each device must store a full copy of parameters, gradients, and optimizer state. Hence, DP is most effective when combined with memory-centric optimizations (e.g., ZeRO/FSDP [201]) so that parameters, gradients, and optimizer state are partially sharded rather than fully replicated. Moreover, the all-reduce step can become a major source of overhead as the number of devices grows or when interconnect bandwidth is limited. Here, modern systems apply several communication optimizations, such as overlap of computation and communication, gradient bucketing, and gradient accumulation [146].

3.1.1 Implications for RL-LLM Pipelines

In RL-LLM pipelines, a crucial point for DP is *whether a given model or stage requires gradients or is forward-only*.

Forward-only models. If a model is not updated in a given stage, DP reduces to independent batched inference with replicated weights and scales almost linearly in the number of devices until limited by input or network I/O. This is the typical regime for the actor during Generation (sampling rollouts), the actor during Assessment when

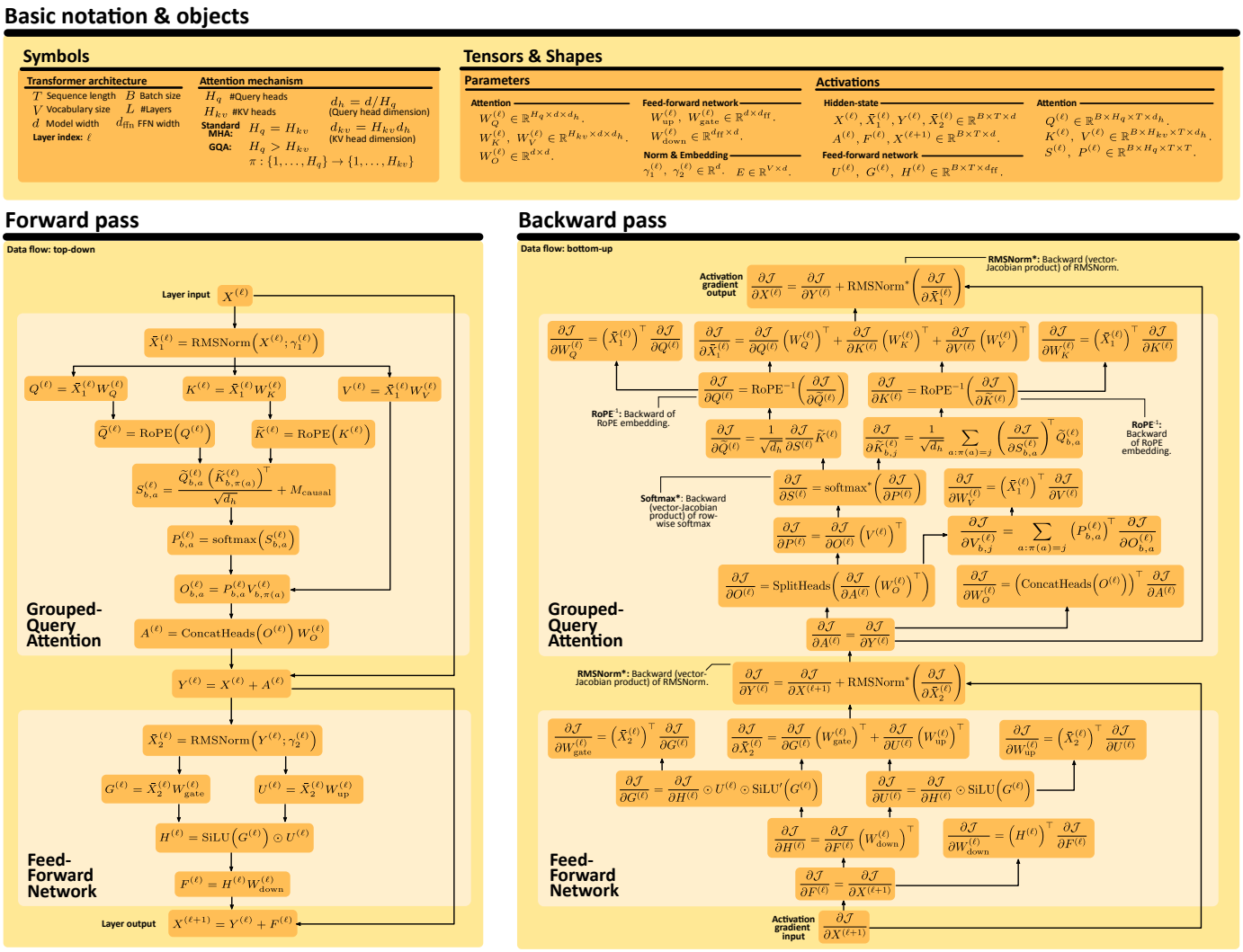


Fig. 3: **Mathematical notation and equations for the forward and backward passes of each decoder block.** Further details on computational aspects are in Figures 4-5 while parallelization details are in Figures 6-7. *No AI was used to conceive or to draw the figure.*

only forward log-probabilities and KL terms are needed, the reward model during Assessment, and the reference model during Assessment. OpenRLHF, for example, uses Ray and vLLM to run actor, reward, and reference as large-scale batched inference services, separate from training loops [117]. Here, DP is highly effective and typically the first choice.

Trainable models. Here, DP must all-reduce gradients across workers and thus benefits from sharded data-parallel variants such as ZeRO-2/3 or FSDP. This applies to the actor in the Training stage (PPO/GRPO/DPO updates) and to the critic in actor–critic methods. Frameworks such as OpenRLHF explicitly apply ZeRO-3/FSDP-style sharded data parallelism to actor and critic training while keeping forward-only models replicated [117, 201].

In RLM pipelines, a common pattern is therefore: DP + ZeRO for trainable components; pure DP for forward-only components.

3.2 Tensor Parallelism

Tensor (or operator) parallelism (TP) partitions individual layers across devices by sharding their parameters along hidden dimensions, so that each device computes only a slice of the layer. Synchronization via all-reduce or all-gather then reconciles partial results.

In Transformer blocks, TP is commonly applied to both feed-forward networks (FFNs) and to the multi-head attention (MHA). For FFN, the first linear projection (e.g., XA) is split column-wise across devices, and the second projection (e.g., YB) is split row-wise. Each device holds a shard of A or B and computes a partial result. For MHA, query/key/value projections are sharded column-wise, assigning subsets of heads to devices. The output projection is sharded row-wise, mirroring the FFN pattern. TP is typically combined with DP (and sometimes pipeline parallelism) to form 2D, 3D, or 5D parallelism configurations capable of training trillion parameter models [176, 219].

Megatron-LM and related systems implement TP using communication operators that behave differently in forward and backward passes, e.g., an operator f that is identity in the forward pass and all-reduce in the backward, and an operator g that is all-reduce in the forward and identity in the backward [219]. This enables communication-computation overlap and reduces idle time.

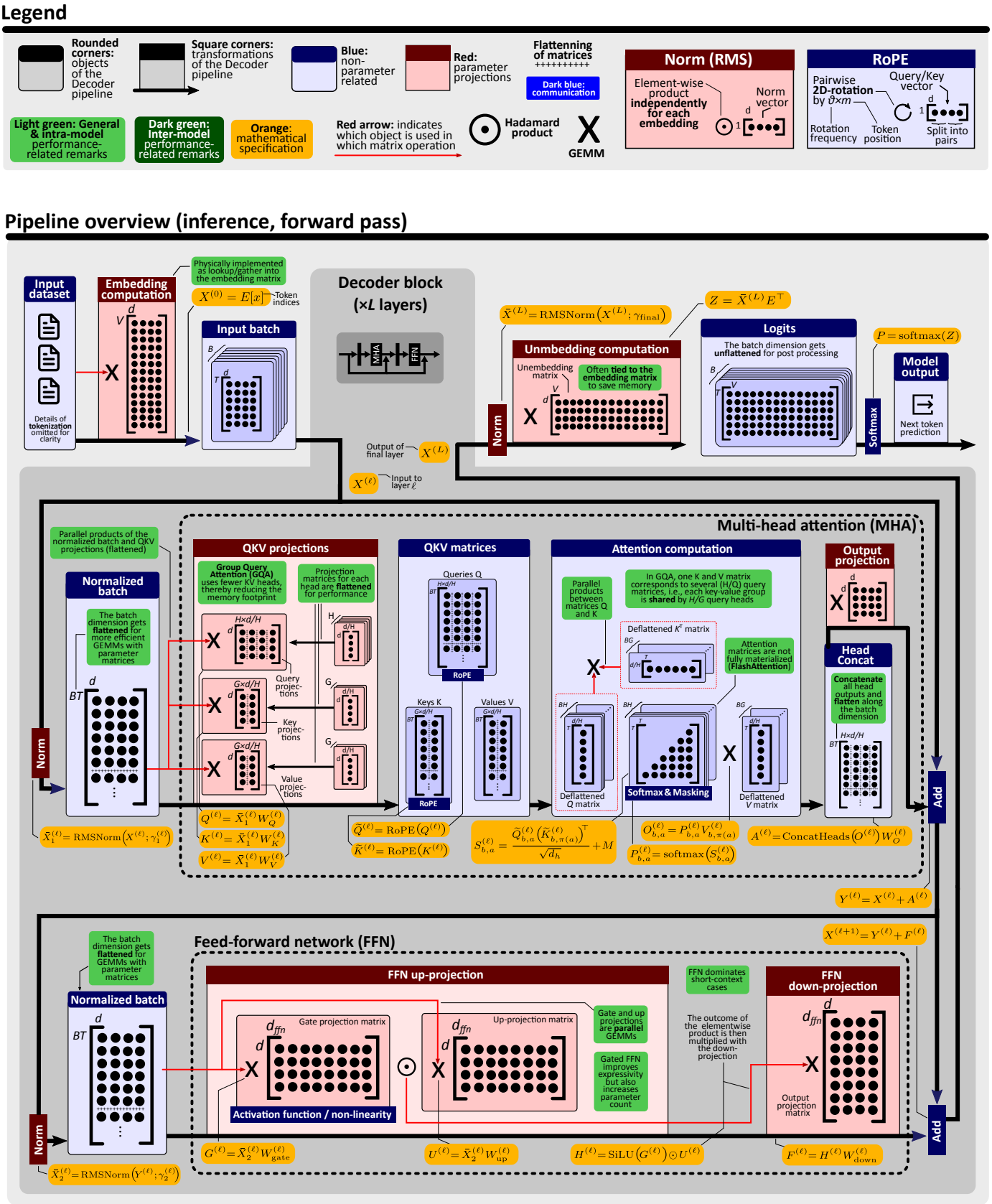


Fig. 4: **Intra-model execution details (forward pass) for each RLM model execution.** The design details are based on Llama-3. Further details on parallelization and the corresponding taxonomy are provided in Figures 6-7. *No AI was used to conceive or to draw the figure.*

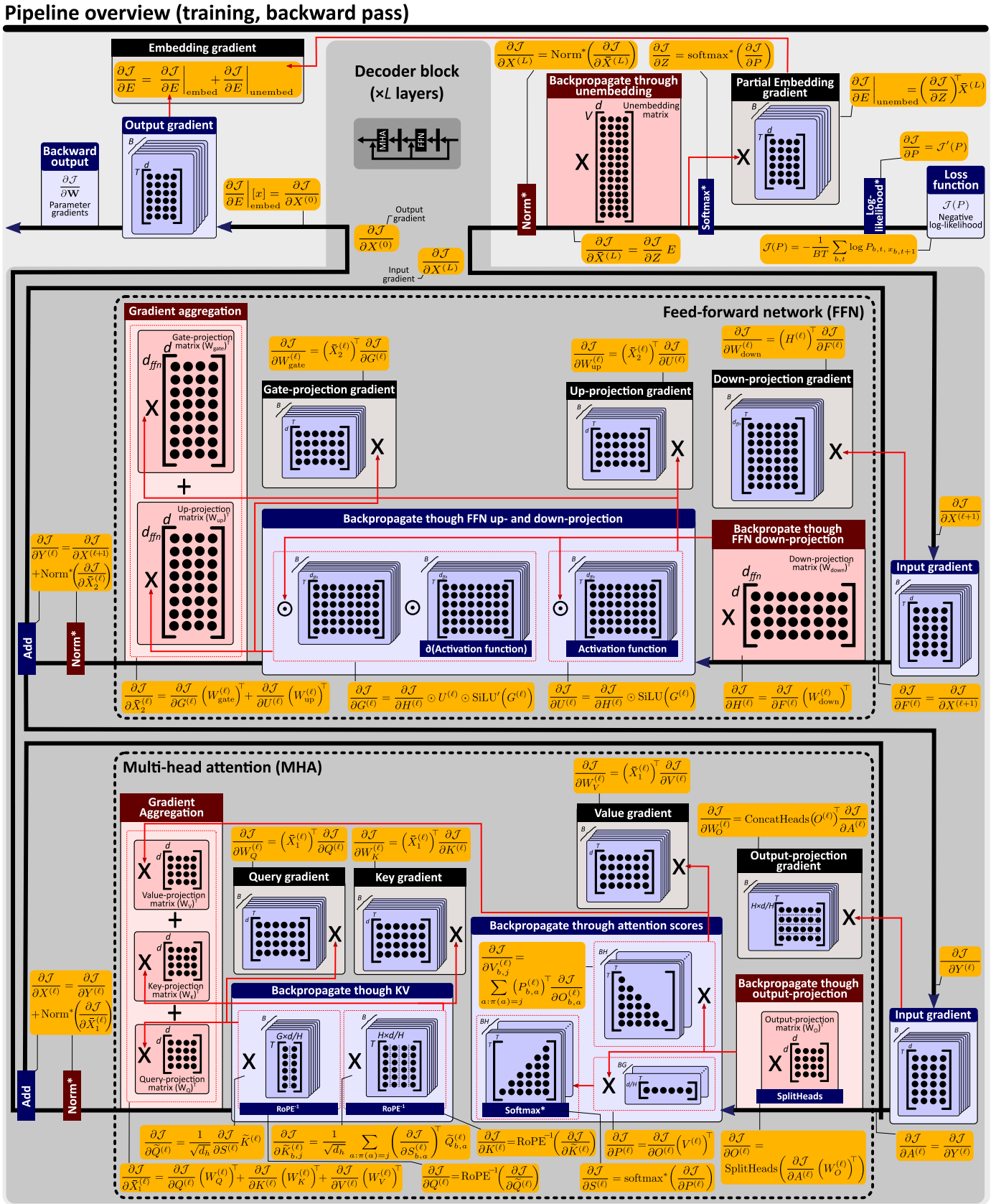
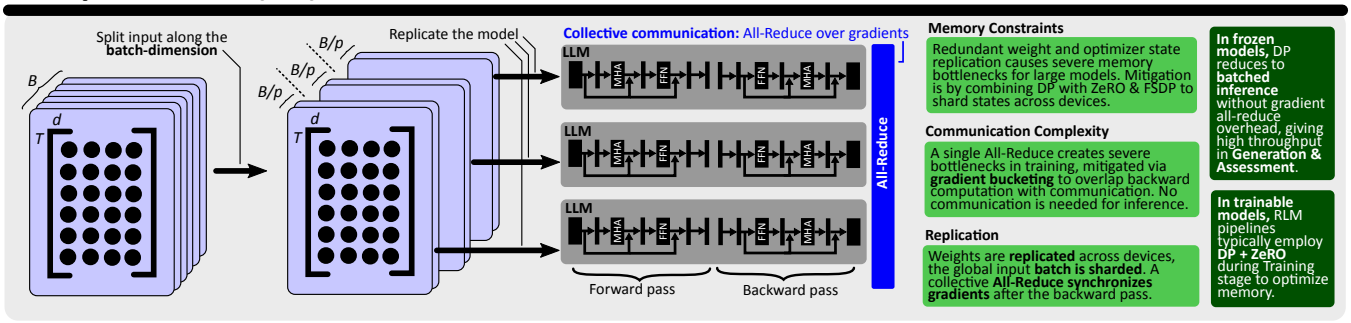
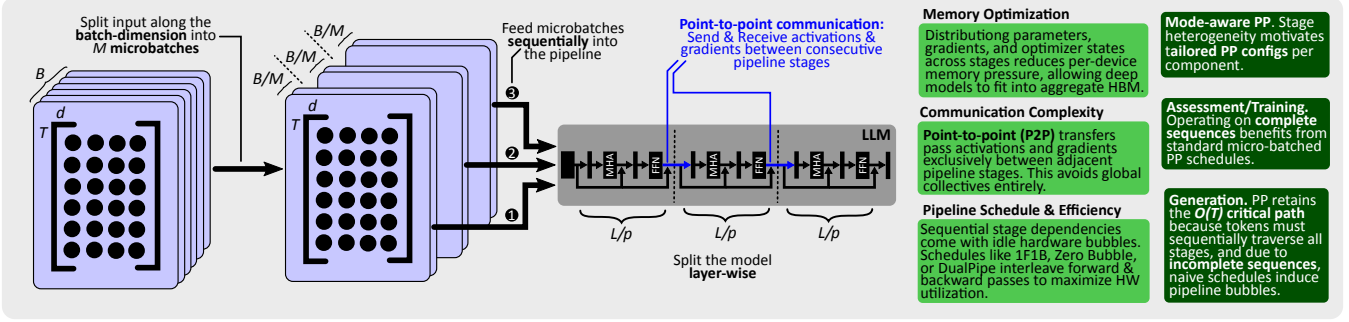


Fig. 5: **Intra-model execution details (backward pass) for each RLM model execution.** Legend is provided in Figure 4. The design details are based on Llama-3. Further details on parallelization and the corresponding taxonomy are provided in Figures 6-7. *No AI was used to conceive or to draw the figure.*

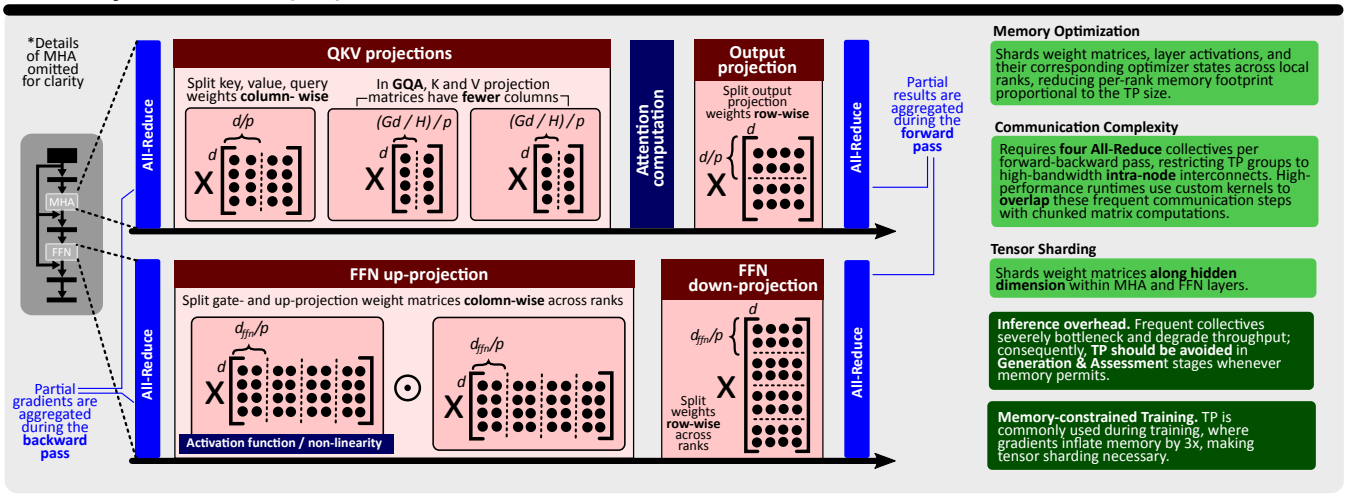
Data parallelism (DP)



Pipeline parallelism (PP)



Tensor parallelism (TP)



Expert parallelism (EP)

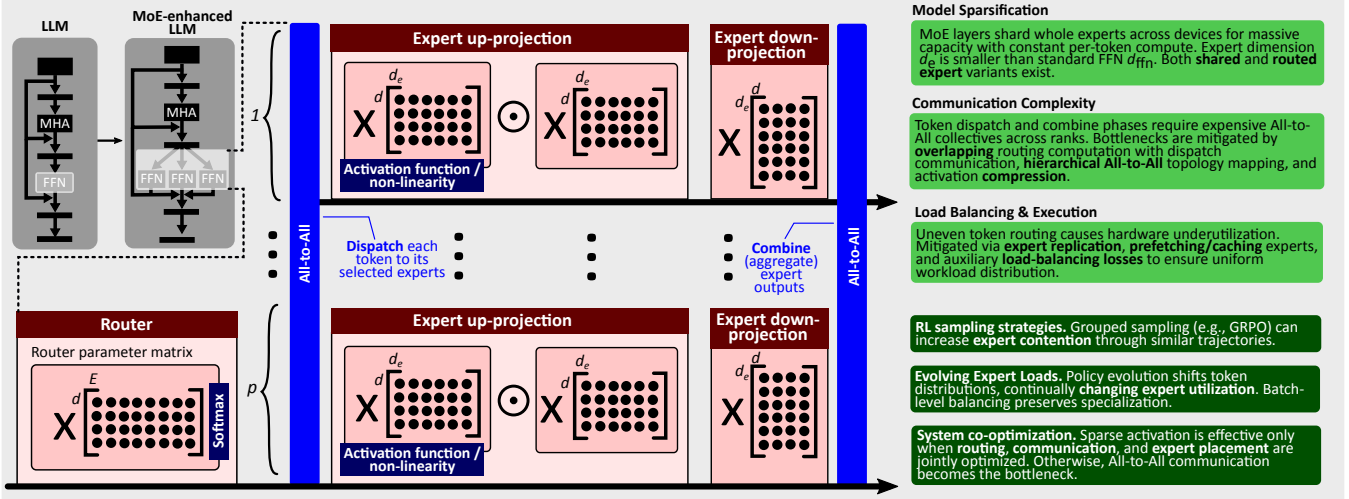


Fig. 6: **Intra-model parallelism (data, pipeline, tensor, and expert parallelism) details for each RLM model execution; legend is provided in Figure 4.** *No AI was used to conceive or to draw the figure.*

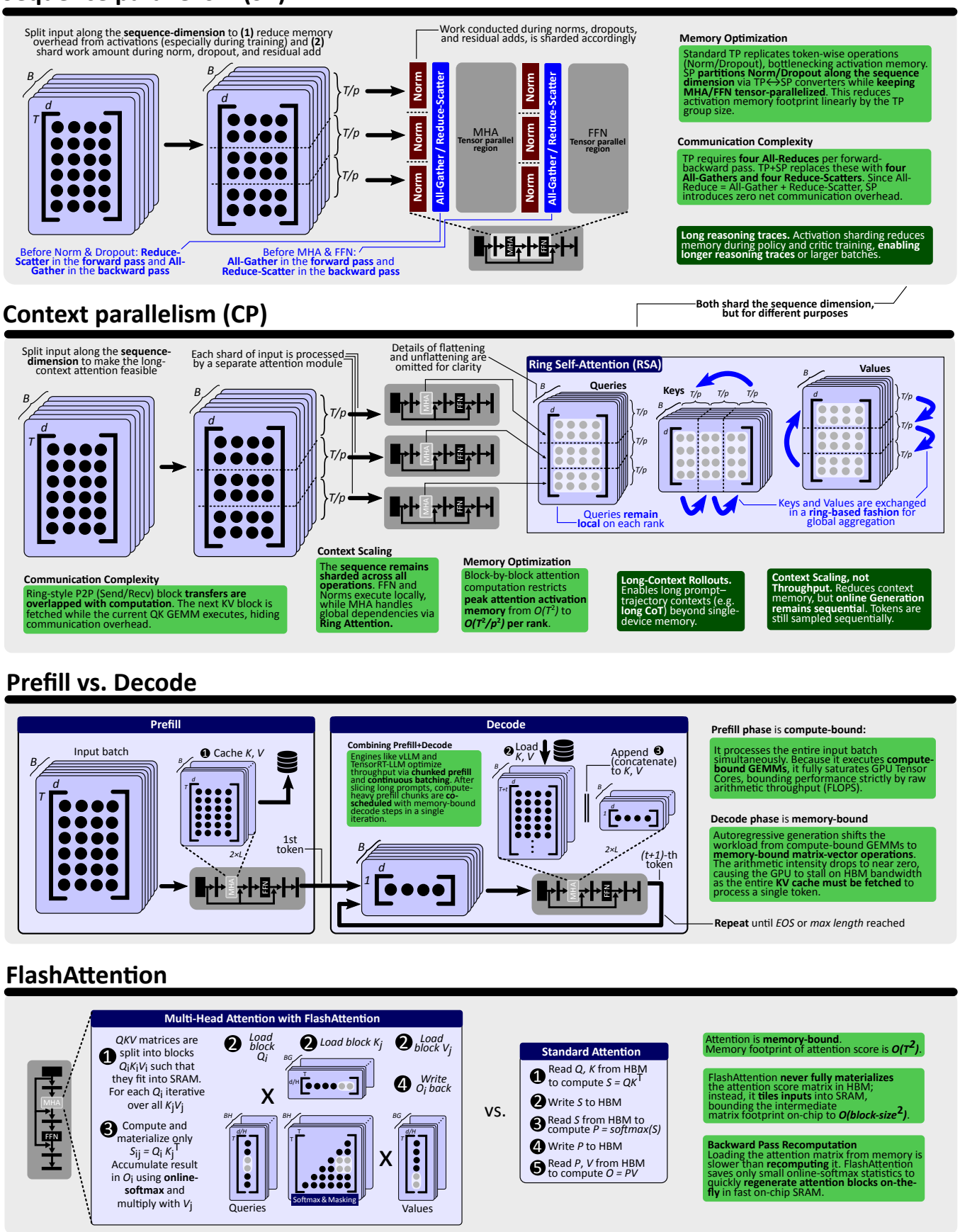


Fig. 7: **Intra-model parallelism (sequence & context parallelism, as well as prefill vs. decode and flash attention) details for each RLM model execution; legend is provided in Figure 4.** *No AI was used to conceive or to draw the figure.*

3.2.1 Implications for RL–LLM pipelines

Tensor parallelism is essential when a *single* model does not fit on one device even after applying memory optimizations (e.g., ZeRO, mixed precision, quantization). However, for many RLHF workloads, inference dominates runtime. For example, OpenRLHF reports that rollout generation and forward Assessment account for more than 90% of wall-clock time in RLHF loops [117]. Here, TP can hurt throughput when not strictly necessary. For instance, Megatron-style TP introduces two all-reduces per Transformer block during inference and four during training [176, 219]. When a model (actor, reward, or reference) fits on a single device, empirical studies from vLLM and DeepSpeed show that enabling TP reduces total tokens-per-second because collective communication becomes the bottleneck [201, 230]. AMD's Qwen2-7B RLHF experiments on MI300X (192 GB) similarly report that disabling TP gives 2.3–4.3× higher rollout throughput than when sharding with TP by 2 or 4 for the same model, as communication is negligible in the single-device setting [263].

The resulting rule-of-thumb for using TP in RLM pipelines is: (1) use TP only when a model does not fit on a single device even with ZeRO/FSDP and mixed precision; (2) prefer no TP for inference-heavy stages (Generation, forward-only Assessment) whenever memory allows; and (3) tolerate TP overhead more readily in training stages, where gradient storage already inflates memory by $\approx 3\times$ and TP may be necessary to fit the model [219]. In practice, large RLM deployments often use TP inside high-bandwidth device groups and combine it with pipeline and data parallelism across nodes [176].

3.3 Sequence Parallelism

TP shards heavyweight projections but leaves token-wise operations such as layer normalization and dropout replicated on each TP shard, leading to duplicated activations and higher memory use, especially for long sequences. Sequence parallelism (SP) [132] addresses this by partitioning activations along the sequence dimension. Because layer norm and dropout operate independently per token, their activations can be split across devices without changing the model's semantics. This reduces per-device activation memory at essentially no additional computation cost.

Sequence parallelism has been adopted in large-scale stacks such as Megatron-LM, DeepSpeed, and ColossalAI for long-context or high-batch training [125, 132, 147, 219], where activation memory is a primary limiter.

3.3.1 Implications for RL-LLM Pipelines

Sequence parallelism is primarily beneficial in *training* stages where activations must be stored or recomputed for every token, such as policy/critic updates over long responses and reasoning traces. Sharding activations by sequence length reduces per-GPU memory and allows larger T or batch sizes under fixed memory budgets. Because SP reuses the same collective bandwidth as TP (by switching all-reduce to reduce-scatter/all-gather pairs), its net communication volume stays similar, with modest changes in latency due to additional synchronization points [132, 219].

3.4 Context Parallelism

Context parallelism extends sequence-based partitioning further by maintaining the sequence split throughout the entire Transformer layer, including attention and FFN blocks [256]. Most operations remain token-wise and are naturally compatible; the main challenge is attention, which requires access to keys and values from all sequence partitions. Recent implementations address this using Ring Attention [160], which pipelines key/value communication in a ring topology to reduce latency and memory pressure. CP enables training and inference with very long contexts (hundreds of thousands or millions of tokens) by combining tensor, sequence, and context partitioning without replicating activations or weights unnecessarily [256].

3.4.1 Implications for RL-LLM Pipelines

Context parallelism is most useful when RLM training is dominated by long prompt–trajectory contexts (e.g., long CoT traces, tool histories, verifier outputs, etc.). In this regime, the bottleneck is often not only model size but also attention and activation memory over the full context. CP directly targets this axis by partitioning the sequence dimension across devices while preserving exact attention through cross-device KV exchange [125,160,183]. This yields concrete implications for RLMs. First, CP can make long-CoT rollouts feasible when the actor's context length exceeds single-device memory. Second, in verifier- or tool-augmented RL, CP allows prompts to include larger execution traces, retrieved evidence, and interaction histories without truncating the reasoning state.

However, CP does not remove the autoregressive dependence of online Generation: it reduces per-device context memory and attention work, but tokens are still sampled sequentially. Thus, CP is primarily a feasibility and long-context scaling mechanism; online rollout latency still requires complementary methods such as TP, efficient KV-cache exchange, continuous batching, or stage-level overlap. Recent long-context systems validate this direction: Ring Attention overlaps KV-block communication with blockwise attention, DeepSpeed-Ulysses uses sequence partitioning and all-to-all communication for long-sequence training, and CP-style inference systems report near-linear scaling for million-token prefill workloads [125, 160, 256].

3.5 Pipeline Parallelism

Pipeline parallelism (PP) partitions the layers of a model into *stages*, each placed on a different device. Mini-batches are further split into *micro-batches* that flow through the pipeline stages. This allows models that are too deep to fit on a single device to be trained, at the cost of pipeline *bubbles* (periods where some stages are idle). PP is often combined with DP and TP, yielding 3D/5D parallelism that supports extremely large models [176].

Numerous pipeline scheduling strategies have been proposed, examples include AFAB [120] (all-forward-all-backward, all micro-batches complete forward passes before any backward pass starts – simple but bubble-heavy), 1F1B [175] (forward and backward passes interleave, improving memory efficiency and reducing bubbles), Interleaved 1F1B [176] (devices host multiple non-contiguous

pipeline stages to further overlap computation and communication), Zero Bubble [197] (decomposes the backward pass into B-steps (gradients w.r.t. activations) and W-steps (gradients w.r.t. weights) scheduling them separately to eliminate bubbles without increasing peak memory), and DualPipe [156] (runs two pipelines in opposite directions across the same devices, overlapping forward and backward streams to improve utilization).

3.5.1 Implications for RL-LLM Pipelines

The relevance of PP to RL-LLM heavily depends on the considered stage. In Generation, the actor performs autoregressive decoding, so tokens must traverse all pipeline stages sequentially; thus, PP helps fit large actors but does not remove the $O(T)$ critical path, and naive schedules can suffer from large pipeline bubbles and poor utilization [251, 262]. By contrast, Assessment and Training operate on complete sequences and therefore benefit more directly from standard micro-batched PP schedules such as GPipe- or 1F1B-style execution. This heterogeneity makes *mode-aware* PP especially important in RLHF systems. The best sharding and scheduling strategy for low-latency Generation need not match the best one for high-throughput Training. Recent systems therefore decouple configurations across stages and overlap different RL iterations, e.g., by letting one batch generate while another is being assessed or trained [117, 261, 262]. Such inter-batch pipelining can substantially reduce idle time. In large reasoning-oriented systems, this design is often combined with explicit communication–computation overlap to reduce pipeline bubbles further [156].

PP also interacts with optimization stability. Interleaved or asynchronous schedules may introduce weight staleness, where forward and backward passes for a micro-batch observe different parameter versions. In RLHF, where gradients are already noisy, this can destabilize training unless explicit versioning or consistency mechanisms are used [251, 262]. Finally, PP strengthens the case for disaggregated placement: placing actor, critic, reward, and reference on separate device groups allows each component to use a PP configuration matched to its role, instead of forcing a single compromise configuration for the whole pipeline [117, 251].

3.6 Expert Parallelism

Expert parallelism (EP) is the standard way to scale Mixture-of-Experts (MoE) models by distributing whole experts across devices. Unlike DP, which replicates the full model, or TP, which shards individual operators, EP assigns different experts to different GPUs or nodes. Since only a small subset of experts is activated per token, EP enables very large model capacity with much smaller per-token compute than an equally sized dense model [74,156]. Modern MoE designs also separate *shared* and *routed* experts. Shared experts are activated for all tokens and capture common linguistic features, while routed experts specialize in narrower domains such as mathematics or code. In practice, EP is rarely used alone; it is typically combined with TP, PP, and DP in hybrid multi-dimensional layouts [74, 156].

Formally, an MoE layer contains E experts and a router $G(x)$ that selects the top-k experts for each token representation x, with $k \ll E$. The output can be written as $y = \sum_{i=1}^{k} g_i(x) f_i(x)$, where f_i is the transformation of the i-th selected expert and $g_i(x)$ is its routing weight. EP exploits this sparsity by storing experts on different devices and executing only the selected ones [74].

EP's main systems cost is communication. Each MoE layer typically requires two all-to-all phases: *dispatch*, which sends token activations to the devices hosting the selected experts, and *combine*, which returns expert outputs to their originating ranks. As expert count and cluster size increase, this communication can dominate runtime, so practical EP deployments rely on hierarchical collectives, locality-aware routing, and communication-computation overlap [264].

3.6.1 Implications for RL-LLM Pipelines

EP can adapt a very large model while activating and updating only a small subset of parameters per token. This is particularly valuable for long reasoning trajectories, where dense models would make both rollout generation and policy updates prohibitively expensive [74, 156].

However, RL post-training makes routing harder. As the policy evolves, the token distribution shifts, so expert loads can become highly imbalanced. Load balancing schemes can help hardware efficiency, but they may also blur expert specialization by forcing artificially uniform routing [108]. For this reason, recent reasoning-oriented MoE systems increasingly prefer auxiliary-loss-free or bias-based balancing mechanisms that preserve specialization while correcting large load skew at the batch level [108, 156]. More broadly, EP in RLHF requires joint optimization of routing quality, communication cost, and systems balance: if routing is good but overloaded experts are poorly placed, all-to-all communication and straggler effects can erase the theoretical gains from sparse activation [178, 264].

EP also interacts strongly with RL framework design. Grouped sampling methods such as GRPO may create many structurally similar trajectories for the same prompt, which can stress the same experts simultaneously. Thus, efficient RL-MoE training often requires combining EP with dynamic load-balancing policies [117, 156, 178].

3.7 Complexity Analysis

We now analyze how intra-model parallelism changes the work, depth, and memory of the Transformer invocations inside RL-LLM pipelines; detailed derivations are in Appendix B.3. The same local formulas apply to the actor in Generation, the reward/reference/critic models in Assessment, and the policy or critic in Training; the RL-specific consequences come from whether the invocation is autoregressive, forward-only, or trainable.

Generation uses the actor in an autoregressive loop, so reducing the depth of one model invocation helps but cannot remove the outer T-step dependence. Assessment is teacher-forced and forward-only for frozen models, so it is mainly a batched-inference and memory-placement problem. Training requires backward passes, activations, gradients, and optimizer state, so memory sharding becomes central. Tables 5–8 therefore should be read as architecture-independent scaling laws for the individual model calls that compose the RL-LLM loop, not as hardware-calibrated throughput predictions.

A logical model invocation may be distributed across N participating ranks according to one or more parallelism dimensions. Depending on the parallelization strategy, each rank may operate on a partition of the input, parameters, layers, activations, or experts while other components remain replicated. We report both per-rank costs, which characterize local device pressure, and global costs, which aggregate costs across all N ranks participating in the complete logical model invocation. Global work is the sum of arithmetic FLOPs executed across these ranks, including replicated computation on every rank where it occurs. Global memory analogously sums resident state across all participating ranks. Depth denotes the critical path of the complete distributed execution.

Throughout Tables 5–8, we use an idealized arithmetic work–depth–memory model intended to expose scaling laws rather than predict hardware runtime. Unless stated otherwise, we exclude communication, synchronization, kernel-launch and scheduling overheads, finite-device utilization, load imbalance, and temporary communication workspaces. Pipeline parallelism assumes an approximately uniform partition of the L layers and does not model the number of microbatches, pipeline schedules, fill/drain bubbles, or inter-stage activation transfers. Training activation memory is represented by the leading-order term $B(S+T)Ld$, suppressing constant-factor storage for Q/K/V tensors, FFN intermediates, normalization temporaries, and other implementation-specific buffers; activation checkpointing is treated separately. The Adam model-state expressions count parameters, gradients, and first and second moments using a common element-size abstraction, omitting precision-specific byte factors and transient buffers. The TP activation terms use an idealized shardable-activation model; replicated token-wise components are suppressed. Attention is assumed not to materialize the full attention matrix, as in FlashAttention-style execution. For MoE models, the tables retain the dominant shared-Transformer and expert-FFN terms; router projection, softmax/top-k, auxiliary load-balancing operations, and their relatively small parameter state are omitted. Finally, the isolated EP row assumes that only expert FFNs are partitioned across P_e ranks while shared dense computation is replicated; the combined 5D row instead assumes a coupled execution in which EP ranks also process disjoint token/batch shards for the shared path, so dense arithmetic is not redundantly executed across the EP dimension.

Each parallelism strategy partitions a different dimension. DP splits the batch, reducing per-group work and activation memory but leaving the critical path unchanged. PP splits layers, reducing per-stage work, memory, and architectural depth by replacing L with L/P_p, though realized runtime also depends on microbatch bubbles. TP splits hidden-dimensional operators, reducing per-device GEMM work and replacing hidden-dimension depth terms by $\log(d/P_t)$. CP splits the sequence dimension, reducing context-dependent attention and activation memory, which is crucial for long reasoning traces. EP splits MoE experts, reducing per-device expert memory and compute while leaving dense attention largely unchanged. Finally, 3D/5D parallelism combines data, pipeline, and tensor sharding, giving the strongest per-device memory reduction but also the most communication and scheduling constraints.

For RL-LLMs, this means DP is usually best for throughput-oriented batched Generation or Assessment when models fit per device; TP/PP are needed for large actors, critics, or low-latency single invocations; CP is most useful for long-context reasoning; and 3D/5D/ZeRO-style sharding is most important during Training, where optimizer state and activations dominate memory.

Note that **pipeline parallelism** does *not* reduce the *global* depth layer-wise dependency from L to L/P_p; the L/P_p factor appears only in the per-rank depth. This is because, for global end-to-end critical path, the microbatch still passes through all P_p pipeline stages, so it traverses all L layers.

Context parallelism reduces the number of query positions and stored activations per device, but each query still depends on the global key/value context. Therefore, under the work–depth model, CP does not replace the global attention reduction length S by S/P_c; its principal benefits are reduced per-rank work and memory. Distributed attention additionally incurs communication-round dependencies, which are outside the present FLOP-based depth abstraction.

3.7.1 ZeRO & FSDP

The memory expressions in Tables 5–8 use a simplified Adam model-state factor consisting of parameters, gradients, and first and second moments. ZeRO [201] and FSDP [277] refine this term without changing the layerwise Transformer computation. If P denotes the parameter count of the trainable model and P_z the sharding degree, then standard data parallelism stores approximately $4P$ model-state elements per replica. ZeRO-1 shards only optimizer states, giving $P + P + 2P/P_z$; ZeRO-2 shards optimizer states and gradients, giving $P + 3P/P_z$; and ZeRO-3/FSDP shards parameters, gradients, and optimizer states, giving approximately $4P/P_z$, up to transient all-gather buffers.

3.7.2 Activation Checkpointing

The activation terms in the tables correspond to stored training activations without checkpointing. Activation checkpointing reduces this term by storing only a subset of intermediate activations and recomputing the missing ones during backpropagation. In our notation, this replaces M_{Act} by $\kappa_{\text{ckpt}} M_{\text{Act}}$ for some $0 < \kappa_{\text{ckpt}} < 1$, while increasing training work and depth by the extra forward recomputation required during the backward pass. This trade-off is most relevant for actor and critic training, especially for long reasoning trajectories.

3.8 Key Insights & Takeaways

The main lesson is *stage-aware hybridization*: no single intra-model strategy is best for the whole RL-LLM loop. Generation, Assessment, and Training invoke similar Transformer building blocks, but they stress different dimensions of the system. Generation is autoregressive and latency-sensitive; Assessment is mostly teacher-forced, forward-only, and throughput-oriented; Training is memory-intensive as it uses activations, gradients, and optimizer state.

▲ **DP scales throughput, not single-sample latency.** Data parallelism is ideal for increasing the number of prompts,

Parallelism	Work	Depth	Memory
None†	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(Ld^2+Vd+BmLd\big)$
Data	$O\Big(\frac{BmL(d^2+md)}{P_d}\Big)$	$O\Big(L[\log d+\log m]+\log\Big(\frac{B}{P_d}m\Big)\Big)$	$O\Big(Ld^2+Vd+\frac{BmLd}{P_d}\Big)$
Pipeline*	$O\Big(\frac{BmL(d^2+md)}{P_p}\Big)$	$O\Big(\frac{L}{P_p}[\log d+\log m]+\log(Bm)\Big)$	$O\Big(\frac{Ld^2}{P_p}+Vd+\frac{BmLd}{P_p}\Big)$
Tensor	$O\Big(\frac{BmL(d^2+md)}{P_t}\Big)$	$O\Big(L\Big[\log\frac{d}{P_t}+\log m\Big]+\log(Bm)\Big)$	$O\Big(\frac{Ld^2+Vd}{P_t}+\frac{BmLd}{P_t}\Big)$
Context	$O\Big(\frac{BmL(d^2+md)}{P_c}\Big)$	$O\Big(L[\log d+\log m]+\log\Big(\frac{Bm}{P_c}\Big)\Big)$	$O\Big(Ld^2+Vd+\frac{BmLd}{P_c}\Big)$
Expert	$O\Big(BmL\Big[(d^2+md)+\frac{E_a dd_e}{P_e}\Big]\Big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\Big(L\Big[d^2+\frac{Edd_e}{P_e}\Big]+Vd+BmLd\Big)$
3D‡	$O\Big(\frac{BmL(d^2+md)}{P_dP_pP_t}\Big)$	$O\Big(\frac{L}{P_p}\Big[\log\frac{d}{P_t}+\log m\Big]+\log\Big(\frac{B}{P_d}m\Big)\Big)$	$O\Big(\frac{Ld^2+Vd}{P_pP_t}+\frac{BmLd}{P_dP_pP_t}\Big)$
5D‡	$O\Big(\frac{BmL}{P_dP_eP_cP_pP_t}\big[d^2+md+E_add_e\big]\Big)$	$O\Big(\frac{L}{P_p}\Big[\log\frac{d}{P_t}+\log m\Big]+\log\Big(\frac{Bm}{P_dP_eP_c}\Big)\Big)$	$O\Big(\frac{Ld^2}{P_pP_t}+\frac{LEdd_e}{P_pP_tP_e}+\frac{Vd}{P_t}+\frac{BmLd}{P_dP_eP_cP_pP_t}\Big)$

TABLE 5: **Complexity Analysis (Per rank).** Asymptotic work, depth, and memory for a single Transformer training iteration (forward and backward pass), expressed in Big-O notation. For clarity, we use $m := S+T$. The work expressions use the regime where $d_{ff} = O(d)$. †: Baseline configuration without parallelism. *For PP, the Vd memory term denotes the embedding/output state resident on boundary pipeline ranks; interior ranks need not store this state. Thus, this term represents the boundary/peak per-rank footprint rather than replication across all P_p ranks. ‡: 3D and 5D parallelism combine data, pipeline, tensor (3D), context, and expert (5D) parallelism, with $P_dP_pP_t = N$ (3D) and $P_dP_pP_tP_cP_e = N$ (5D) total devices. For the combined 5D configuration, we assume the EP dimension also partitions source-token ownership for shared-layer computation. Training memory assumes Adam without activation checkpointing.

Parallelism	Work	Depth	Memory
None	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(Ld^2+Vd+BmLd\big)$
Data	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(P_d[Ld^2+Vd]+BmLd\big)$
Pipeline	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(Ld^2+Vd+BmLd\big)$
Tensor	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(Ld^2+Vd+BmLd\big)$
Context	$O\big(BmL(d^2+md)\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(P_c[Ld^2+Vd]+BmLd\big)$
Expert	$O\big(BmL\big[P_e(d^2+md)+E_add_e\big]\big)$	$O(L[\log d+\log m]+\log(Bm))$	$O\big(P_e[Ld^2+Vd+BmLd]+LEdd_e\big)$
3D	$O\big(BmL(d^2+md)\big)$	$O(L\,[\log d+\log m]+\log(Bm))$	$O\big(P_d[Ld^2+Vd]+BmLd\big)$
5D	$O\big(BmL\big[d^2+md+E_add_e\big]\big)$	$O(L\,[\log d+\log m]+\log(Bm))$	$O\big(P_dP_cP_e[Ld^2+Vd]+P_dP_cLEdd_e+BmLd\big)$

TABLE 6: **Complexity Analysis (Global).** Asymptotic work, depth, and memory for a single Transformer training iteration (forward & backward pass). For clarity, we use $m := S+T$. The work expressions use the regime where $d_{ff} = O(d)$.

Parallelism	Work	Depth	Memory
None	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\Big(\frac{d^4d_{\text{ff}}m}{h}\Big)+\log(Bm)$	$L(16d^2+8dd_{\text{ff}})+4Vd+BmLd$
Data	$6\frac{B}{P_d}mL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\Big(\frac{d^4d_{\text{ff}}m}{h}\Big)+\log\Big(\frac{B}{P_d}m\Big)$	$L(16d^2+8dd_{\text{ff}})+4Vd+\frac{B}{P_d}mLd$
Pipeline	$6Bm\frac{L}{P_p}\left(4d^2+2dd_{\text{ff}}+md\right)$	$2\frac{L}{P_p}\log\Big(\frac{d^4d_{\text{ff}}m}{h}\Big)+\log(Bm)$	$\frac{L}{P_p}(16d^2+8dd_{\text{ff}})+4Vd+Bm\frac{L}{P_p}d$
Tensor	$6BmL\left(\frac{4d^2+2dd_{\text{ff}}+md}{P_t}\right)$	$2L\log\Big(\frac{d^4d_{\text{ff}}m}{P_t^2h}\Big)+\log(Bm)$	$\frac{L(16d^2+8dd_{\text{ff}})}{P_t}+\frac{4Vd}{P_t}+\frac{BmLd}{P_t}$
Context	$6B\frac{S+T}{P_c}L\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\Big(\frac{d^4d_{\text{ff}}m}{h}\Big)+\log\Big(\frac{Bm}{P_c}\Big)$	$L(16d^2+8dd_{\text{ff}})+4Vd+B\frac{S+T}{P_c}Ld$
Expert	$6BmL\left(4d^2+\frac{2E_add_e}{P_e}+md\right)$	$2L\log\Big(\frac{d^4d_em}{h}\Big)+\log(Bm)$	$L\left(16d^2+\frac{8Edd_e}{P_e}\right)+4Vd+BmLd$
3D	$6\frac{B}{P_d}m\frac{L}{P_p}\frac{4d^2+2dd_{\text{ff}}+md}{P_t}$	$2\frac{L}{P_p}\log\Big(\frac{d^4d_{\text{ff}}m}{P_t^2h}\Big)+\log\Big(\frac{B}{P_d}m\Big)$	$\frac{L(16d^2+8dd_{\text{ff}})}{P_pP_t}+\frac{4Vd}{P_t}+\frac{BmLd}{P_dP_pP_t}$
5D	$6\frac{B}{P_dP_e}\frac{m}{P_c}\frac{L}{P_p}\left(\frac{4d^2+2E_add_e+md}{P_t}\right)$	$2\frac{L}{P_p}\log\Big(\frac{d^4d_em}{P_t^2h}\Big)+\log\Big(\frac{Bm}{P_dP_eP_c}\Big)$	$\frac{16Ld^2}{P_pP_t}+\frac{8LEdd_e}{P_pP_tP_e}+\frac{4Vd}{P_t}+\frac{BmLd}{P_dP_eP_cP_pP_t}$

TABLE 7: **Complexity Analysis (Per rank).** Explicit leading-order costs under our simplified Transformer cost model, for a single Transformer training iteration (forward and backward pass), including explicit constant factors. For clarity, we use $m := S+T$.

Parallelism	Work	Depth	Memory
None	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$L(16d^2+8dd_{\text{ff}})+4Vd+BmLd$
Data	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$P_d\left[L(16d^2+8dd_{\text{ff}})+4Vd\right]+BmLd$
Pipeline	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$L(16d^2+8dd_{\text{ff}})+4Vd+BmLd$
Tensor	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$L(16d^2+8dd_{\text{ff}})+4Vd+BmLd$
Context	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$P_c\left[L(16d^2+8dd_{\text{ff}})+4Vd\right]+BmLd$
Expert	$6BmL\left(P_e\left(4d^2+md\right)+2E_add_e\right)$	$2L\log\big(d^4d_em/h\big)+\log(Bm)$	$P_e\left(16Ld^2+4Vd+BmLd\right)+8LEdd_e$
3D	$6BmL\left(4d^2+2dd_{\text{ff}}+md\right)$	$2L\log\big(d^4d_{\text{ff}}m/h\big)+\log(Bm)$	$P_d\left[L(16d^2+8dd_{\text{ff}})+4Vd\right]+BmLd$
5D	$6BmL\left(4d^2+2E_add_e+md\right)$	$2L\log\big(d^4d_em/h\big)+\log(Bm)$	$P_dP_cP_e\left(16Ld^2+4Vd\right)+8P_dP_cLEdd_e+BmLd$

TABLE 8: **Complexity Analysis (Global).** Explicit leading-order costs under our simplified Transformer cost model, for a single Transformer training iteration (forward and backward pass), including explicit constant factors. For clarity, we use $m := S+T$.

completions, or preference pairs processed per unit time. In online Generation, it can process more prompts or candidates in parallel, but each replica still executes the full autoregressive chain $D_{\text{gen}}(\pi_\theta; S, T) = O(TD_f(\pi_\theta; S + T))$. Thus, DP does not shorten the critical path of one rollout trajectory or one model invocation.

▲ **TP and PP are capacity/latency tools with communication costs.** Tensor and pipeline parallelism are useful when a model does not fit on one device or when the latency of a single invocation must be reduced. In Generation, they reduce the cost of each actor invocation but do not remove the outer token-by-token dependence. In Assessment and Training, they help execute larger reward, reference, actor, or critic models. Their benefit must be balanced against TP collectives, PP activation transfers, pipeline bubbles, and resharding overheads.

▲ **SP/CP matter increasingly for reasoning.** Long CoT traces, tool histories, retrieved documents, verifier outputs, and multi-turn interaction histories increase $S + T$, making activation and attention memory central bottlenecks. Sequence and context parallelism directly target this regime by partitioning the sequence/context dimension. They are especially useful for long-context Assessment and Training, and for actor Generation when the rollout context exceeds single-device memory.

▲ **Frozen and trainable models prefer different layouts.** Reward and reference models are usually frozen and forward-only, so they often scale best as replicated batched-inference services when they fit per device. The actor and critic are trainable, and therefore dominate memory through activations, gradients, and optimizer state. Consequently, actor/critic Training benefits most from ZeRO/FSDP-style sharding, TP/PP, and 3D/5D combinations. PPO is the most demanding case because it may train both π_θ and V_ψ, whereas GRPO and DPO train only π_θ.

▲ **EP scales capability but can create systems skew.** MoE policies can increase reasoning capacity at limited per-token compute by activating only a subset of experts. However, dense attention remains on the critical path, and expert routing introduces all-to-all communication. RL post-training further complicates EP because the evolving policy changes the token distribution, which can create expert-load imbalance. Efficient EP for RLMs therefore requires routing-aware load balancing and communication-aware expert placement.

▲ **Practical RLM systems should specialize by model and stage.** Actor Generation may prioritize KV-cache memory, rollout latency, and batching; reward/reference Assessment may use cheap replicated inference; actor/critic Training may require 3D parallelism and optimizer-state sharding. This intra-model specialization is precisely what motivates the inter-model placement, stage fusion, hybrid execution, and asynchronous strategies discussed next.

4 Inter-Model Parallelism

We organize inter-model parallelism into five categories. **Model structure configurations** decide whether trainable components such as the actor and critic share parameters. **Model placement strategies** decide whether model roles are co-located, partially co-located, or disaggregated across device groups. **Stage fusion** removes coarse barriers between stages or sub-stages by streaming partial outputs downstream. **Hybrid execution** allows different invocations of the same model to use different intra-model layouts, often requiring resharding or automated planning. Finally, **asynchronous execution** overlaps different RL iterations by allowing bounded-stale parameter reads. These categories are often combined in high-performance systems such as ReaL, RLHFuse, HybridFlow/verl, OpenRLHF, StreamRL, and AReaL [92, 117, 169, 217, 281, 282]. Figures 8 and 9 illustrate this taxonomy.

4.1 Model Structure Configurations

Model structure configurations determine the *architectural relation between the trainable model components* in the RL-LLM loop. The central question is whether the actor π_θ and critic V_ψ are implemented as independent models, as a shared actor–critic network, or whether the critic is removed by the learning algorithm. PPO-style RLHF uses an actor and a value model, so this choice directly affects work, memory, and optimization stability [208, 279]. Critic-free methods such as GRPO remove V_ψ and replace value-based advantage estimation by group-relative normalization or related baselines, changing both the algorithm and the systems profile [107, 213].

In a **shared actor–critic configuration**, a single Transformer backbone f_ω produces hidden states that feed two task-specific heads: a policy head h_π for token logits and a value head h_V for scalar value estimates (i.e., $\pi_\theta(\cdot \mid x, y_{<t}) = h_\pi(f_\omega(x, y_{<t}))$ and $V_\psi(x, y_{<t}) = h_V(f_\omega(x, y_{<t}))$). The main systems benefit is that the actor and critic no longer require two separate backbones. This reduces parameter, gradient, optimizer-state, and activation memory, and can replace separate actor/critic forward passes by one shared backbone pass followed by two small heads. In a PPO-style pipeline, this is an inter-model decision: it changes the number of trainable model invocations in Assessment and Training, not merely the internal architecture of one model.

The trade-off is optimization coupling. The policy-gradient loss and value loss both update f_ω, so their gradients may pull the shared representation in different directions. A single backbone must serve both action selection and return prediction, and a shared optimizer schedule must accommodate losses with different scales and curvature. In practice, this often requires careful value loss weighting, separate learning rates for heads, or partial sharing in which only lower layers are shared and upper layers remain task-specific. Shared actor–critic is therefore most attractive when memory is scarce or when the critic is close in scale to the actor; it is less attractive when value learning requires substantially different representations or optimization dynamics.

With **independent actor and critic models**, π_θ and V_ψ have separate backbones, optimizer states, and hyperparameters. This avoids gradient interference and allows the critic to be sized differently from the actor, for example by using a smaller critic to reduce memory and compute. The cost is a larger model-state footprint and a separate critic forward/backward path. In the work–depth analysis of Section 2.5, this is exactly the additional PPO cost

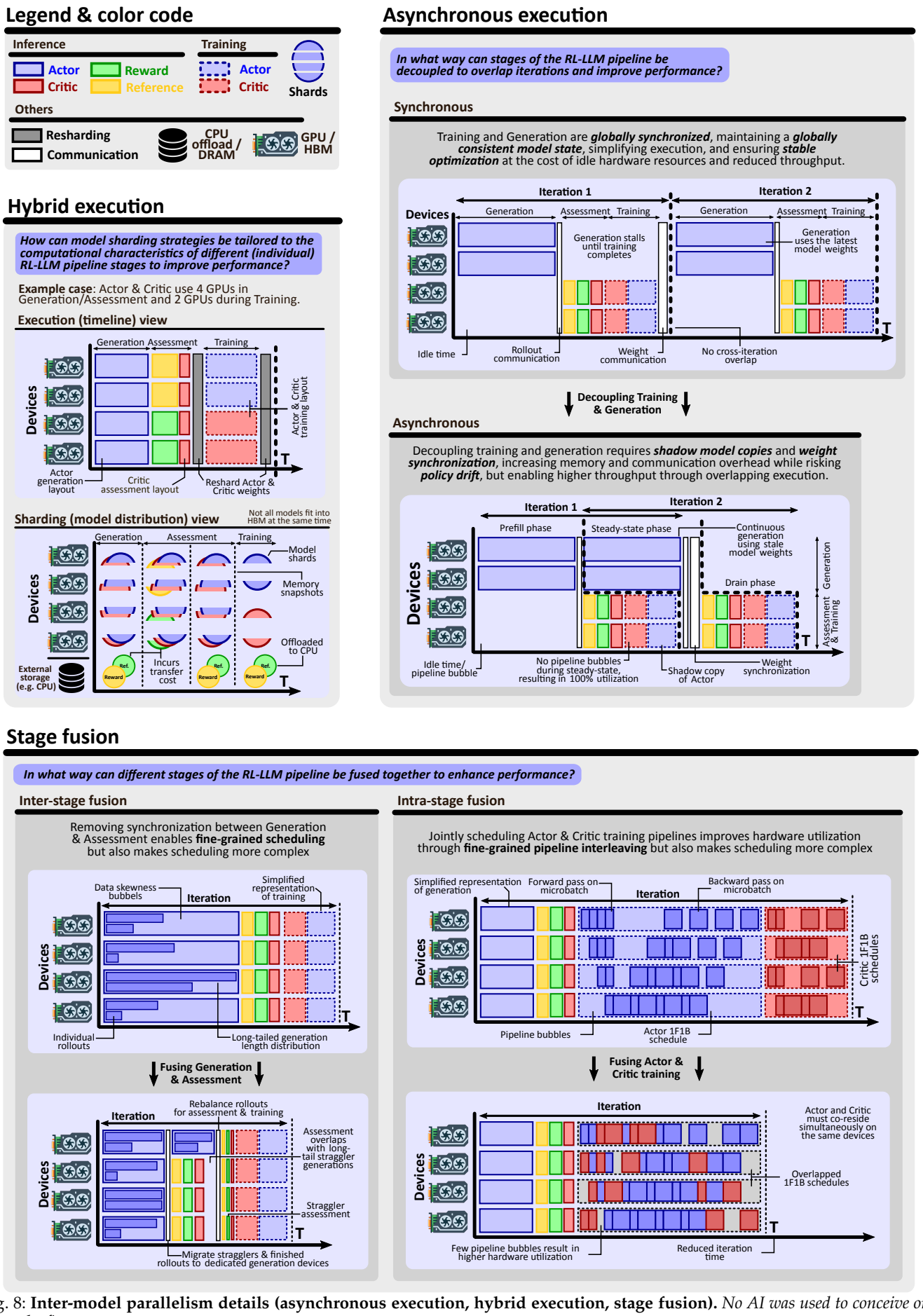


Fig. 8: **Inter-model parallelism details (asynchronous execution, hybrid execution, stage fusion).** *No AI was used to conceive or to draw the figure.*

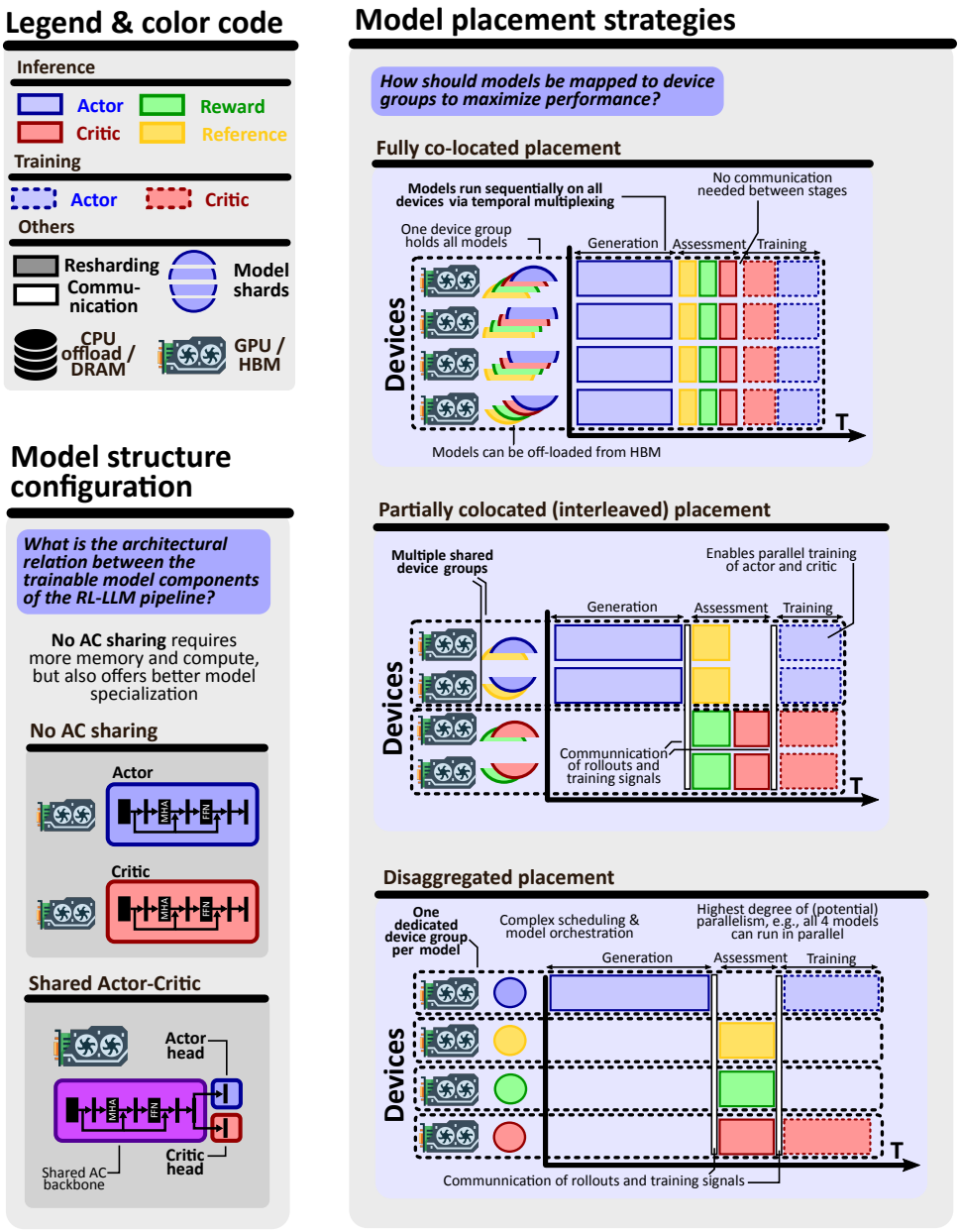


Fig. 9: **Inter-model parallelism details (placement strategies, model structure configurations).** *No AI was used to conceive or to draw the figure.*

$\Theta\Big(BK(S+T)[C_f^{\text{tok}}(V_\psi)+C_b^{\text{tok}}(V_\psi)]\Big)$ plus the corresponding critic model-state memory.

Finally, **critic-free methods** move this trade-off from architecture to algorithm design. GRPO removes the learned value model and computes relative advantages from groups of completions sampled for the same prompt, while DPO removes the online actor–critic loop altogether and optimizes over preference pairs.

4.2 Model Placement Strategies

Model placement specifies the mapping from model execution to device groups. A device group is a set of GPUs that jointly stores and executes one or more model invocations, possibly using an intra-model strategy internally. The placement decision controls aspects such as memory co-residency, inter-stage communication, and the amount of concurrency available across model roles.

In **fully co-located placement**, all model roles share one device group and execute through temporal multiplexing. The same pool of GPUs first performs actor generation, then reward/reference/critic inference, and then actor/critic training. This maximizes locality: intermediate tensors can remain within the same device group, and no inter-group communication is needed to pass rollouts, log-probabilities, values, or rewards. It also simplifies orchestration because the runtime schedules one device pool rather than multiple distributed services.

The cost is co-resident memory pressure and poor role specialization. The device group must be provisioned for the largest memory requirement, often Training, yet it is also used for Generation, which has different arithmetic intensity and is often memory-bandwidth or latency dominated. Moreover, "redundant memory" in this context means that the same devices must hold the resident state for multiple model roles even when only one role is active. If the system

uses data parallelism, this co-residency may be replicated across data-parallel groups: each replica may need actor, critic, reward, and reference state or the ability to materialize them. To mitigate this, one can employ techniques such as offloading, loading and unloading idle models, or weight gathering from sharded states. These techniques reduce memory pressure but introduce context-switching and data-movement overheads. As a result, fully co-located execution is simple and locality-friendly, but it often leaves hardware idle because the stage service times and resource needs are highly imbalanced.

Partially co-located, or interleaved, placement sits between full colocation and full disaggregation. Selected models or stages share a device group; others are placed separately. For example, actor generation and training may share a group to avoid repeated actor-weight transfers, while reward and reference inference run on smaller inference-oriented groups. Alternatively, reward and reference models may be co-located because both are frozen and forward-only, while the trainable actor and critic are separated.

This design is useful when some model pairs benefit from locality while others benefit from concurrency. It reduces the worst co-residency pressure of full colocation without requiring every stage boundary to become a network boundary. Yet, it also complicates scheduling: groups that share some but not all models must coordinate execution order, memory residency, and data handoff. The best interleaving depends on model sizes, stage service times, cluster topology, and whether a given edge carries small metadata, token IDs, activations, or large parameter shards.

In **disaggregated placement**, different model roles or stages run on dedicated, disjoint device groups. The actor group stores and executes the actor, the reward group stores the reward model, the critic group stores the value model, and the reference group stores the frozen reference model. This eliminates cross-model co-residency: a reward-model GPU no longer needs actor or critic weights, and a training GPU no longer needs to keep frozen reward/reference models resident. It also enables heterogeneous hardware specialization. Large trainable actors can be assigned to high-memory GPUs, while smaller frozen reward or reference models can run on cheaper or inference-optimized devices. StreamRL explicitly argues for such disaggregation because colocated generation and training couple resources with very different compute and memory profiles, while disaggregated stream generation enables flexible resource allocation, heterogeneous training setups, and even cross-datacenter deployment [281].

Disaggregation is a prerequisite for many forms of inter-model concurrency, but it is not by itself a speedup. If the stages still execute strictly Generation $\rightarrow$ Assessment $\rightarrow$ Training, then the iteration latency remains the sum of stage times and some groups will wait idle. The benefit appears when disaggregation is combined with stage fusion, streaming, or asynchronous execution. The price is explicit communication: rollouts, token IDs, log-probabilities, rewards, values, and sometimes updated parameters must cross device-group boundaries. Systems such as OpenRLHF, HybridFlow/verl, ReaL, FlexRLHF, and StreamRL explore different points in this placement space, using Ray-style orchestration, hierarchical APIs, parameter reallocation, or streaming data paths to manage the additional complexity [117, 169, 217, 251, 281].

4.3 Stage Fusion

Stage fusion removes coarse barriers between consecutive stages or between subtasks within a stage. In the baseline RLHF loop, the Assessment stage waits until the entire Generation batch has completed, and Training waits until all assessment outputs are ready. This barrier structure is inefficient for RLMs because generation lengths are long-tailed: short trajectories may finish much earlier than the longest ones, yet their reward/reference/value inference is delayed by stragglers. Stage fusion decomposes stages into finer-grained subtasks and schedules downstream work as soon as its input is available.

There are two common forms. In *inter-stage fusion*, completed samples are streamed from Generation into Assessment without waiting for the whole rollout batch. A short completion can be scored by the reward model and processed by the reference or critic while the actor is still decoding longer completions. This turns generation-length skewness into useful overlap. In *intra-stage fusion*, operators within the same stage interleave microbatches or pipeline schedules. For example, actor and critic training pipelines may be interleaved so that one model's bubble time is filled by another model's forward or backward microbatch.

RLHFuse is a canonical example: it splits generation and assessment into sample-level subtasks for inter-stage fusion and splits training into microbatch-level subtasks for intra-stage fusion, enhancing throughput by up to $3.7\times$ by mitigating long-tail generation skew and pipeline bubbles [282]. Related overlap-oriented work such as OPPO also targets PPO-style serialization and long-tail response lengths by overlapping pipeline stages [255]. The key point is that fusion does not reduce total work; it reduces depth and idle time by changing when ready subtasks may execute.

Fusion is throughput-oriented and can increase single-sample latency. A sample that finishes early may wait in a queue until enough samples form an efficient microbatch, or it may be delayed by backpressure from a downstream stage. Thus, the steady-state time per batch can improve even if the time from one sample's generation to its training update increases. Fusion also increases peak memory because multiple stages are live simultaneously: model weights, KV caches, activations, queues, and communication buffers may coexist. The design problem is therefore to choose a granularity that is fine enough to hide skew and bubbles but not so fine that scheduling overhead, fragmentation, or memory pressure dominates.

4.4 Hybrid Parallelism and Adaptive Scheduling

Hybrid parallelism means that different model invocations in the same RL-LLM pipeline may use different intra-model strategies, device allocations, or execution modes. This is distinct from ordinary 3D parallelism inside one model: the same actor π_θ may be invoked once for autoregressive generation and later for training, and these two invocations have different optimal layouts. Generation favors low-latency inference and efficient KV-cache use while Training favors higher memory capacity, efficient gradient synchronization, activation checkpointing, or optimizer-state

sharding. A single static layout forces a compromise; hybrid execution allows each role to use a matched layout.

For example, if the actor is stored in a ZeRO/FSDP-style sharded training layout, generation may require gathering or resharding the weights into an inference-centered layout. Conversely, after generation, the system may need to redistribute weights or optimizer state back into the training layout. DeepSpeed-Chat's Hybrid Engine is an early example of this idea, combining training-mode memory optimizations with inference-mode kernel and parallelism optimizations [261]. ReaL generalizes the idea as parameter reallocation: an execution plan chooses role-specific allocations and parallelization strategies, and the runtime redistributes parameters between them [169]. HybridFlow/verl's 3D-HybridEngine similarly targets efficient actor resharding between training and generation with low redundancy, while NeMo RL exposes a mode that can train with pipeline parallelism but run TensorRT-LLM inference in a tensor-parallel layout [184, 214, 217].

The cost is *resharding*. When two invocations of the same model use incompatible layouts or different device groups, parameter shards must be communicated and repartitioned. During the transfer, the system may need extra memory for source and destination layouts, and the transfer may sit on the critical path unless it is overlapped with unrelated work. Hybrid execution is profitable only when the per-stage gains from using specialized layouts exceed the resharding cost.

Pipe-RLHF is an example of computation-mode-aware parallelism: it uses stage-specific parallelization to improve resource utilization [262]. ReaL and HybridFlow also move in this direction by representing the RLHF pipeline as a dataflow or execution plan rather than a fixed sequence of scripts [169, 217]. The trade-off is engineering complexity: an adaptive scheduler requires profiling, cost modeling, memory feasibility checks, and robust orchestration. It is most valuable at scales where a static hand-designed layout leaves substantial hardware idle.

4.5 Asynchronous Execution

Stage fusion overlaps work within an iteration. Asynchronous execution overlaps different RL iterations by relaxing cross-iteration parameter freshness. In a synchronous loop, iteration $k+1$ cannot generate data until iteration k has completed Training and published updated parameters. In an asynchronous loop, Generation or Assessment may read a bounded-stale snapshot while Training from the previous iteration has still not completed.

Bounded asynchrony introduces a systems–algorithm trade-off. Systems benefit because rollout workers and learners no longer wait for one another. This can substantially improve utilization when generation is long-tailed or much slower than training. Asynchronous RLHF explicitly studies this off-policy setting and shows that generation and learning can be decoupled for better efficiency, while AReaL further develops a fully asynchronous RL system with a staleness-aware PPO [92, 180]. Laminar pushes the systems side further through trajectory-level asynchrony and a distributed parameter relay tier, while StaleFlow explicitly coordinates rollouts under staleness constraints to balance convergence and throughput [145, 216].

The algorithmic risk is *policy drift*. A trajectory may have been sampled from a behavior policy π_{beh}, while the learner updates a newer policy π_θ. To alleviate this, systems can track statistics such as staleness, token-level KL, ratio variance, clip fraction, effective sample size, and reward/advantage shifts. Standard PPO tolerates small drift because clipped importance ratios limit the update, but large staleness can move the trust-region center toward an outdated low-quality policy. AReaL addresses this by separating the behavior policy used for off-policy correction from the proximal policy used as the trust-region center.

Asynchrony also changes memory accounting. If generation and training run on disjoint device groups and must proceed concurrently, generation needs a stable read-only snapshot while training updates another copy. This creates a *shadow pair*: a training copy with optimizer state and a generation copy used for inference. In the worst case, this roughly doubles the parameter memory for each asynchronously consumed trainable model. The factor is not necessarily a full $2\times$ increase in total training memory, because optimizer states usually remain only on the training side and the shadow copy may be BF16, quantized, or offloaded; nevertheless, any table or memory model for asynchronous execution should include an additional persistent inference-side parameter copy whenever trainable models are disaggregated across stale readers and writers.

4.6 Complexity Analysis

We now rigorously quantify the inter-model strategies. The analysis is conducted for a PPO-style online RL-LLM pipeline, which is the most demanding common setting; GRPO and DPO can be recovered by deleting the critic and/or online assessment components. Detailed derivations are in Appendix B.4.

We first define stage-level terms. **Generation** comes with

$$\begin{aligned} W_G &= BK\, C_{\mathrm{gen}}(\pi_\theta) && \text{(work)},\\ D_G &= D_{\mathrm{gen}}(\pi_\theta) && \text{(depth)},\\ M_G &= |\pi_\theta| + BK\, M_{\mathrm{KV}} && \text{(memory)}. \end{aligned}$$

Here C_{gen} and D_{gen} include the prefill–decode decomposition from Table 3.

Next, the work of the **assessment** stage is

$$W_A = BK(S+T)\,[C_f^{\mathrm{tok}}(\pi_{\mathrm{ref}}) + C_f^{\mathrm{tok}}(R_\varphi) + C_f^{\mathrm{tok}}(V_\psi)].$$

For depth, we distinguish co-located sequential execution

$$D_A^{\Sigma} = D_f(\pi_{\mathrm{ref}}) + D_f(R_\varphi) + D_f(V_\psi),$$

from disaggregated execution

$$D_A^{\max} = \max\{D_f(\pi_{\mathrm{ref}}), D_f(R_\varphi), D_f(V_\psi)\}.$$

Similarly, **training** work is

$$W_T = BK(S+T)\,[C_{\mathrm{tr}}^{\mathrm{tok}}(\pi_\theta) + C_{\mathrm{tr}}^{\mathrm{tok}}(V_\psi)],$$

with sequential and disaggregated depths

$$D_T^{\Sigma} = D_{\mathrm{tr}}(\pi_\theta) + D_{\mathrm{tr}}(V_\psi), \quad D_T^{\max} = \max\{D_{\mathrm{tr}}(\pi_\theta), D_{\mathrm{tr}}(V_\psi)\}.$$

For memory, we use M_A^{Σ} for the co-resident assessment footprint, M_T^{Σ} for the co-resident actor–critic training footprint, and $M_T^{\max}$ for the maximum over disaggregated actor

Config	Stage	Global Work	Global Depth	Peak Memory per-device group
Baseline	Gen.	$BK \cdot C_{\text{gen}}(\pi_\theta)$	$D_{\text{gen}}(\pi_\theta)$	$\lvert\pi_\theta\rvert + BK \cdot M_{\text{KV}}$
	Assess.	$BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)]$	$D_f(\pi_{\text{ref}}) + D_f(R_\varphi) + D_f(V_\psi)$	$\lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + \lvert V_\psi\rvert$ $+BK(S+T)$ $\cdot \max\{M_{\text{Inf}}(\pi_{\text{ref}}), M_{\text{Inf}}(R_\varphi), M_{\text{Inf}}(V_\psi)\}$
	Train.	$BK(S+T) \cdot [C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$D_{\text{tr}}(\pi_\theta) + D_{\text{tr}}(V_\psi)$	$4 \cdot (\lvert\pi_\theta\rvert + \lvert V_\psi\rvert)$ $+BK(S+T)(L_\pi d_\pi + L_\psi d_\psi)$
AC shared	Gen.	$BK \cdot C_{\text{gen}}(\pi_{\text{SH}})$	$D_{\text{gen}}(\pi_{\text{SH}})$	$\lvert\pi_{\text{SH}}\rvert + BK \cdot M_{\text{KV}}$
	Assess.	$BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(\pi_{\text{SH}})]$	$D_f(\pi_{\text{ref}}) + D_f(R_\varphi) + D_f(\pi_{\text{SH}})$	$\lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + \lvert\pi_{\text{SH}}\rvert$ $+BK(S+T)$ $\cdot \max\{M_{\text{Inf}}(\pi_{\text{ref}}), M_{\text{Inf}}(R_\varphi), M_{\text{Inf}}(\pi_{\text{SH}})\}$
	Train.	$BK(S+T) \cdot C_{\text{tr}}^{\text{tok}}(\pi_{\text{SH}})$	$D_{\text{tr}}(\pi_{\text{SH}})$	$4 \cdot \lvert\pi_{\text{SH}}\rvert + BK(S+T)L_{\text{SH}}d_{\text{SH}}$
Disagg.	Gen.	$BK \cdot C_{\text{gen}}(\pi_\theta)$	$D_{\text{gen}}(\pi_\theta)$	$\lvert\pi_\theta\rvert + BK \cdot M_{\text{KV}}$
	Assess.	$BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)]$	$\max\{D_f(\pi_{\text{ref}}), D_f(R_\varphi), D_f(V_\psi)\}$	$\max\{\lvert\pi_{\text{ref}}\rvert + BK(S+T)M_{\text{Inf}}(\pi_{\text{ref}}),$ $\lvert R_\varphi\rvert + BK(S+T)M_{\text{Inf}}(R_\varphi),$ $\lvert V_\psi\rvert + BK(S+T)M_{\text{Inf}}(V_\psi)\}$
	Train.	$BK(S+T) \cdot [C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$\max\{D_{\text{tr}}(\pi_\theta), D_{\text{tr}}(V_\psi)\}$	$\max\{4\lvert\pi_\theta\rvert + BK(S+T)L_\pi d_\pi,$ $4\lvert V_\psi\rvert + BK(S+T)L_\psi d_\psi\}$
Hybrid	Gen.	$BK \cdot C_{\text{gen}}(\pi_\theta)$	$D_{\text{gen}}^{\text{TP}}(\pi_\theta)$	$\lvert\pi_\theta\rvert + BK \cdot M_{\text{KV}}$
	Assess.	$BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)]$	$D_f^{\text{TP}}(\pi_{\text{ref}}) + D_f^{\text{TP}}(R_\varphi) + D_f^{\text{TP}}(V_\psi)$	$\lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + \lvert V_\psi\rvert$ $+BK(S+T)$ $\cdot \max\{M_{\text{Inf}}(\pi_{\text{ref}}), M_{\text{Inf}}(R_\varphi), M_{\text{Inf}}(V_\psi)\}$
	Train.	$BK(S+T) \cdot [C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$D_{\text{tr}}(\pi_\theta) + D_{\text{tr}}(V_\psi)$	$4 \cdot (\lvert\pi_\theta\rvert + \lvert V_\psi\rvert)$ $+BK(S+T)(L_\pi d_\pi + L_\psi d_\psi)$
Stage F.	Inter	$BK\, C_{\text{gen}}(\pi_\theta) + BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)]$	$\max\{D_{\text{gen}}(\pi_\theta), D_f(\pi_{\text{ref}}),$ $D_f(R_\varphi), D_f(V_\psi)\}$	$\max\{\lvert\pi_\theta\rvert + BKM_{\text{KV}},\ \lvert\pi_{\text{ref}}\rvert + \lvert R_\varphi\rvert + \lvert V_\psi\rvert\}$
	Intra	$BK(S+T) \cdot [C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$\max\{D_{\text{tr}}(\pi_\theta), D_{\text{tr}}(V_\psi)\}$	$4 \cdot (\lvert\pi_\theta\rvert + \lvert V_\psi\rvert)$ $+BK(S+T)(L_\pi d_\pi + L_\psi d_\psi)$
Async	Aggr.	$BK\, C_{\text{gen}}(\pi_\theta) + BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)$ $+C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$\max\{D_{\text{gen}}(\pi_\theta) +$ $\max\{D_f(\pi_{\text{ref}}), D_f(R_\varphi), D_f(V_\psi)\},$ $\max\{D_{\text{tr}}(\pi_\theta), D_{\text{tr}}(V_\psi)\}\}$	$\max\{\lvert\pi_\theta\rvert + BKM_{\text{KV}} + M_{\text{sh}}(\pi_\theta),$ $\lvert\pi_{\text{ref}}\rvert, \lvert R_\varphi\rvert, \lvert V_\psi\rvert,$ $4\lvert\pi_\theta\rvert + BK(S+T)L_\pi d_\pi,$ $4\lvert V_\psi\rvert + BK(S+T)L_\psi d_\psi\}$
Combined	Aggr.	$BK\, C_{\text{gen}}(\pi_\theta) + BK(S+T)$ $\cdot[C_f^{\text{tok}}(\pi_{\text{ref}}) + C_f^{\text{tok}}(R_\varphi) + C_f^{\text{tok}}(V_\psi)$ $+C_{\text{tr}}^{\text{tok}}(\pi_\theta) + C_{\text{tr}}^{\text{tok}}(V_\psi)]$	$\max\Big\{\max\{D_{\text{gen}}^{\text{TP}}(\pi_\theta), D_f(\pi_{\text{ref}}),$ $D_f(R_\varphi), D_f(V_\psi)\},$ $\max\{D_{\text{tr}}(\pi_\theta), D_{\text{tr}}(V_\psi)\}\Big\}$	$\max\{\lvert\pi_\theta\rvert + BKM_{\text{KV}} + M_{\text{sh}}(\pi_\theta),$ $\lvert\pi_{\text{ref}}\rvert, \lvert R_\varphi\rvert, \lvert V_\psi\rvert,$ $4\lvert\pi_\theta\rvert + BK(S+T)L_\pi d_\pi,$ $4\lvert V_\psi\rvert + BK(S+T)L_\psi d_\psi\}$

TABLE 9: **Inter-model parallelism.** Global work, global depth, and peak memory per device group reported for the Generation, Assessment, and Training stages of a single RL iteration within one epoch under different inter-model parallelism techniques. Global work and depth report total FLOPs and the critical-path length per stage. Peak memory is measured per device group and taken as the maximum over all groups active within a stage; in configurations without disaggregation, all models share the same device group and execute through temporal multiplexing, where different models become active at different stages. **"TP" superscript & "SH" subscript:** A TP superscript denotes tensor-parallel execution; π_{SH} denotes the shared actor–critic configuration. **"tr" subscript:** For conciseness, for training-stage model invocations, we define $C_{\text{tr}}^{\text{tok}}(M) := C_f^{\text{tok}}(M) + C_b^{\text{tok}}(M) \approx 3C_f^{\text{tok}}(M)$ and $D_{\text{tr}}(M) := D_f(M) + D_b(M)$, because a parameter update requires a forward pass followed by backpropagation. $M_{\text{sh}}(\pi_\theta)$ denotes the persistent inference-side shadow copy of the actor required when asynchronous consumers use a bounded-stale snapshot. Also, sample-time actor log-probabilities are assumed to be computed on the fly during Generation from the same logits used for sampling; therefore they do not add a separate actor forward pass. **Baselines:** The *Baseline* configuration sequentially (i.e., one device group) executes all RL components, resulting in additive depth terms. *AC shared* uses a shared Transformer backbone for actor and critic. All models run sequentially in a single device group. *Disaggregated* execution assigns models to distinct device groups, allowing concurrent execution. The *Hybrid* configuration applies TP to Generation, TP/PP to Assessment, and ZeRO-3 to Training. All models share the same device group. *Stage F.* applies inter-stage fusion to Generation and Assessment and intra-stage fusion to Training. *Asynchronous execution* overlaps Generation and Assessment with Training by allowing bounded weight staleness. *Combined execution* combines asynchronous overlap with disaggregated model placement and hybrid intra-model parallelism, following the RLHFuse execution model. Training memory assumes the Adam optimizer without activation checkpointing, accounting for parameters, gradients, optimizer states, and activations.

and critic training groups. When asynchronous execution requires a persistent inference-side copy of a trainable model, we write this shadow-copy cost as M_{sh}.

Table 9 summarizes the resulting work, depth, and peak per-device-group memory. The most important point is that most inter-model techniques do not reduce total work: they reduce depth by replacing sequential sums with maxima, or reduce peak memory by eliminating co-residency. The main exception is shared actor–critic, which also reduces work and model state by replacing two trainable backbones with one shared backbone.

The baseline has critical path $D_G + D_A^{\Sigma} + D_T^{\Sigma}$. Disaggregation preserves work but changes within-stage composition from sums to maxima, giving $D_G + D_A^{\max} + D_T^{\max}$. Stage fusion further overlaps Generation and Assessment, resulting in $\max\{D_G, D_A^{\max}\} + D_T^{\max} \approx D_G + D_T^{\max}$ whenever generation dominates assessment. Bounded asynchrony overlaps generation/assessment of one iteration with training of another, yielding a steady-state recurrence depth $\max\{D_G + D_A^{\max}, D_T^{\max}\}$. The strongest configuration combines intra-model sharding, disaggregated placement, stage fusion, and bounded asynchrony. Its idealized depth is $\max\{D_G^{\text{TP}}, D_T^{\text{shard}}\}$, up to any unhidden assessment tail, resharding, and communication overheads. This expression makes the central limitation explicit: inter-model parallelism can hide or overlap non-generation work, but it cannot remove the autoregressive decode chain inside D_G. Reducing that term requires intra-model inference parallelism, faster decoding kernels, batching, or algorithmic changes that shorten generated trajectories.

4.7 Key Insights & Takeaways

The inter-model analysis yields several lessons.

▲ **Most inter-model schemes reduce depth, not work.** Placement, fusion, hybrid execution, and asynchrony mainly change the critical path by replacing sequential sums with maxima or by overlapping stages. The total FLOP work remains essentially unchanged. Shared actor–critic is the main exception because it removes a separate critic backbone and therefore reduces both work and trainable model-state memory.

▲ **Disaggregation is a memory and concurrency enabler, not a standalone speedup.** Moving actor, critic, reward, and reference models to separate device groups reduces co-resident memory and turns independent subcomputations into max-depth terms. However, if the pipeline still executes strict Generation → Assessment → Training barriers, disaggregation only moves idle time to different device groups. It must be combined with fusion, streaming, or asynchrony to reduce iteration depth.

▲ **Stage fusion and asynchrony attack different barriers.** Stage fusion removes within-iteration barriers, especially the barrier between long-tailed Generation and Assessment. Asynchrony removes cross-iteration barriers by allowing bounded-stale parameter reads. Fusion gives $\max\{D_G, D_A^{\max}\} + D_T^{\max}$, while asynchrony gives $\max\{D_G + D_A^{\max}, D_T^{\max}\}$. They can be combined.

▲ **The actor remains the dominant bottleneck.** Even the combined configuration cannot remove the autoregressive decode chain in $D_G = D_{\text{gen}}(\pi_\theta) = O(TD_f(\pi_\theta))$. Inter-model scheduling can hide reward, reference, critic, and training work around actor generation, but reducing D_G itself requires better inference parallelism (most notably tensor parallelism), batching, KV-cache management, shorter trajectories, or algorithmic changes.

▲ **Asynchrony trades freshness for hardware utilization and memory.** Bounded staleness can reduce steady-state depth from a sum to a max, but it introduces policy drift and may require shadow copies. Thus, the relevant control variables are not only throughput and memory, but also staleness, KL drift between behavior and current policies, PPO clip fraction, and others.

▲ **The best systems are stage- and role-aware.** A scalable RLM system should not assign one global execution mode to all models. Actor Generation, reward/reference Assessment, critic inference, actor Training, and critic Training have different bottlenecks. The strongest designs therefore combine shared or critic-free structure when appropriate, disaggregated placement for memory and concurrency, stage fusion for long-tail overlap, hybrid layouts for stage-specific efficiency, and bounded asynchrony when the learning rule tolerates stale data.

▲ **Shared actor–critic is effective under resource constraints.** Shared actor–critic architectures are particularly effective when device groups and memory are scarce and inter-model concurrency is limited thanks to sharing a single backbone between the policy and value functions.

5 Parallelism-Focused Specifications

We present a unified algorithmic and mathematical formulation of PPO (Algorithm 2) and GRPO (Algorithm 3), which share a common rollout-generation routine (Algorithm 1), together with the offline counterpart DPO (Algorithm 4). The goal is to expose where the RL-LLM pipeline admits intra- and inter-model parallelism, and this to facilitate the development of more efficient RLM architectures.

We use a notation in which boldface denotes a per-token vector and a non-bold letter with a t subscript denotes one of its per-token components. For example, $\boldsymbol{V}_n \leftarrow V_\psi(\boldsymbol{x}_n, \boldsymbol{y}_n)$ abbreviates $(V_{n,t})_{t=1}^{|\boldsymbol{y}_n|} \leftarrow (V_\psi(\boldsymbol{x}_n, \boldsymbol{y}_{n,<t}))_{t=1}^{|\boldsymbol{y}_n|}$, with all $|\boldsymbol{y}_n|$ entries produced by a single teacher-forced forward pass. A plain letter without a t subscript is a scalar (e.g., $r_n \leftarrow R_\varphi(\boldsymbol{x}_n, \boldsymbol{y}_n)$). The unit basis vector $\boldsymbol{e}_t$ has 1 at position t and 0 elsewhere. The all-ones vector $\mathbf{1}$ has 1 at every position, with length inferred from context.

If the actor and critic share a backbone, the PPO actor/-critic two invocations should instead be read as a single shared-model invocation with two heads. For DPO, the reference path may be omitted from the online training loop if $\boldsymbol{\ell}_{\text{ref},w}^{(n)}$ and $\boldsymbol{\ell}_{\text{ref},l}^{(n)}$ are precomputed and stored with the preference dataset.

Background colors indicate model roles. The label **[inter-model concurrency]** marks following fork–join (colored) regions in which distinct model invocations have no data dependency and may run concurrently when placed on separate device groups. The label **[DP]** (data parallelism) marks independent sample-level work, such as prompts, completions, rollouts, or preference pairs. The tags **[TP]**, **[PP]**, **[SP]**, **[CP]**, and **[EP]** indicate intra-model parallel strategies that can be used for the corresponding invocation. Boldface (e.g., **TP**) indicates a usual use, non-bold font (e.g.,

SP) is a possible but not a necessarily common use case. Each tag describes the next following line or code block.

TP/PP are mainly capacity or single-invocation latency tools, SP is mainly useful for trainable teacher-forced passes with large activation memory, CP is useful for long-context invocations, and EP applies only to MoE models.

Algorithm 1: A common part of Generation, used by PPO and GRPO

Input: Batch of prompts $X = \{\boldsymbol{x}_b\}_{b=1}^{B}$, candidates/prompt K.
Output: Set $\mathcal{B}_{\text{roll}}$ of generated samples paired with per-token rollout log-probabilities.

Initialize $\mathcal{B}_{\text{roll}} \leftarrow \emptyset$;
[DP] foreach *prompt* $\boldsymbol{x}_b \in X$ **do**
  [DP] for *candidate* $i = 1$ **to** K **do**
    Initialize $\boldsymbol{y}_b^{(i)} \leftarrow []$ and $\boldsymbol{\ell}_{\text{old},b}^{(i)} \leftarrow []$;
    actor path
    for $t = 1, 2, \ldots$ *until EOS or max length* **do**
      [TP, PP, SP, CP, EP]
      Sample next token $y_{b,t}^{(i)} \sim \pi_\theta\left(\cdot \mid \boldsymbol{x}_b, \boldsymbol{y}_{b,<t}^{(i)}\right)$ via stochastic decoding;
      Actor log-prob $\ell_{\text{old},b,t}^{(i)} \leftarrow \log \pi_\theta\left(y_{b,t}^{(i)} \mid \boldsymbol{x}_b, \boldsymbol{y}_{b,<t}^{(i)}\right)$;
      Append $y_{b,t}^{(i)}$ to $\boldsymbol{y}_b^{(i)}$ and $\ell_{\text{old},b,t}^{(i)}$ to $\boldsymbol{\ell}_{\text{old},b}^{(i)}$;
    end
    Append $\left(\boldsymbol{x}_b, \boldsymbol{y}_b^{(i)}, \boldsymbol{\ell}_{\text{old},b}^{(i)}\right)$ to $\mathcal{B}_{\text{roll}}$;
  end
end

The intra-model annotations are attached only to operations that invoke a Transformer model or backpropagate through one. For independent samples, prompts, candidates, rollouts, or preference pairs, **DP** is the natural default because these units have no semantic dependency and can be assigned to different replicas with only later reductions for losses, statistics, or gradients. For model invocations, TP and PP are marked whenever the actor, critic, reward, or reference model may need to be sharded to fit memory or reduce single-invocation latency; however, they are conditional because their collectives, activation transfers, and pipeline bubbles can hurt throughput when the model already fits on one device. CP is marked on long-context forward or training invocations, including autoregressive decode, because KV/attention memory of RLM rollouts may exceed single-device capacity. SP is marked only on trainable teacher-forced forward/backward paths, not on ordinary forward-only Assessment or decode-time Generation, because its main benefit is reducing stored activation memory during Training. EP is marked only as an MoE-dependent option: it applies if the corresponding actor, critic, reward, or reference model contains routed experts, where it reduces per-device expert storage and compute but introduces routing and all-to-all communication.

6 Analysis of Existing Models & Designs

We also analyze existing models and frameworks.

6.1 Reasoning Language Models

We compare representative post-trained LLMs and RLMs in Table 10. We classify a model as a *general aligned LLM* if post-training primarily improves instruction-following, helpfulness, safety, or general assistant behavior, even when the model can solve reasoning tasks. We classify a model as an *RLM* if its training or inference pipeline explicitly targets

Algorithm 2: Algorithmic and Mathematical Specification of PPO

Input: GAE discount γ and smoothing λ, KL coefficient β, PPO clip range ε, entropy coefficient c_H; set of prompts X, prompt batch size B, candidates/prompt K, training epochs E_{PPO}.
Output: Trained parameters (θ, ψ).

[DP] foreach *prompt batch* $\{\boldsymbol{x}_b\}_{b=1}^{B} \subset X$ **do**
  Generation (Online)
  Generate completions $\mathcal{B}_{\text{roll}} \leftarrow \left\{\left(\boldsymbol{x}_b, \boldsymbol{y}_b^{(i)}, \boldsymbol{\ell}_{\text{old},b}^{(i)}\right)\right\}_{b=1,i=1}^{B,K}$ via **Algorithm 1**;
  Flatten $\mathcal{B}_{\text{roll}} \leftarrow \left\{\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\ell}_{\text{old}}^{(n)}\right)\right\}_{n=1}^{BK}$;
  Assessment
  [DP] foreach $\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\ell}_{\text{old}}^{(n)}\right) \in \mathcal{B}_{\text{roll}}$ **do**
    [inter-model concurrency]
    reward path
    [TP, PP, SP, CP, EP]
    Compute scalar reward $s_n \leftarrow R_\varphi\left(\boldsymbol{x}_n, \boldsymbol{y}_n\right)$;
    critic path
    [TP, PP, SP, CP, EP]
    Compute values $\boldsymbol{V}_n \leftarrow V_\psi(\boldsymbol{x}_n, \boldsymbol{y}_n)$;
    EOS terminal value $V_{n,|\boldsymbol{y}_n|+1} \leftarrow 0$;
    reference path
    [TP, PP, SP, CP, EP]
    Compute reference log-probs $\boldsymbol{\ell}_{\text{ref}}^{(n)} \leftarrow \log \pi_{\text{ref}}(\boldsymbol{y}_n \mid \boldsymbol{x}_n)$;
    Per-token KL-shaped rewards $\boldsymbol{r}_n \leftarrow -\beta\left(\boldsymbol{\ell}_{\text{old}}^{(n)} - \boldsymbol{\ell}_{\text{ref}}^{(n)}\right) + s_n\, \boldsymbol{e}_{|\boldsymbol{y}_n|}$ *(KL penalty enters via reward and propagates through GAE; no KL term in PPO loss)*
    Initialize $\boldsymbol{A}_n \leftarrow []$, $\hat{\boldsymbol{G}}_n \leftarrow []$, and EOS terminal advantage $A_{n,|\boldsymbol{y}_n|+1} \leftarrow 0$;
    for $t = |\boldsymbol{y}_n|$ **to** 1 **do**
      Temporal difference residual $\delta_{n,t} \leftarrow r_{n,t} + \gamma V_{n,t+1} - V_{n,t}$;
      Compute advantage (GAE) $A_{n,t} \leftarrow \delta_{n,t} + \gamma\lambda A_{n,t+1}$;
      Compute discounted return $\hat{G}_{n,t} \leftarrow V_{n,t} + A_{n,t}$;
      Prepend $A_{n,t}$ to $\boldsymbol{A}_n$ and $\hat{G}_{n,t}$ to $\hat{\boldsymbol{G}}_n$;
    end
    Augment rollout n in $\mathcal{B}_{\text{roll}}$ with $\left(\boldsymbol{A}_n, \hat{\boldsymbol{G}}_n\right)$;
  end
  Training via PPO
  for $e = 1$ **to** E_{PPO} **do**
    Shuffle $\mathcal{B}_{\text{roll}}$ and partition into minibatches $\{\mathcal{B}\}$;
    [DP] foreach *minibatch* $\mathcal{B}$ **do**
      [DP] foreach *rollout* $n \in \mathcal{B}$ **do**
        [inter-model concurrency]
        actor path
        [TP, PP, SP, CP, EP]
        Current log-probs $\boldsymbol{\ell}_\theta^{(n)} \leftarrow \log \pi_\theta(\boldsymbol{y}_n \mid \boldsymbol{x}_n)$;
        Importance ratios $\boldsymbol{\rho}_n \leftarrow \exp\left(\boldsymbol{\ell}_\theta^{(n)} - \boldsymbol{\ell}_{\text{old}}^{(n)}\right)$;
        critic path
        [TP, PP, SP, CP, EP]
        Current values $\boldsymbol{V}'_n \leftarrow V_\psi(\boldsymbol{x}_n, \boldsymbol{y}_n)$;
      end
      Number of tokens in batch $Z \leftarrow \sum_{n\in\mathcal{B}} |\boldsymbol{y}_n|$;
      [inter-model concurrency]
      actor path
      Compute PPO loss: $\mathcal{L}_{\text{PPO}} = -\frac{1}{Z}\sum_{n\in\mathcal{B}}\sum_{t=1}^{|\boldsymbol{y}_n|} \min(\rho_{n,t}A_{n,t}, \text{clip}(\rho_{n,t}, 1\pm\varepsilon)\, A_{n,t})$;
      (Optional) Entropy bonus; $H(\cdot)$ is the actor's per-token Shannon entropy ($c_H = 0$ disables) $\mathcal{L}_H = -\frac{1}{Z}\sum_{n\in\mathcal{B}}\sum_{t=1}^{|\boldsymbol{y}_n|} H\left(\pi_\theta\left(\cdot \mid \boldsymbol{x}_n, \boldsymbol{y}_{n,<t}\right)\right)$;
      Define actor objective: $\mathcal{L}_{\text{PPO}} \leftarrow \mathcal{L}_{\text{PPO}} + c_H\, \mathcal{L}_H$;
      [TP, PP, SP, CP, EP]
      Backpropagate & update on θ to minimize $\mathcal{L}_{\text{PPO}}$;
      critic path
      Compute critic loss: $\mathcal{L}_{\text{critic}} = \frac{1}{Z}\sum_{n\in\mathcal{B}}\sum_{t=1}^{|\boldsymbol{y}_n|}(V'_{n,t} - \hat{G}_{n,t})^2$;
      [TP, PP, SP, CP, EP]
      Backpropagate & update on ψ to minimize $\mathcal{L}_{\text{critic}}$;
    end
  end
end

Algorithm 3: Algorithmic and Mathematical Specification of GRPO

Input: KL coefficient β, GRPO clip range ε; set of prompts X, prompt batch size B, candidates/prompt K, training epochs E_{GRPO}.
Output: Trained parameters θ.

foreach *prompt batch* $\{\boldsymbol{x}_b\}_{b=1}^{B} \subset X$ **do**
  ——— **Generation (Online)** ———
  Generate completions $\mathcal{B}_{\text{roll}} \leftarrow \left\{\left(\boldsymbol{x}_b, \boldsymbol{y}_b^{(i)}, \boldsymbol{\ell}_{\text{old},b}^{(i)}\right)\right\}_{b=1,i=1}^{B,K}$ via **Algorithm 1**;
  ——— **Assessment** ———
  [DP] for *prompt* $b = 1$ **to** B **do**
    [DP] for *candidate* $i = 1$ **to** K **do**
      [inter-model concurrency]
      reward path
      [TP, PP, SP, CP, EP]
      Compute scalar reward $r_b^{(i)} \leftarrow R_\varphi\left(\boldsymbol{x}_b, \boldsymbol{y}_b^{(i)}\right)$;
      reference path
      [TP, PP, SP, CP, EP]
      Compute reference log-probs $\boldsymbol{\ell}_{\text{ref},b}^{(i)} \leftarrow \log \pi_{\text{ref}}\left(\boldsymbol{y}_b^{(i)} \mid \boldsymbol{x}_b\right)$;
    end
    Group mean $\mu_b \leftarrow \frac{1}{K}\sum_{i=1}^{K} r_b^{(i)}$;
    Group std $\sigma_b \leftarrow \left(\frac{1}{K}\sum_{i=1}^{K}\left(r_b^{(i)} - \mu_b\right)^2\right)^{1/2}$;
    [DP] for *candidate* $i = 1$ **to** K **do**
      Per-token advantage (broadcast across tokens) $\boldsymbol{A}_b^{(i)} \leftarrow \frac{r_b^{(i)} - \mu_b}{\sigma_b + 10^{-8}}\,\mathbf{1}$;
      Augment rollout (b, i) in $\mathcal{B}_{\text{roll}}$ with $\left(\boldsymbol{A}_b^{(i)}, \boldsymbol{\ell}_{\text{ref},b}^{(i)}\right)$;
    end
  end
  Flatten $\mathcal{B}_{\text{roll}} \leftarrow \left\{\left(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{\ell}_{\text{old}}^{(n)}, \boldsymbol{A}_n, \boldsymbol{\ell}_{\text{ref}}^{(n)}\right)\right\}_{n=1}^{BK}$;
  ——— **Training via GRPO** ———
  for $e = 1$ **to** E_{GRPO} **do**
    Shuffle $\mathcal{B}_{\text{roll}}$ and partition into minibatches $\{\mathcal{B}\}$;
    [DP] foreach *minibatch* $\mathcal{B}$ **do**
      [DP] foreach *rollout* $n \in \mathcal{B}$ **do**
        actor path
        [TP, PP, SP, CP, EP]
        Current log-probs $\boldsymbol{\ell}_\theta^{(n)} \leftarrow \log \pi_\theta(\boldsymbol{y}_n \mid \boldsymbol{x}_n)$;
        Importance ratios $\boldsymbol{\rho}_n \leftarrow \exp\left(\boldsymbol{\ell}_\theta^{(n)} - \boldsymbol{\ell}_{\text{old}}^{(n)}\right)$;
        Per-token KL approximation $\boldsymbol{d}_n \leftarrow \exp\left(\boldsymbol{\ell}_{\text{ref}}^{(n)} - \boldsymbol{\ell}_\theta^{(n)}\right) - \left(\boldsymbol{\ell}_{\text{ref}}^{(n)} - \boldsymbol{\ell}_\theta^{(n)}\right) - 1$;
      end
      Number of tokens in batch $Z \leftarrow \sum_{n\in\mathcal{B}} |\boldsymbol{y}_n|$;
      actor path
      Compute GRPO loss: $\mathcal{L}_{\text{GRPO}} = -\frac{1}{Z}\sum_{n\in\mathcal{B}}\sum_{t=1}^{|\boldsymbol{y}_n|}\Big[\min(\rho_{n,t}A_{n,t}, \text{clip}(\rho_{n,t}, 1\pm\varepsilon)\,A_{n,t}) - \beta\, d_{n,t}\Big]$;
      [TP, PP, SP, CP, EP]
      Backpropagate & update on θ to minimize $\mathcal{L}_{\text{GRPO}}$;
    end
  end
end

Algorithm 4: Algorithmic and Mathematical Specification of DPO

Input: KL coefficient β, training epochs E_{DPO}; preference dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}_n, \boldsymbol{y}_w^{(n)}, \boldsymbol{y}_l^{(n)}\right)\right\}_{n=1}^{N}$.
Output: Trained parameters θ.

——— **Training via DPO** ———
for $e = 1$ **to** E_{DPO} **do**
  Shuffle $\mathcal{D}$ and partition into minibatches $\{\mathcal{B}\}$;
  [DP] foreach *minibatch* $\mathcal{B}$ **do**
    [DP] foreach $\left(\boldsymbol{x}_n, \boldsymbol{y}_w^{(n)}, \boldsymbol{y}_l^{(n)}\right) \in \mathcal{B}$ **do**
      [inter-model concurrency]
      actor path
      [TP, PP, SP, CP, EP] Actor log-probs:
      $\boldsymbol{\ell}_{\theta,w}^{(n)} \leftarrow \log \pi_\theta\left(\boldsymbol{y}_w^{(n)} \mid \boldsymbol{x}_n\right)$ *(preferred completion)*;
      $\boldsymbol{\ell}_{\theta,l}^{(n)} \leftarrow \log \pi_\theta\left(\boldsymbol{y}_l^{(n)} \mid \boldsymbol{x}_n\right)$ *(dispreferred completion)*;
      reference path
      [TP, PP, SP, CP, EP] Reference log-probs:
      $\boldsymbol{\ell}_{\text{ref},w}^{(n)} \leftarrow \log \pi_{\text{ref}}\left(\boldsymbol{y}_w^{(n)} \mid \boldsymbol{x}_n\right)$ *(preferred completion)*;
      $\boldsymbol{\ell}_{\text{ref},l}^{(n)} \leftarrow \log \pi_{\text{ref}}\left(\boldsymbol{y}_l^{(n)} \mid \boldsymbol{x}_n\right)$ *(dispreferred completion)*;
      Trajectory-level log-ratio difference: $h_n \leftarrow \mathbf{1}^\top\left(\boldsymbol{\ell}_{\theta,w}^{(n)} - \boldsymbol{\ell}_{\text{ref},w}^{(n)}\right) - \mathbf{1}^\top\left(\boldsymbol{\ell}_{\theta,l}^{(n)} - \boldsymbol{\ell}_{\text{ref},l}^{(n)}\right)$;
    end
    actor path
    Compute DPO loss, where $\sigma(\cdot)$ is the sigmoid: $\mathcal{L}_{\text{DPO}} = -\frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \log \sigma(\beta\, h_n)$;
    [TP, PP, SP, CP, EP]
    Backpropagate & update on θ to minimize $\mathcal{L}_{\text{DPO}}$;
  end
end

reasoning trajectories, verifiable rewards, long CoT, search, tool use, or controllable test-time computation. This distinction is often blurred in practice: recent systems increasingly unify fast response modes and slow reasoning modes inside a single routed or hybrid model family.

Overall, the comparison shows a shift from reference-driven alignment as *preference optimization* to alignment as *reasoning-time computation*. Early RL-enhanced LLMs primarily used PPO-like RLHF or more lightweight DPO-style preference optimization to improve instruction following, safety, and general assistant quality, whereas RLMs increasingly optimize or allocate compute to long-CoT trajectories, RLVR, search, tools, agentic environments, or hybrid fast/thinking modes. The training signal is often more predictive of reasoning gains than parameter count alone: exact-answer rewards, unit tests, process/preference models, MCTS, or environment feedback can make smaller specialized models competitive on narrow math/code tasks, while frontier systems retain broader coverage through scale, data diversity, and tool integration. Architecturally, sparse MoE has become central for open frontier models because it increases total capacity at moderate active-parameter cost, but it shifts systems pressure toward expert parallelism, routing balance, all-to-all communication, and memory placement; dense models remain simpler and more predictable for deployment.

6.2 Training & Inference Frameworks for RLMs

Table 11 shows that modern RL-LLM frameworks differ primarily by how aggressively they separate, specialize, and overlap the rollout and training workloads. In practice, wall-clock time is often dominated by rollout/generation under long-output reasoning settings, while gradient updates are comparatively more parallelizable; consequently, the highest-throughput frameworks either (a) maximize rollout throughput via inference engines and stage-specific parallelism, or (b) overlap generation with evaluation and training via stage fusion or asynchronous execution.

Library-centric stacks such as TRL [231], DeepSpeed-Chat [261], and ColossalChat [267] are easiest to use and rely mainly on DP/ZeRO/FSDP-style sharding [201, 277] plus optional fast rollout engines such as vLLM [133], but they expose limited inter-model scheduling. Orchestrator-centric stacks such as OpenRLHF [117], HybridFlow/verl [217], SkyRL [181], NeMo RL [184, 214], slime [287], ROLL [240]

Model / family	Access	Scale / arch.	Post-training signal	Selected system details
General aligned large language models (LLMs)				
InstructGPT [192]	Closed	1.3B, 6B, 175B dense	PPO-like (SFT + RM + PPO)	Canonical actor–reward–reference PPO loop.
GPT-4 [4]	Closed	Undisclosed	PPO-like (rule-based rewards)	Frontier-scale alignment; limited public systems detail.
Gemini 1.x [12]	Closed	Undisclosed	PPO-like (iterative RM refinement)	Repeated RM/RL cycles increase assessment cost.
Claude 3 [13, 21]	Closed	Undisclosed	PPO-like (RLAIF / Constitutional AI)	AI-generated preference labels from written principles.
Reka [191]	Mixed	7B, 21B dense	PPO-like (multi-round RLHF)	Standard PPO-style alignment at moderate scale.
InternLM2 [59]	Open	1.8B, 7B, 20B dense	PPO-like (conditional RM)	Multiple domain-specific reward heads.
Zephyr [114, 229]	Open	MoE variant reported	DPO-like (ORPO)	No explicit reward model or PPO loop.
Phi-3 [2]	Open	3.8B, 7B, 14B dense	DPO-like	SFT-like alignment cost.
Phi-4 [3]	Open	7B, 14B dense	DPO-like (RLAIF data)	Preference tuning over AI/human feedback data.
ChatGLM [284]	Open	6B, 9B dense	Mixed (PPO-like vs. DPO-like)	Controlled comparison of online vs. DPO training.
Gemma 2 [204]	Open	2B, 9B, 27B dense	PPO-like (Bradley–Terry RM)	Reward-model alignment.
Starling-7B [285]	Open	7B dense	PPO-like (RLAIF; Plackett–Luce RM)	Ranking-based RM; partial model updates reduce memory.
Hermes 3 [226]	Open	8B, 70B, 405B dense	DPO-like (LoRA-DPO)	Only LoRA/adapters are trained; optimizer memory is small.
Athene-70B [91]	Open	70B dense	PPO-like (details limited)	A model with preference/safety tuning.
Llama 3 & 3.1 [104]	Open	8B, 70B, 405B dense	DPO-like (RM + rejection sampling)	—
Qwen2 & Qwen2.5 [258]	Open	0.5B–72B dense; 57B MoE (14B active)	DPO-like (offline + online refresh)	New preference pairs are created from online model samples.
Nemotron-4 340B [5]	Open	340B-class model	DPO-like (RPO)	Quality-aware preference optimization without PPO.
DeepSeek-V3 [156]	Open	671B MoE ($\sim$37B active)	Hybrid alignment	MoE, MLA, FP8, MTP, and DualPipe.
Llama 4 Scout [171]	Open	$\sim$109B MoE (17B active)	Hybrid alignment	Long context.
Llama 4 Maverick [171]	Open	$\sim$400B MoE (17B active)	Hybrid alignment	MoE and multimodality.
Reasoning language models (RLMs)				
OpenAI o1 [186]	Closed	Undisclosed	Hybrid RL (CoT)	"Reasoning tokens" become inference-time cost.
OpenAI o3 [188]	Closed	Undisclosed	Hybrid RL (CoT + tools)	Training-time RL and tool-using inference both dominate.
OpenAI o4-mini [188]	Closed	Undisclosed	Hybrid RL (CoT + tools)	Lower-cost reasoning model; latency/quality trade-off.
GPT-5 [220]	Closed	Undisclosed	Hybrid (fast/thinking router)	Runtime routes between fast and reasoning modes.
GPT-5.4 Thinking [190]	Closed	Undisclosed	Hybrid (coding + agents)	Agentic workflow.
GPT-5.5 Thinking [189]	Closed	Undisclosed	Hybrid (agentic reasoning)	Higher autonomy increases orchestration cost.
Claude 3.7 Sonnet [14]	Closed	Undisclosed	PPO-like/RLAIF + thinking mode	Same model supports quick and extended reasoning.
Claude 4 family [15]	Closed	Undisclosed	PPO-like/RLAIF + thinking mode	Reasoning budget is a user/system cost knob.
Gemini 2.5 Pro [71]	Closed	Sparse MoE; multimodal	Hybrid RL (thinking mode)	Long context.
Gemini 2.5 Flash [71]	Closed	Sparse MoE with a lower-latency variant	Hybrid RL (low-cost thinking)	Cheaper reasoning.
Gemini 2.5 Deep Think [102]	Closed	Gemini 2.5 family	Search-like, parallel reasoning	Multiple reasoning paths increase per-query compute.
DeepSeek-R1 [107]	Open	671B MoE / $\sim$37B active	GRPO-like (RLVR; no critic)	Removes V_ψ; generation remains bottleneck.
DAPO [265]	Open	32B dense	GRPO-like (DAPO; no critic)	Critic-free long-CoT RL.
Qwen3 [257]	Open	Dense/MoE, 0.6B–235B	Hybrid (thinking/non-thinking)	Variable reasoning length.
Kimi k1.5 [86]	Closed	Undisclosed; long-context multimodal	DPO-like/custom RL (KL-DPO)	Long trajectories dominate training and serving cost.
Kimi K2 [19]	Open	1T MoE (32B active)	Agentic RL-like (tool environments)	—
Kimi K2.5 [18]	Open	MoE; multimodal agentic extension	Agentic RL-like (visual tools)	Adds multimodal environment overheads.
Phi-4-reasoning [1]	Open	14B dense	DPO-like (RLVR)	Small strong STEM model at low serving cost.
rStar-Math [106]	Open	7B-scale policy/reward components	Search/RLVR-like (MCTS + PRM)	Search improves math but increases inference compute.
Grok 4 [249]	Closed	Undisclosed	Hybrid RL	—
Grok 4 Heavy [249]	Closed	Undisclosed; multi-agent variant	Agentic / parallel reasoning	Parallel agents increase inference compute.

TABLE 10: **A comparison of representative general aligned LLMs and RLMs.** The upper block lists general aligned LLMs, which primarily target instruction following, safety, preference alignment, and broad assistant quality. The lower block lists RLMs, which explicitly target reasoning trajectories, long-CoT, RLVR/verifiable rewards, search, tools, multi-agent inference, or controllable test-time computation. The "Post-training signal" column follows the taxonomy used in this paper: PPO-like methods use reward-model- or RLAIF-based policy optimization; GRPO-like methods use critic-free grouped RL/RLVR; DPO-like methods use direct preference optimization; Hybrid denotes mixed or undisclosed pipelines; Search/Agentic denotes explicit search, tools, environments, or multi-agent reasoning. Proprietary details are based on public reports and should be treated as approximate.

Framework	Intra-Model Parallelism								Inter-Model Parallelism					Remarks
	DP	Z1	Z2	Z3	TP	CP	EP	PP	AC Share	Dis-agg.	Stage Fusion	Hybrid	Async	
TRL [231]	●	◐	●	●	●	●	✖	✖	●	●	◐	●	●	Accelerate; TP/vLLM mostly rollout; CP via FSDP2/Ulysses
ColossalChat [267]	●	●	●	●	●	●	◐	●	✖	✖	✖	✖	✖	Colossal-AI HybridParallel + Gemini; TP/PP/SP, Z1/2 and Z3
DeepSpeed-Chat [261]	●	●	●	●	●	✖	✖	✖	✖	✖	✖	●	✖	Train ZeRO-2/3; Hybrid Engine uses TP mainly for rollout
OpenRLHF [117]	●	◐	●	●	●	●	✖	●	✖	●	◐	●	●	Train: ZeRO-3/AutoTP; rollout: vLLM TP/PP; RingAttention
HybridFlow / verl [217]	●	◐	◐	●	●	●	●	●	✖	●	◐	●	●	Train: FSDP or Megatron; rollout: vLLM/SGLang/TensorRT-LLM
NeMo RL [184, 214]	●	◐	◐	●	●	●	●	●	✖	●	✖	●	●	Train: DTensor or Megatron; rollout: vLLM/Megatron/SGLang
ReaL [169]	●	◐	✖	✖	●	✖	✖	●	✖	●	◐	●	✖	Per-stage DP/TP/PP/SP plans; ZeRO backend not stage-enumerated
RLHFuse [282]	◐	◐	✖	✖	◐	✖	✖	●	✖	◐	●	◐	◐	Scheduling layer; inter-stage sample fusion + intra-stage fused PP
FlexRLHF [251]	●	◐	◐	◐	◐	✖	✖	◐	●	●	◐	●	✖	DeepSpeed-based; AC-share/nonshare; interleaving/disaggregated layouts
AReaL [92]	●	◐	◐	●	●	●	●	●	✖	●	◐	●	●	Train: Megatron/FSDP/Archon; infer: SGLang default, vLLM optional
MindSpeed RL [89]	●	◐	◐	◐	●	●	●	●	✖	●	◐	●	●	Ascend stack; co-card/decoupled deploy, repartition, partial rollout
slime [287]	●	◐	◐	◐	●	●	●	●	✖	●	◐	●	●	Megatron train + SGLang rollout
ROLL [240]	●	◐	●	●	●	●	●	●	✖	●	◐	●	●	train: DeepSpeed/FSDP2/Megatron; infer: vLLM/SGLang/Megatron
StreamRL [281]	outside system focus								✖	●	●	●	●	Disaggregation-first; stream generation; full overlap in async mode
AsyncFlow [111]	outside system focus								✖	●	◐	●	●	Distributed data/param streaming; producer–consumer async
Laminar [216]	outside system focus								✖	●	◐	●	●	Relay-worker parameter service; trajectory-level async repack
StaleFlow [145]	outside system focus								✖	●	◐	●	●	Data+parameter servers; explicit global staleness control

TABLE 11: **Comparison of public training / inference frameworks for LLM and RLM post-training under the systems taxonomy. Intra-Model Parallelism**: DP=data parallel; Z1/Z2/Z3=ZeRO/FSDP-like sharding levels (optimizer / optimizer+gradients / parameters+gradients+optimizer; FSDP2 is counted under Z3); TP=tensor parallel; CP=context or long-sequence parallelism; EP=expert parallelism; PP=pipeline parallelism. **Inter-Model Parallelism**: AC Share=shared actor/critic parameters or value-head mode; Disagg.=separate services or device groups for rollout, reward, or training; Stage Fusion=explicit overlap or fusion across normally sequential RL stages or subtasks; Hybrid=stage-specific backends/layouts or runtime resharding/refit; Async=asynchronous or bounded-stale execution. **Symbols**: ● documented support; ◐ indirect, partial, backend-specific, or train-vs-rollout-specific support; ✖ no documented public support; ? undocumented or unclear in public materials.

and ReaL [169] treat RLHF as a multi-role dataflow. They provide explicit placement and resource management for actor, rollout, reward, reference, critic, and data-buffer components, and commonly combine training backends such as DeepSpeed ZeRO, PyTorch FSDP/FSDP2, or Megatron-Core that support TP/PP/CP/DP/EP [125, 160, 176, 219], with rollout engines such as vLLM, SGLang, or TensorRT-LLM [133, 185, 278]. These systems often support heterogeneous parallelism across roles, weight synchronization, and resharding between training and rollout with explicit weight synchronization or resharding [169, 217]. Scheduling-centric systems such as RLHFuse [282], PipeRLHF [262], StreamRL [281], AReaL [92], AsyncFlow [111], Laminar [216], StaleFlow [145], and MindSpeed RL [89] are best viewed as emphasizing scheduling policies rather than defining a disjoint framework class. In practice, many orchestrator-centric frameworks can also manually schedule workloads or incorporate similar optimizations. These systems target the dominant bottlenecks of long-tailed autoregressive rollout by improving utilization by stage fusion [282], streaming and disaggregation [281], dynamic load balancing [111], and bounded-stale asynchronous execution [92, 145, 180], at the cost of more complex queues, weight-version management, and convergence validation.

Practically, small experiments should favor library-centric stacks; memory-bounded large-model runs should favor mature state-sharding backends such as FSDP/ZeRO or Megatron [201, 219, 277]; RL post-training workloads whose wall-clock time is dominated by rollout/generation should favor disaggregated vLLM/SGLang-style rollout [133, 278, 281]; and frontier-scale reasoning RL increasingly requires async or streaming systems that explicitly control staleness and generation-length skew [92, 145, 216].

Across frameworks, data parallelism remains a common outer abstraction, but modern RL post-training stacks increasingly compose multiple forms of parallelism across roles and stages. FSDP/ZeRO-style state sharding and Megatron-FSDP occupy the same broad design space for sharding optimizer states, gradients, and parameters, while Megatron-style backends can further combine DP with TP, PP, CP, and EP when model size, sequence length, or MoE require it. Thus, the main systems issue is not a fixed choice between FSDP/ZeRO and Megatron, but how the framework places heterogeneous roles, synchronizes or reshards weights between the trainer and rollout workers, and balances optimizer-state memory, communication, KV-cache pressure, and variable rollout lengths. In this sense, orchestrator-centric and scheduling-centric systems are better viewed as different emphases within the same design space: the former provides the multi-role execution substrate, while the latter emphasizes utilization policies for long-tailed generation workloads.

7 Research Opportunities

We briefly outline potential opportunities for future research in the performance aspects of RLMs.

Resource-aware test-time compute. RLMs increasingly trade inference-time compute for accuracy through long-CoT generation, self-consistency, search, tool use, and other strategies, exploring different scaling regimes and trade-offs [71, 115, 115, 130, 188, 221, 225, 247, 270]. Future systems could treat reasoning as a constrained optimization problem: maximize expected utility subject to token, latency, memory, tool-call, and energy budgets. This opens the door to interesting research into making reasoning budget a first-class scheduling variable, and into online predictors of task difficulty, verifier value, branch utility, and stopping time.

Efficient RLM execution structures beyond chains. RLM reasoning is usually represented as a linear token sequence. Prior works such as Tree of Thoughts, Graph of Thoughts, and Hypergraph-of-Thoughts work shows

that nonlinear reasoning structures can improve search and aggregation in-context [26, 33, 45, 179, 259, 260]. Some efforts to distill this behavior into weights have been made into this direction by harnessing aggregation during fine-tuning [8, 150, 151, 275]. Efficient and effective integration of such structures and ideas in the RL execution pipeline is an interesting novel direction in the science of RLM performance and parallel design.

Automatic selection of parallelization schemes. One concrete idea of how to enhance the parallel design of RLM pipelines, is to provide an effective way of *automating* the intra-parallelization of each model invocation based on the available hardware and cluster conditions. There have been works into this direction, for example AutoDDL [65] and PyTorch/XLA SPMD [127], but they focus on an individual Transformer invocations and not whole pipelines [65, 127].

Automatic derivation of reasoning topologies and schedules. In the RLM designs that harness explicit structures such as MCTS, most methods still use hand-designed chains, trees, beams, and others [26]. A potential direction is to automatically derive both the reasoning topology and the execution schedule from aspects such as task difficulty, model uncertainty, verifier availability, hardware availability and performance properties, and others. Dynamic graph representations and algorithms could offer useful templates for such planners in terms of performance-focused aspects such as algorithm design [32], effective task graph decompositions and partitioning [58, 101], scheduling [27, 77], and others. Example questions to pursuit would be how many branches to explore, when to prune a branch, when to call a verifier, when to merge thoughts, how to map the resulting DAG to device groups, and others.

RLVR beyond final-answer rewards. RLVR scales reasoning post-training through exact-answer checks, unit tests, and task-specific verifiers, but its mechanisms remain incompletely understood; for example, weak or spurious rewards can sometimes still improve reasoning, and outcomes are model-family dependent [212]. Future work could focus on developing reliable and efficient process-level rewards, building upon existing work [69, 129, 149, 165, 241, 271, 273], and potentially even extend them to consider the *structural* aspects of the reasoning process beyond chains. Since process rewards create many assessment nodes, their usefulness will depend on performance-oriented policies that involve caching, batching, placement, and overlap with generation.

Harnessing data analytics for enhanced reasoning. A rich landscape of research opportunities exist at the overlap of reasoning and data analytics. One could delegate certain parts of reasoning tasks that are hard to instill into model weights to existing algorithms and frameworks; examples are graph mining and analytics [16, 36, 46, 99, 193]. One could also harness graph learning for reasoning supervision, i.e., graph neural networks (GNNs) [50, 131, 248, 274, 283] and broader graph representation learning methods [28, 57, 62, 100, 110] may help score branches, help constructing process rewards, or predict useful tool calls, based on learning useful reasoning patterns. The systems challenge is to integrate such designs efficiently [40, 49] without adding bottlenecks to the Generation–Assessment–Training loop or broader MCTS.

Enhancing data analytics pipelines. On the other hand, data analytics frameworks [31, 32, 52, 54, 95, 109, 137, 236] as well as databases [10, 11, 33, 35, 41, 78] could also integrate reasoning LLMs and agents for more effective data processing, especially at the user–framework interface. Different designs have already been proposed to integrate LLMs into analytics architectures [42, 55, 64, 67, 85, 139, 199, 244, 269]. This includes document analysis (e.g., Aryn [9], DocETL [211], Palimpzest [158], PalimpChat [159]), tabular data (e.g., InsightPilot [164], CoddLLM [272], Pneuma [23], Chat2data [276], LOTUS [194], DB-GPT [254]), video processing [167], and others [80]. Here, harnessing RLMs and their efficient integration into such frameworks is an interesting novel research direction.

Asynchronous and stale-data RL. Online RLM training is limited by long-tailed autoregressive generation. Asynchronous systems such as AReaL, AsyncFlow, Laminar, StreamRL, and StaleFlow show that bounded staleness and streaming rollouts can improve utilization [92, 111, 145, 216, 281]. The open problem is to characterize in a more rigorous way when and to what degree involve staleness – for example, when stale rollouts remain useful, how to correct policy drift, and how to trade staleness against throughput without degrading reasoning quality.

Retrieval, tools, and external state as operators. Effective and efficient integration of tools into the reasoning process is another interesting research opportunity [161, 215]. Various tool calls (e.g., retrieval [44, 93, 126, 142, 206], simulators, verifiers [47, 105], agent calls [29, 48, 70, 119, 245], etc.) could be modeled as operators in the same execution graph as actor generation, reward evaluation, and training. Retrieval-augmented and graph-based LLM methods such as Topologies of Reasoning provide starting points for such integration [45].

Harnessing HPC architectures as well as emerging hardware. RLMs stress hardware through aspects such as – among others – its long outputs [177], large KV caches [144, 157, 218], repeated verifier calls, tool use, and many samples per prompt. Beyond established forms of parallelism, promising directions include harnessing emerging hardware such as processing-in-memory [7, 43, 97, 174, 209], RDMA and SmartNICs [38, 39, 81, 82, 94, 207, 222], serverless architectures [72, 112, 128, 152, 152, 168, 210], next-generation interconnects [30, 51, 135, 136], chiplets [90, 121, 124, 148, 163, 166, 172], and others [34, 37, 98, 122, 123].

Reasoning for discovering novel optimizations. Inspired by the schemes such as AlphaTensor [88] and enabled by the emergence of RLMs with their innate coding capabilities [22, 66, 84, 202, 205, 234], such models could also help discover novel kernels and other performance-centric schemes, including collectives, placement heuristics, and routing policies, following recent LLM-driven algorithm-discovery systems such as AlphaEvolve [182].

Evaluation and reproducibility. The evaluation of RLMs involves a plethora of aspects such as reasoning budget, sampling strategy, rollout length, staleness, and metrics associated with whole RLM components and subsystems such as tools, retrieval, and verification. It poses an opportunity for designing effective evaluation pipelines, reusing and extending evaluation methodologies for other domains such as parallel programming [24, 113].

Shared-weight RLHF and snapshot-consistent rollout

workers. Another example concrete systems opportunity is to reduce the cost of synchronizing actor weights in disaggregated or asynchronous RLHF pipelines. Current systems often push full policy snapshots from the learner to rollout workers, which creates latency, bandwidth pressure, and synchronization stalls at large model scale. A promising alternative is a shared-weight design in which rollout actors access versioned trainer-resident parameter slots, switching only at rollout boundaries to preserve snapshot consistency. Such a system could use read-copy-update-style double or triple buffering, version counters, and publication fences so that actors never observe partially written parameters. On a single node, this suggests CUDA IPC [195] or peer-to-peer refresh over NVLink or PCIe [143]; across nodes, one can harness GPUDirect RDMA [196] or NVSHMEM-style one-sided communication [116].

8 Conclusion

Reasoning Language Models (RLMs) and their underlying paradigms such as Reinforcement Learning with Verifiable Rewards (RLVR) are not only an algorithmic advance, but also an enormous parallel and distributed systems challenge due to their compute requirements. Our work systematizes the RL-for-LLM pipeline, analyzes PPO-, GRPO-, and DPO-like frameworks through work–depth–memory complexity, and develops a taxonomy of intra- and inter-model parallelism strategies for scalable RLM training and inference. The result is a **unified performance vocabulary** for understanding where computation, memory, and critical-path bottlenecks arise, and for offering opportunities for performance optimizations while considering performance-critical aspects such as autoregressive rollout generation, auxiliary-model assessment, trainable actor/critic updates, long-context memory, model-state sharding, placement, fusion, and bounded-stale execution. By making these trade-offs explicit, our analysis provides a foundation for designing the next generation of RLM systems.

Acknowledgements

We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Alps machines, and for their excellent technical support. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing compute facilities within computational grant no. PLG/2026/019437; we also thank Łukasz Flis and the whole Cyfronet team for their excellent technical support. We thank Timo Schneider for help with infrastructure at SPCL. This project received funding from the European Research Council (Project PSAP, No. 101002047), and the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955513 (MAELSTROM). This project was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. We acknowledge the Swiss AI Initiative for the computational grant. A language model served as an editorial tool for this manuscript, while ideas and content are original work of the authors.

References

[1] M. Abdin, S. Agarwal, A. Awadallah, V. Balachandran, H. Behl, L. Chen, G. de Rosa, S. Gunasekar, M. Javaheripi, N. Joshi *et al.*, "Phi-4-reasoning Technical Report," Apr. 2025, arXiv:2504.21318. [Online]. Available: https://arxiv.org/abs/2504.21318

[2] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl *et al.*, "Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone," Aug. 2024, arXiv:2404.14219. [Online]. Available: https://arxiv.org/abs/2404.14219

[3] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 Technical Report," Dec. 2024, arXiv:2412.08905. [Online]. Available: https://arxiv.org/abs/2412.08905

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman *et al.*, "GPT-4 Technical Report," Mar. 2024, arXiv:2303.08774. [Online]. Available: https://arxiv.org/abs/2303.08774

[5] B. Adler, N. Agarwal, A. Aithal, D. H. Anh, P. Bhattacharya, A. Brundyn, J. Casper, B. Catanzaro, S. Clay, J. Cohen *et al.*, "Nemotron-4 340B Technical Report," Aug. 2024, arXiv:2406.11704. [Online]. Available: https://arxiv.org/abs/2406.11704

[6] A. Ahmadian, C. Cremer, M. Gallé, M. Fadaee, J. Kreutzer, O. Pietquin, A. Üstün, and S. Hooker, "Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '24, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 12 248–12 267. [Online]. Available: https://aclanthology.org/2024.acl-long.662/

[7] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-In-Memory Architecture," *SIGARCH Comput. Archit. News*, vol. 43, no. 3S, pp. 336–348, Jun. 2015. [Online]. Available: https://doi.org/10.1145/2872887.2750385

[8] R. Ai, Y. Pan, D. Simchi-Levi, M. Tambe, and H. Xu, "Beyond Majority Voting: LLM Aggregation by Leveraging Higher-Order Information," May 2026, arXiv:2510.01499. [Online]. Available: https://arxiv.org/abs/2510.01499

[9] E. Anderson, J. Fritz, A. Lee, B. Li, M. Lindblad, H. Lindeman, A. Meyer, P. Parmar, T. Ranade, M. A. Shah, B. Sowell, D. Tecuci, V. Thapliyal, and M. Welsh, "The Design of an LLM-Powered Unstructured Analytics System," in *Proceedings of the 15th Annual Conference on Innovative Data Systems Research*, ser. CIDR '25. Amsterdam, Netherlands: VLDB Endowment, Jan. 2025, pp. 1–10. [Online]. Available: https://vldb.org/cidrdb/2025/the-design-of-an-llm-powered-unstructured-analytics-system.html

[10] R. Angles and C. Gutierrez, "Survey of Graph Database Models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1:1–1:39, Feb. 2008. [Online]. Available: https://doi.org/10.1145/1322432.1322433

[11] ——, "An Introduction to Graph Data Management," in *Graph Data Management: Fundamental Issues and Recent Developments*, ser. Data-Centric Systems and Applications (DCSA), G. Fletcher, J. Hidders, and J. L. Larriba-Pey, Eds. Berlin, Germany: Springer, Nov. 2018, pp. 1–32. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-96193-4_1

[12] R. Anil, S. Borgeaud, J.-B. Alayrac, R. S. Jiahui Yu, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver *et al.*, "Gemini: A Family of Highly Capable Multimodal Models," May 2025, arXiv:2312.11805. [Online]. Available: https://arxiv.org/abs/2312.11805

[13] Anthropic, "Claude, Large Language Model Conversational AI," https://claude.ai/new, 2024, [Accessed June 11, 2026].

[14] ——, "Claude 3.7 Sonnet and Claude Code," https://www.anthropic.com/news/claude-3-7-sonnet, Feb. 2025, [Accessed June 12, 2026].

[15] ——, "Building with Extended Thinking," https://platform.claude.com/docs/en/build-with-claude/extended-thinking, 2026, [Accessed June 12, 2026].

[16] G. Atluri, A. Karpatne, and V. Kumar, "Spatio-Temporal Data Mining: A Survey of Problems and Methods," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 83:1–83:41, Aug. 2018. [Online]. Available: https://doi.org/10.1145/3161602

[17] G. Bai, Z. Chai, C. Ling, S. Wang, J. Lu, N. Zhang, T. Shi, Z. Yu, M. Zhu, Y. Zhang, X. Song, C. Yang, Y. Cheng, and L. Zhao, "Beyond Efficiency: A Systematic Survey of Resource-Efficient Large Language Models," Dec. 2024, arXiv:2401.00625. [Online]. Available: https://arxiv.org/abs/2401.00625

[18] T. Bai, Y. Bai, Y. Bao, S. Cai, Y. Cao, Y. Charles, H. Che, C. Chen, G. Chen, H. Chen *et al.*, "Kimi K2.5: Visual Agentic Intelligence," Feb. 2026, arXiv:2602.02276. [Online]. Available: https://arxiv.org/abs/2602.02276

[19] Y. Bai, Y. Bao, Y. Charles, C. Chen, G. Chen, H. Chen, H. Chen, J. Chen, N. Chen, R. Chen *et al.*, "Kimi K2: Open Agentic Intelligence," Feb. 2026, arXiv:2507.20534. [Online]. Available: https://arxiv.org/abs/2507.20534

[20] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, N. Joseph *et al.*, "Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback," Apr. 2022, arXiv:2204.05862. [Online]. Available: https://arxiv.org/abs/2204.05862

[21] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon *et al.*, "Constitutional AI: Harmlessness from AI Feedback," Dec. 2022, arXiv:2212.08073. [Online]. Available: https://arxiv.org/abs/2212.08073

[22] R. Bairi, A. Sonwane, A. Kanade, V. D. C., A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, and S. Shet, "CodePlan: Repository-Level Coding using LLMs and Planning," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 31:1–31:24, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3643757

[23] M. I. L. Balaka, D. Alexander, Q. Wang, Y. Gong, A. Krisnadhi, and R. Castro Fernandez, "Pneuma: Leveraging LLMs for Tabular Data Representation and Retrieval in an End-to-End System," *Proc. ACM Manag. Data*, vol. 3, no. 3, pp. 200:1–200:28, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3725337

[24] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning," in *Proceedings of the IEEE 33rd International Parallel and Distributed Processing Symposium*, ser. IPDPS '19. Rio de Janeiro, Brazil: IEEE Press, May 2019, pp. 66–77. [Online]. Available: https://ieeexplore.ieee.org/document/8821020

[25] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 65:1–65:43, Aug. 2019. [Online]. Available: https://doi.org/10.1145/3320060

[26] M. Besta, J. Barth, E. Schreiber, A. Kubicek, A. Catarino, R. Gerstenberger, P. Nyczyk, P. Iff, Y. Li, S. Houliston, T. Sternal, M. Copik, G. Kwaśniewski, J. Müller, Łukasz Flis, H. Eberhard, Z. Chen, H. Niewiadomski, and T. Hoefler, "Reasoning Language Models: A Blueprint," Jun. 2025, arXiv:2501.11223. [Online]. Available: https://arxiv.org/abs/2501.11223

[27] M. Besta, A. Carigiet, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, and T. Hoefler, "High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. Atlanta, GA, USA: IEEE Press, Nov. 2020, pp. 99:1–99:17. [Online]. Available: https://ieeexplore.ieee.org/document/9355236

[28] M. Besta, A. C. Catarino, L. Gianinazzi, N. Blach, P. Nyczyk, H. Niewiadomski, and T. Hoefler, "HOT: Higher-Order Dynamic Graph Representation Learning with Efficient Transformers," in *Proceedings of the Second Learning on Graphs Conference (LOG '23)*, ser. Proceedings of Machine Learning Research, S. Villar and B. Chamberlain, Eds., vol. 231. Virtual Event: PMLR, Nov. 2023, pp. 15:1–15:20. [Online]. Available: https://proceedings.mlr.press/v231/besta24a.html

[29] M. Besta, S. Chandran, R. Gerstenberger, M. Lindner, M. Chrapek, S. H. Martschat, T. Ghandi, P. Iff, H. Niewiadomski, P. Nyczyk, J. Müller, and T. Hoefler, "Psychologically Enhanced AI Agents," Sep. 2025, arXiv:2509.04343. [Online]. Available: https://arxiv.org/abs/2509.04343

[30] M. Besta, J. Domke, M. Schneider, M. Konieczny, S. Di Girolamo, T. Schneider, A. Singla, and T. Hoefler, "High-Performance Routing with Multipathing and Path Diversity in Ethernet and HPC Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 943–959, Apr. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9248644

[31] M. Besta, M. Fischer, T. Ben-Nun, D. Stanojevic, J. D. F. Licht, and T. Hoefler, "Substream-Centric Maximum Matchings on FPGA," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 8:1–8:33, Apr. 2020. [Online]. Available: https://doi.org/10.1145/3377871

[32] M. Besta, M. Fischer, V. Kalavri, M. Kapralov, and T. Hoefler, "Practice of Streaming Processing of Dynamic Graphs: Concepts, Models, and Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 6, pp. 1860–1876, Jun. 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9629281

[33] M. Besta, R. Gerstenberger, M. Fischer, M. Podstawski, N. Blach, B. Egeli, G. Mitenkov, W. Chlapek, M. Michalewicz, H. Niewiadomski, J. Mueller, and T. Hoefler, "The Graph Database Interface: Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. Denver, CO, USA: Association for Computing Machinery, Nov. 2023, pp. 22:1–22:18. [Online]. Available: https://doi.org/10.1145/3581784.3607068

[34] M. Besta, R. Gerstenberger, P. Iff, P. Sonawane, J. G. Luna, R. Kanakagiri, R. Min, O. Mutlu, T. Hoefler, R. Appuswamy, and A. O Mahony, "Hardware Acceleration for Knowledge Graph Processing: Challenges & Recent Developments," Nov. 2024, arXiv:2408.12173. [Online]. Available: https://arxiv.org/abs/2408.12173

[35] M. Besta, R. Gerstenberger, E. Peter, M. Fischer, M. Podstawski, C. Barthels, G. Alonso, and T. Hoefler, "Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries," *ACM Comput. Surv.*, vol. 56, no. 2, pp. 31:1–31:40, Sep. 2023. [Online]. Available: https://doi.org/10.1145/3604932

[36] M. Besta, R. Grob, C. Miglioli, N. Bernold, G. Kwasniewski, G. Gjini, R. Kanakagiri, S. Ashkboos, L. Gianinazzi, N. Dryden, and T. Hoefler, "Motif Prediction with Graph Neural Networks," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. Washington, DC, USA: Association for Computing Machinery, Aug. 2022, pp. 35–45. [Online]. Available: https://doi.org/10.1145/3534678.3539343

[37] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler, "Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability," *SIGPLAN Not.*, vol. 53, no. 2, pp. 43–55, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3296957.3177158

[38] M. Besta and T. Hoefler, "Fault Tolerance for Remote Memory Access Programming Models," in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '14. Vancouver, BC, Canada: Association for Computing Machinery, Jun. 2014, pp. 37–48. [Online]. Available: https://doi.org/10.1145/2600212.2600224

[39] ——, "Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15. Newport Beach, CA, USA: Association for Computing Machinery, Jun. 2015, pp. 155–164. [Online]. Available: https://doi.org/10.1145/2751205.2751219

[40] ——, "Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 5, pp. 2584–2606, May 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10443519

[41] M. Besta, P. Iff, F. Scheidl, K. Osawa, N. Dryden, M. Podstawski, T. Chen, and T. Hoefler, "Neural Graph Databases," in *Proceedings of the First Learning on Graphs Conference*, ser. Proceedings of Machine Learning Research, B. Rieck and R. Pascanu, Eds., vol. 198. Virtual Event: PMLR, Dec. 2022. [Online]. Available: https://proceedings.mlr.press/v198/besta22a.html

[42] M. Besta, Ł. Jarmocik, O. Hrycyna, S. Klaiman, K. Maczka, R. Gerstenberger, J. Müller, P. Nyczyk, H. Niewiadomski, and T. Hoefler, "GraphSeek: Next-Generation Graph Analytics with LLMs," Mar. 2026, arXiv:2602.11052. [Online]. Available: https://arxiv.org/abs/2602.11052

[43] M. Besta, R. Kanakagiri, G. Kwaśniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan, J. G. Luna, J. Golinowski, M. Copik, L. Kapp-Schwoerer, S. Di Girolamo, N. Blach, M. Konieczny, O. Mutlu, and T. Hoefler, "SISA: Set-

Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. Virtual Event: Association for Computing Machinery, Oct. 2021, pp. 282–297. [Online]. Available: https://doi.org/10.1145/3466752.3480133

[44] M. Besta, A. Kubicek, R. Gerstenberger, M. Chrapek, R. Niggli, P. Okanovic, Y. Zhu, P. Iff, M. Podstawski, L. Weitzendorf, M. Chi, J. Gajda, P. Nyczyk, J. Müller, H. Niewiadomski, and T. Hoefler, "Multi-Head RAG: Solving Multi-Aspect Problems with LLMs," Sep. 2025, arXiv:2406.05085. [Online]. Available: https://arxiv.org/abs/2406.05085

[45] M. Besta, F. Memedi, Z. Zhang, R. Gerstenberger, G. Piao, N. Blach, P. Nyczyk, M. Copik, G. Kwaśniewski, J. Müller, L. Gianinazzi, A. Kubicek, H. Niewiadomski, A. O'Mahony, O. Mutlu, and T. Hoefler, "Demystifying Chains, Trees, and Graphs of Thoughts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 12, pp. 10 967–10 989, Dec. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11123142

[46] M. Besta, C. Miglioli, P. S. Labini, J. Tětek, P. Iff, R. Kanakagiri, S. Ashkboos, K. Janda, M. Podstawski, G. Kwaśniewski, N. Gleinig, F. Vella, O. Mutlu, and T. Hoefler, "ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. Dallas, TX, USA: IEEE Press, Nov. 2022, pp. 43:1–43:17. [Online]. Available: https://ieeexplore.ieee.org/document/10046121

[47] M. Besta, L. Paleari, M. Copik, R. Gerstenberger, A. Kubicek, P. Nyczyk, P. Iff, E. Schreiber, T. Srindran, T. Lehmann, H. Niewiadomski, and T. Hoefler, "CheckEmbed: Effective Verification of LLM Solutions to Open-Ended Tasks," Jul. 2025, arXiv:2406.02524. [Online]. Available: https://arxiv.org/abs/2406.02524

[48] M. Besta, L. Paleari, J. H. A. Jiang, R. Gerstenberger, Y. Wu, J. G. Hannesson, P. Iff, A. Kubicek, P. Nyczyk, D. Khimey *et al.*, "Affordable AI Assistants with Knowledge Graph of Thoughts," Oct. 2025, arXiv:2504.02670. [Online]. Available: https://arxiv.org/abs/2504.02670

[49] M. Besta, P. Renc, R. Gerstenberger, P. Sylos Labini, A. Ziogas, T. Chen, L. Gianinazzi, F. Scheidl, K. Szenes, A. Carigiet, P. Iff, G. Kwaśniewski, R. Kanakagiri, C. Ge, S. Jaeger, J. Was, F. Vella, and T. Hoefler, "High-Performance and Programmable Attentional Graph Neural Networks with Global Tensor Formulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. Denver, CO, USA: Association for Computing Machinery, Nov. 2023, pp. 66:1–66:16. [Online]. Available: https://doi.org/10.1145/3581784.3607067

[50] M. Besta, F. Scheidl, L. Gianinazzi, G. Kwaśniewski, S. Klaiman, J. Müller, and T. Hoefler, "Demystifying Higher-Order Graph Neural Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 48, no. 3, pp. 2544–2565, Mar. 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11267259

[51] M. Besta, M. Schneider, M. Konieczny, K. Cynk, E. Henriksson, S. Di Girolamo, A. Singla, and T. Hoefler, "FatPaths: Routing in Supercomputers and Data Centers When Shortest Paths Fall Short," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. Atlanta, GA, USA: IEEE Press, Nov. 2020, pp. 27:1–27:18. [Online]. Available: https://ieeexplore.ieee.org/document/9355307

[52] M. Besta, Z. Vonarburg-Shmaria, Y. Schaffner, L. Schwarz, G. Kwaśniewski, L. Gianinazzi, J. Beranek, K. Janda, T. Holenstein, S. Leisinger, P. Tatkowski, E. Ozdemir, A. Balla, M. Copik, P. Lindenberger, M. Konieczny, O. Mutlu, and T. Hoefler, "GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 1922–1935, Jul. 2021. [Online]. Available: https://doi.org/10.14778/3476249.3476252

[53] G. Bilardi and A. Pietracaprina, "Models of Computation, Theoretical," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA, USA: Springer, Sep. 2011, pp. 1150–1158. [Online]. Available: https://link.springer.com/rwe/10.1007/978-0-387-09766-4_218

[54] S. M. Birjandi and S. H. Khasteh, "A Survey on Data Mining Techniques Used in Medicine," *Journal of Diabetes & Metabolic Disorders*, vol. 20, no. 2, pp. 2055–2071, Dec. 2021. [Online]. Available: https://link.springer.com/article/10.1007/s40200-021-00884-2

[55] A. Biswal, L. Patel, S. Jha, A. Kamsetty, S. Liu, J. E. Gonzalez, C. Guestrin, and M. Zaharia, "Text2SQL is Not Enough: Unifying AI and Databases with TAG," in *Proceedings of the 15th Annual Conference on Innovative Data Systems Research*, ser. CIDR '25. Amsterdam, Netherlands: VLDB Endowment, Jan. 2025, pp. 1–10. [Online]. Available: https://vldb.org/cidrdb/2025/text2sql-is-not-enough-unifying-ai-and-databases-with-tag.html

[56] G. E. Blelloch and B. M. Maggs, "Parallel Algorithms," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 51–54, Mar. 1996. [Online]. Available: https://doi.org/10.1145/234313.234339

[57] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going Beyond Euclidean Data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, Jul. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7974879

[58] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent Advances in Graph Partitioning," in *Algorithm Engineering: Selected Results and Surveys*, ser. Lecture Notes in Computer Science, L. Kliemann and P. Sanders, Eds. Springer Nature, Nov. 2016, vol. 9220, pp. 117–158. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-49487-6_4

[59] Z. Cai, M. Cao, H. Chen, K. Chen, K. Chen, X. Chen, X. Chena, Z. Chen, Z. Chen, P. Chu *et al.*, "InternLM2 Technical Report," Mar. 2024, arXiv:2403.17297. [Online]. Available: https://arxiv.org/abs/2403.17297

[60] Y. Cang, X. Zhang, E. Zhao, Z. Ji, Y. Liu, Y. He, Z. Ning, Y. Chen, W. Que, and L. Shi, "Graph-GRPO: Stabilizing Multi-Agent Topology Learning via Group Relative Policy Optimization," Mar. 2026, arXiv:2603.02701. [Online]. Available: https://arxiv.org/abs/2603.02701

[61] Y. Cao, H. Zhao, Y. Cheng, T. Shu, Y. Chen, G. Liu, G. Liang, J. Zhao, J. Yan, and Y. Li, "Survey on Large Language Model-Enhanced Reinforcement Learning: Concept, Taxonomy, and Methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 6, pp. 9737–9757, Jun. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10766898

[62] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine Learning on Graphs: A Model and Comprehensive Taxonomy," *Journal of Machine Learning Research*, vol. 23, pp. 89:1–89:64, May 2022. [Online]. Available: http://jmlr.org/papers/v23/20-852.html

[63] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective Communication: Theory, Practice, and Experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, Jul. 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cpe.1206

[64] J. Chang and F. Nargesian, "Approximating Opaque Top-k Queries," *Proc. ACM Manag. Data*, vol. 3, no. 3, pp. 129:1–129:25, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3725266

[65] J. Chen, S. Li, R. Guo, J. Yuan, and T. Hoefler, "AutoDDL: Automatic Distributed Deep Learning With Near-Optimal Bandwidth Cost," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 8, pp. 1331–1344, Aug. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10521778

[66] J. Chen, Z. Li, X. Hu, and X. Xia, "NLPerturbator: Studying the Robustness of Code LLMs to Natural Language Variations," *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 4, pp. 89:1–89:20, Mar. 2026. [Online]. Available: https://doi.org/10.1145/3745764

[67] L. Chen, B. Acun, N. Ardalani, Y. Sun, F. Kang, H. Lyu, Y. Kwon, R. Jia, C.-J. Wu, M. Zaharia, and J. Zou, "Data Acquisition: A New Frontier in Data-Centric AI," *Journal of Data-Centric Machine Learning Research*, vol. 2, Jan. 2025. [Online]. Available: https://data.mlr.press/assets/pdf/v02-11.pdf

[68] Z. Chen, S. Wang, Z. Tan, X. Fu, Z. Lei, P. Wang, H. Liu, C. Shen, and J. Li, "A Survey of Scaling in Large Language Model Reasoning," Apr. 2026, arXiv:2504.02181. [Online]. Available: https://arxiv.org/abs/2504.02181

[69] S. Choudhury, "Process Reward Models for LLM Agents: Practical Framework and Directions," Feb. 2025, arXiv:2502.10325. [Online]. Available: https://arxiv.org/abs/2502.10325

[70] Z. Chu, S. Wang, J. Xie, T. Zhu, Y. Yan, J. Ye, A. Zhong, X. Hu, J. Liang, P. S. Yu, and Q. Wen, "LLM Agents

for Education: Advances and Applications," in *Findings of the Association for Computational Linguistics: EMNLP 2025*, C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng, Eds. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 13 782–13 810. [Online]. Available: https://aclanthology.org/2025.findings-emnlp.743/

[71] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen *et al.*, "Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities," Dec. 2025, arXiv:2507.06261. [Online]. Available: https://arxiv.org/abs/2507.06261

[72] M. Copik, G. Kwaśniewski, M. Besta, M. Podstawski, and T. Hoefler, "SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. Québec City, Canada: Association for Computing Machinery, Dec. 2021, pp. 64–78. [Online]. Available: https://doi.org/10.1145/3464298.3476133

[73] B. Cottier, R. Rahman, L. Fattorini, N. Maslej, T. Besiroglu, and D. Owen, "The Rising Costs of Training Frontier AI Models," Feb. 2025, arXiv:2404.05952. [Online]. Available: https://arxiv.org/abs/2405.21015

[74] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang, "DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '24, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 1280–1297. [Online]. Available: https://aclanthology.org/2024.acl-long.70/

[75] J. Dai, X. Pan, R. Sun, J. Ji, X. Xu, M. Liu, Y. Wang, and Y. Yang, "Safe RLHF: Safe Reinforcement Learning from Human Feedback," in *Proceedings of the Twelfth International Conference on Learning Representations*, ser. ICLR '24, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., Vienna, Austria, May 2024, pp. 50 750–50 777. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/hash/dd1577afd396928ed64216f3f1fd5556-Abstract-Conference.html

[76] N. Dai, Z. Wu, R. Zheng, Z. Wei, W. Shi, X. Jin, G. Liu, C. Dun, L. Huang, and L. Yan, "Process Supervision-Guided Policy Optimization for Code Generation," Feb. 2025, arXiv:2410.17621. [Online]. Available: https://arxiv.org/abs/2410.17621

[77] A. Dandashi and M. Al-Mouhamed, "Graph Coloring for Class Scheduling," in *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, ser. AICCSA '10. Hammamet, Tunisia: IEEE Press, May 2010, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/5586963

[78] A. Davoudian, L. Chen, and M. Liu, "A Survey on NoSQL Stores," *ACM Comput. Surv.*, vol. 51, no. 2, pp. 40:1–40:43, Apr. 2018. [Online]. Available: https://doi.org/10.1145/3158661

[79] B. L. M. de Oliveira, F. V. Frujeri, M. P. C. M. Queiroz, L. G. B. Martins, T. W. de L. Soares, and L. C. Melo, "Learning Without Critics? Revisiting GRPO in Classical Reinforcement Learning Environments," Nov. 2025, arXiv:2511.03527. [Online]. Available: https://arxiv.org/abs/2511.03527

[80] S. Devunuri and L. Lehe, "TransitGPT: A Generative AI-Based Framework for Interacting with GTFS Data Using Large Language Models," *Public Transport*, vol. 17, no. 2, pp. 319–345, Jun. 2025. [Online]. Available: https://link.springer.com/article/10.1007/s12469-025-00395-w

[81] S. Di Girolamo, D. De Sensi, K. Taranov, M. Malesevic, M. Besta, T. Schneider, S. Kistler, and T. Hoefler, "Building Blocks for Network-Accelerated Distributed File Systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. Dallas, TX, USA: IEEE Press, Nov. 2022, pp. 10:1–10:14. [Online]. Available: https://doi.org/10.1109/SC41404.2022.00015

[82] S. Di Girolamo, K. Taranov, A. Kurth, M. Schaffner, T. Schneider, J. Beránek, M. Besta, L. Benini, D. Roweth, and T. Hoefler, "Network-Accelerated Non-Contiguous Memory Transfers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. Denver, CO, USA: Association for Computing Machinery, Nov. 2019, pp. 56:1–56:14. [Online]. Available: https://doi.org/10.1145/3295500.3356189

[83] T. Ding, T. Chen, H. Zhu, J. Jiang, Y. Zhong, J. Zhou, G. Wang, Z. Zhu, I. Zharkov, and L. Liang, "The Efficiency Spectrum of Large Language Models: An Algorithmic Survey," Apr. 2024, arXiv:2312.00678. [Online]. Available: https://arxiv.org/abs/2312.00678

[84] Y. Dong, X. Jiang, J. Qian, T. Wang, K. Zhang, Z. Jin, and G. Li, "A Survey on Code Generation with LLM-Based Agents," Sep. 2025, arXiv:2508.00083. [Online]. Available: https://arxiv.org/abs/2508.00083

[85] A. Dorbani, S. Yasser, J. Lin, and A. Mhedhbi, "Beyond Quacking: Deep Integration of Language Models and RAG into DuckDB," *Proc. VLDB Endow.*, vol. 18, no. 12, pp. 5415–5418, Sep. 2025. [Online]. Available: https://doi.org/10.14778/3750601.3750685

[86] A. Du, B. Gao, B. Xing, C. Jiang, C. Chen, C. Li, C. Xiao, C. Du, C. Liao, C. Tang *et al.*, "Kimi k1.5: Scaling Reinforcement Learning with LLMs," Jun. 2025, arXiv:2501.12599. [Online]. Available: https://arxiv.org/abs/2501.12599

[87] K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela, "Model Alignment as Prospect Theoretic Optimization," in *Proceedings of the 41st International Conference on Machine Learning (ICML '24)*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. Vienna, Austria: PMLR, Jul. 2024, pp. 29 634–29 651. [Online]. Available: https://proceedings.mlr.press/v235/ethayarajh24a.html

[88] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli, "Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning," *Nature*, vol. 610, no. 7930, pp. 47–53, Oct. 2022. [Online]. Available: https://www.nature.com/articles/s41586-022-05172-4

[89] L. Feng, C. Pan, X. Guo, F. Mei, B. Ning, J. Zhang, X. Liu, B. Zhou, Z. Shu, C. Liu, G. Yang, Z. Han, J. Wang, and B. Wang, "MindSpeed RL: Distributed Dataflow for Scalable and Efficient RL Training on Ascend NPU Cluster," Jul. 2025, arXiv:2507.19017. [Online]. Available: https://arxiv.org/abs/2507.19017

[90] Y. Feng and K. Ma, "Chiplet Actuary: A Quantitative Cost Model and Multi-Chiplet Architecture Exploration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. San Francisco, CA, USA: Association for Computing Machinery, Jul. 2022, pp. 121–126. [Online]. Available: https://doi.org/10.1145/3489517.3530428

[91] E. Frick, P. Jin, T. Li, K. Ganesan, J. Zhang, J. Jiao, and B. Zhu, "Athene-70B," https://huggingface.co/Nexusflow/Athene-70B, Jul. 2024, [Accessed June 11, 2026].

[92] W. Fu, J. Gao, S. Xu, Z. Mei, C. Zhu, X. Shen, C. He, G. Wei, J. Mei, W. Jiashu, T. Yang, B. Yuan, and Y. Wu, "AREAL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning," in *Proceedings of the 3rd Workshop on Efficient Systems for Foundation Models*, ser. ES-FoMo '25. Vancouver, Canada: OpenReview, Jul. 2025, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=qJ0okaW9Z9

[93] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," Mar. 2024, arXiv:2312.10997. [Online]. Available: https://arxiv.org/abs/2312.10997

[94] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. Denver, CO, USA: Association for Computing Machinery, Nov. 2013, pp. 53:1–53:12. [Online]. Available: https://doi.org/10.1145/2503210.2503286

[95] M. Gheisari, H. Hamidpour, Y. Liu, P. Saedi, A. Raza, A. Jalili, H. Rokhsati, and R. Amin, "Data Mining Techniques for Web Mining: A Survey," *Artificial Intelligence and Applications*, vol. 1, no. 1, pp. 3–10, Jan. 2023. [Online]. Available: https://ojs.bonviewpress.com/index.php/AIA/article/view/290

[96] M. Gheshlaghi Azar, Z. Daniel Guo, B. Piot, R. Munos, M. Rowland, M. Valko, and D. Calandriello, "A General Theoretical Paradigm to Understand Learning from Human Preferences," in *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS '24)*, ser. Proceedings of Machine Learning Research, S. Dasgupta,

S. Mandt, and Y. Li, Eds., vol. 238. Valencia, Spain: PMLR, May 2024, pp. 4447–4455. [Online]. Available: https://proceedings.mlr.press/v238/gheshlaghi-azar24a.html

[97] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-Memory: A Workload-Driven Perspective," *IBM Journal of Research and Development*, vol. 63, no. 6, pp. 3:1–3:19, Nov. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8792187

[98] L. Gianinazzi, T. Ben-Nun, M. Besta, S. Ashkboos, Y. Baumann, P. Luczynski, and T. Hoefler, "The Spatial Computer: A Model for Energy-Efficient Parallel Computation," Jan. 2023, arXiv:2205.04934. [Online]. Available: https://arxiv.org/abs/2205.04934

[99] L. Gianinazzi, M. Besta, Y. Schaffner, and T. Hoefler, "Parallel Algorithms for Finding Large Cliques in Sparse Graphs," in *Proceedings of the 33rd Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '21. Virtual Event: Association for Computing Machinery, Jul. 2021, pp. 243–253. [Online]. Available: https://doi.org/10.1145/3409964.3461800

[100] L. Gianinazzi, M. Fries, N. Dryden, T. Ben-Nun, M. Besta, and T. Hoefler, "Learning Combinatorial Node Labeling Algorithms," May 2022, arXiv:2106.03594. [Online]. Available: https://arxiv.org/abs/2106.03594

[101] L. Gianinazzi, P. Kalvoda, A. De Palma, M. Besta, and T. Hoefler, "Communication-Avoiding Parallel Minimum Cuts and Connected Components," *SIGPLAN Not.*, vol. 53, no. 1, pp. 219–232, Feb. 2018. [Online]. Available: https://doi.org/10.1145/3200691.3178504

[102] Google DeepMind, "Gemini 2.5 Deep Think," https://blog.google/products-and-platforms/products/gemini/gemini-2-5-deep-think/, Aug. 2025, [Accessed June 15, 2026].

[103] P. Gorinski, M. Zimmer, G. Lampouras, D. G. X. Deik, and I. Iacobacci, "Automatic Unit Test Data Generation and Actor-Critic Reinforcement Learning for Code Synthesis," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 370–384. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.28/

[104] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The Llama 3 Herd of Models," Nov. 2024, arXiv:2407.21783. [Online]. Available: https://arxiv.org/abs/2407.21783

[105] J. Gu, X. Jiang, Z. Shi, H. Tan, X. Zhai, C. Xu, W. Li, Y. Shen, S. Ma, H. Liu, S. Wang, K. Zhang, Z. Lin, B. Zhang, L. Ni, W. Gao, Y. Wang, and J. Guo, "A Survey on LLM-as-a-Judge," *The Innovation*, vol. 7, no. 6, pp. 101 253:1–101 253:30, Jun. 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666675825004564

[106] X. Guan, L. L. Zhang, Y. Liu, N. Shang, Y. Sun, Y. Zhu, F. Yang, and M. Yang, "rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking," in *Proceedings of the 42nd International Conference on Machine Learning (ICML '25)*, ser. Proceedings of Machine Learning Research, A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, Eds., vol. 267. Vancouver, Canada: PMLR, Jul. 2025, pp. 20 640–20 661. [Online]. Available: https://proceedings.mlr.press/v267/guan25f.html

[107] D. Guo, D. Yang, H. Zhang, J. Song, P. Wang, Q. Zhu, R. Xu, R. Zhang, S. Ma, X. Bi *et al.*, "DeepSeek-R1 Incentivizes Reasoning in LLMs Through Reinforcement Learning," *Nature*, vol. 645, no. 8081, pp. 633–638, Sep. 2025. [Online]. Available: https://www.nature.com/articles/s41586-025-09422-z

[108] H. Guo, H. Lu, G. Nan, B. Chu, J. Zhuang, Y. Yang, W. Che, X. Cao, S. Leng, Q. Cui, and X. Jiang, "Advancing Expert Specialization for Better MoE," in *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS '25)*, ser. Advances in Neural Information Processing Systems, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, Eds., vol. 38. San Diego, CA, USA: Curran Associates, Dec. 2025, pp. 48 767–48 809. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/hash/4598de7d243d528e38eb0c5d8155fb52-Abstract-Conference.html

[109] M. K. Gupta and P. Chandra, "A Comprehensive Survey of Data Mining," *International Journal of Information Technology*, vol. 12, no. 4, pp. 1243–1257, Dec. 2020. [Online]. Available: https://link.springer.com/article/10.1007/s41870-020-00427-7

[110] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *Bulletin of the Technical Committee on Data Engineering*, vol. 40, no. 3, pp. 52–74, Sep. 2017. [Online]. Available: http://sites.computer.org/debull/A17sept/p52.pdf

[111] Z. Han, A. You, H. Wang, K. Luo, G. Yang, W. Shi, M. Chen, S. Zhang, Z. Lan, C. Deng *et al.*, "AsyncFlow: An Asynchronous Streaming RL Framework for Efficient LLM Post-Training," Jul. 2025, arXiv:2507.01663. [Online]. Available: https://arxiv.org/abs/2507.01663

[112] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, "Survey on Serverless Computing," *Journal of Cloud Computing*, vol. 10, no. 1, pp. 39:1–39:29, Jul. 2021. [Online]. Available: https://link.springer.com/article/10.1186/s13677-021-00253-7

[113] T. Hoefler and R. Belli, "Scientific Benchmarking of Parallel Computing Systems: Twelve Ways to Tell the Masses When Reporting Performance Results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. Austin, TX, USA: Association for Computing Machinery, Nov. 2015, pp. 73:1–73:12. [Online]. Available: https://doi.org/10.1145/2807591.2807644

[114] J. Hong, N. Lee, and J. Thorne, "ORPO: Monolithic Preference Optimization Without Reference Model," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '24, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, FL, USA: Association for Computational Linguistics, Nov. 2024, pp. 11 170–11 189. [Online]. Available: https://aclanthology.org/2024.emnlp-main.626/

[115] Z. Hou, P. Du, Y. Niu, Z. Du, A. Zeng, X. Liu, M. Huang, H. Wang, J. Tang, and Y. Dong, "Does RLHF Scale? Exploring the Impacts From Data, Model, and Method," Dec. 2024, arXiv:2412.06000. [Online]. Available: https://arxiv.org/abs/2412.06000

[116] C.-H. Hsu, N. Imam, A. Langer, S. Potluri, and C. J. Newburn, "An Initial Assessment of NVSHMEM for High Performance Computing," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops*, ser. IPDPSW '20. New Orleans, LA, USA: IEEE Press, May 2020, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/document/9150438

[117] J. Hu, X. Wu, W. Shen, J. K. Liu, W. Wang, S. Jiang, H. Wang, H. Chen, B. Chen, W. Fang, Xianyu, Y. Cao, H. Xu, and Y. Liu, "OpenRLHF: A Ray-Based Easy-to-Use, Scalable and High-Performance RLHF Framework," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, ser. EMNLP '25, I. Habernal, P. Schulam, and J. Tiedemann, Eds. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 656–666. [Online]. Available: https://aclanthology.org/2025.emnlp-demos.48/

[118] J. Hu, Y. Zhang, Q. Han, D. Jiang, X. Zhang, and H.-Y. Shum, "Open-Reasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model," in *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS '25)*, ser. Advances in Neural Information Processing Systems, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, Eds., vol. 38. San Diego, CA, USA: Curran Associates, Dec. 2025, pp. 162 239–162 262. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/hash/ed873d79e7c268c020c4b4db13a2812a-Abstract-Conference.html

[119] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, and E. Chen, "Understanding the Planning of LLM Agents: A Survey," Feb. 2024, arXiv:2402.02716. [Online]. Available: https://arxiv.org/abs/2402.02716

[120] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," in *Proceedings of the Thirty-Third Annual Conference on Neural Information Processing Systems (NeurIPS '19)*, ser. Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Vancouver, Canada: Curran Associates, Dec. 2019, pp. 103–112. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html

[121] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler, "HexaMesh: Scaling to Hundreds of Chiplets with an Optimized Chiplet Arrangement," in *Proceedings of the 60th ACM/IEEE Design Automation Conference*, ser. DAC '23, San

Francisco, CA, USA, Jul. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10248006

[122] ——, "Sparse Hamming Graph: A Customizable Network-on-Chip Topology," in *Proceedings of the 2023 60th ACM/IEEE Design Automation Conference*, ser. DAC '23. San Francisco, CA, USA: IEEE Press, Jul. 2023, pp. 1–6. [Online]. Available: https://doi.org/10.1109/DAC56929.2023.10247754

[123] P. Iff, T. Bonato, M. Besta, L. Benini, and T. Hoefler, "Network Design for Wafer-Scale Systems with Wafer-on-Wafer Hybrid Bonding," Mar. 2026, arXiv:2603.05266. [Online]. Available: https://arxiv.org/abs/2603.05266

[124] P. Iff, B. Bruggmann, M. Besta, L. Benini, and T. Hoefler, "PlaceIT: Placement-Based Inter-Chiplet Interconnect Topologies," Feb. 2025. [Online]. Available: https://arxiv.org/abs/2502.01449

[125] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, R. Y. Aminadabi, S. L. Song, S. Rajbhandari, and Y. He, "System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models," in *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, ser. PODC '24. Nantes, France: Association for Computing Machinery, Jun. 2024, pp. 121–130. [Online]. Available: https://doi.org/10.1145/3662158.3662806

[126] Z. Jiang, F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active Retrieval Augmented Generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '23, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7969–7992. [Online]. Available: https://aclanthology.org/2023.emnlp-main.495/

[127] N. M. Johnson and J. Cao, "Announcing PyTorch/XLA 2.3: Distributed Training, Dev Improvements, and GPUs," https://cloud.google.com/blog/products/ai-machine-learning/introducing-pytorch-xla-2-3, Apr. 2024, [Accessed June 16, 2026].

[128] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," Feb. 2019, arXiv:1902.03383. [Online]. Available: https://arxiv.org/abs/1902.03383

[129] M. Khalifa, R. Agarwal, L. Logeswaran, J. Kim, H. Peng, M. Lee, H. Lee, and L. Wang, "Process Reward Models That Think," *Transactions on Machine Learning Research*, pp. 1–37, Mar. 2026, j2C Certification. [Online]. Available: https://openreview.net/forum?id=FPVCb0WMuN

[130] J. Kim, E. Ewer, T. Moon, J. Park, and D. Papailiopoulos, "Not All Bits Are Equal: Scale-Dependent Memory Optimization Strategies for Reasoning Models," in *Proceedings of the Fourteenth International Conference on Learning Representations*, ser. ICLR '26. Rio de Janeiro, Brazil: OpenReview, Apr. 2026, pp. 1–24. [Online]. Available: https://openreview.net/forum?id=b6qQmQ2F13

[131] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the International Conference on Learning Representations*, ser. ICLR '17. Toulon, France: OpenReview, Apr. 2017, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[132] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing Activation Recomputation in Large Transformer Models," in *Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys '23)*, ser. Proceedings of Machine Learning and Systems, D. Song, M. Carbin, and T. Chen, Eds., vol. 5. Miami Beach, FL, USA: Curran Associates, Jun. 2023, pp. 341–353. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html

[133] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. Koblenz, Germany: Association for Computing Machinery, Oct. 2023, pp. 611–626. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[134] X. Lai, Z. Tian, Y. Chen, S. Yang, X. Peng, and J. Jia, "Step-DPO: Step-Wise Preference Optimization for Long-Chain Reasoning of LLMs," Jun. 2024, arXiv:2406.18629. [Online]. Available: https://arxiv.org/abs/2406.18629

[135] K. Lakhotia, M. Besta, L. Monroe, K. Isham, P. Iff, T. Hoefler, and F. Petrini, "PolarFly: A Cost-Effective and Flexible Low-Diameter Topology," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. Dallas, TX, USA: IEEE Press, Nov. 2022, pp. 12:1–12:15. [Online]. Available: https://ieeexplore.ieee.org/document/10046084

[136] K. Lakhotia, L. Monroe, K. Isham, M. Besta, N. Blach, T. Hoefler, and F. Petrini, "PolarStar: Expanding the Horizon of Diameter-3 Networks," in *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '24. Nantes, France: Association for Computing Machinery, Jun. 2024, pp. 345–357. [Online]. Available: https://doi.org/10.1145/3626183.3659975

[137] K. Lan, D.-t. Wang, S. Fong, L.-s. Liu, K. K. Wong, and N. Dey, "A Survey of Data Mining and Deep Learning in Bioinformatics," *Journal of Medical Systems*, vol. 42, no. 8, p. 139, Jun. 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10916-018-1003-9

[138] H. Le, Y. Wang, A. D. Gotmare, S. Savarese, and S. C. H. Hoi, "CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning," in *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS '22)*, ser. Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. New Orleans, LA, USA: Curran Associates, Dec. 2022, pp. 21 314–21 328. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html

[139] A. W. Lee, J. Chan, M. Fu, N. Kim, A. Mehta, D. Raghavan, and U. Çetintemel, "Semantic Integrity Constraints: Declarative Guardrails for AI-Augmented Data Processing Systems," *Proc. VLDB Endow.*, vol. 18, no. 11, pp. 4073–4080, Sep. 2025. [Online]. Available: https://doi.org/10.14778/3749646.3749677

[140] H. Lee, S. Phatale, H. Mansoor, T. Mesnard, J. Ferret, K. R. Lu, C. Bishop, E. Hall, V. Carbune, A. Rastogi, and S. Prakash, "RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback," in *Proceedings of the 41st International Conference on Machine Learning (ICML '24)*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. Vienna, Austria: PMLR, Jul. 2024, pp. 26 874–26 901. [Online]. Available: https://proceedings.mlr.press/v235/lee24t.html

[141] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding ," in *Proceedings of the Ninth International Conference on Learning Representations*, ser. ICLR '21. Virtual Event: OpenReview, May 2021, pp. 1–23. [Online]. Available: https://openreview.net/forum?id=qrwe7XHTmYb

[142] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proceedings of the Thirty-Fourth Annual Conference on Neural Information Processing Systems (NeurIPS '20)*, ser. Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Virtual Event: Curran Associates, Dec. 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[143] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 94–110, Jan. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8763922

[144] H. Li, Y. Li, A. Tian, T. Tang, Z. Xu, X. Chen, N. HU, W. Dong, L. Qing, and L. Chen, "A Survey on Large Language Model Acceleration Based on KV Cache Management," *Transactions on Machine Learning Research*, pp. 1–61, May 2025. [Online]. Available: https://openreview.net/forum?id=z3JZzu9EA3

[145] H. Li, S. Lin, F. Fu, Y. Zhou, X. Ji, Y. Zhao, L. Wang, J. Jiang, and B. Cui, "Unleashing Efficient Asynchronous RL Post-Training via Staleness-Constrained Rollout Coordination," Jan. 2026, arXiv:2601.12784. [Online]. Available: https://arxiv.org/abs/2601.12784

[146] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li,

A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala, "PyTorch Distributed: Experiences on Accelerating Data Parallel Training," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3005–3018, Aug. 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415530

[147] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, "Colossal-AI: A Unified Deep Learning System for Large-Scale Parallel Training," in *Proceedings of the 52nd International Conference on Parallel Processing*, ser. ICPP '23. Salt Lake City, UT, USA: Association for Computing Machinery, Aug. 2023, pp. 766–775. [Online]. Available: https://doi.org/10.1145/3605573.3605613

[148] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun, "Chiplet Heterogeneous Integration Technology–Status and Challenges," *Electronics*, vol. 9, no. 4, pp. 670:1–670:12, Apr. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/4/670

[149] W. Li and Y. Li, "Process Reward Model with Q-Value Rankings," in *Proceedings of the Thirteenth International Conference on Learning Representations*, ser. ICLR '25, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., Singapore, Apr. 2025, pp. 14 708–14 726. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/26494b66ae1114d314673e25b4967288-Abstract-Conference.html

[150] Y. Li, Z. Wang, T. Fu, G. Cui, S. Yang, and Y. Cheng, "From Drafts to Answers: Unlocking LLM Potential via Aggregation Fine-Tuning," Jan. 2025, arXiv:2501.11877. [Online]. Available: https://arxiv.org/abs/2501.11877

[151] Z. Li, X. Feng, Y. Cai, Z. Zhang, T. Liu, C. Liang, W. Chen, H. Wang, and T. Zhao, "LLMs Can Generate a Better Answer by Aggregating Their Own Responses," Apr. 2025, arXiv:2503.04104. [Online]. Available: https://arxiv.org/abs/2503.04104

[152] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The Serverless Computing Survey: A Technical Primer for Design Architecture," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 220:1–220:34, Sep. 2022. [Online]. Available: https://doi.org/10.1145/3508360

[153] Z. Li, T. Xu, Y. Zhang, Z. Lin, Y. Yu, R. Sun, and Z.-Q. Luo, "ReMax: A Simple, Effective, and Efficient Reinforcement Learning Method for Aligning Large Language Models," in *Proceedings of the 41st International Conference on Machine Learning (ICML '24)*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. Vienna, Austria: PMLR, Jul. 2024, pp. 29 128–29 163. [Online]. Available: https://proceedings.mlr.press/v235/li24cd.html

[154] Z. Li, S. Huang, Z. Chi, Y. Su, L. Zhou, L. Dong, N. Collier, and F. Wei, "Breaking Training Bottlenecks: Effective and Stable Reinforcement Learning for Coding Models," Mar. 2026, arXiv:2603.07777. [Online]. Available: https://arxiv.org/abs/2603.07777

[155] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's Verify Step by Step," in *Proceedings of the Twelfth International Conference on Learning Representations*, ser. ICLR '24, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., Vienna, Austria, May 2024, pp. 39 578–39 601. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html

[156] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "DeepSeek-V3 Technical Report," Feb. 2025, arXiv:2412.19437. [Online]. Available: https://arxiv.org/abs/2412.19437

[157] A. Liu, J. Liu, Z. Pan, Y. He, G. Haffari, and B. Zhuang, "MiniCache: KV Cache Compression in Depth Dimension for Large Language Models," in *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS '24)*, ser. Advances in Neural Information Processing Systems, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Vancouver, Canada: Curran Associates, Dec. 2024, pp. 139 997–140 031. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/fd0705710bf01b88a60a3d479ea341d9-Abstract-Conference.html

[158] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, R. Shahout, and G. Vitagliano, "Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing," in *Proceedings of the 15th Annual Conference on Innovative Data Systems Research*, ser. CIDR '25. Amsterdam, Netherlands: VLDB Endowment, Jan. 2025, pp. 1–7. [Online]. Available: https://www.vldb.org/cidrdb/2025/palimpzest-optimizing-ai-powered-analytics-with-declarative-query-processing.html

[159] C. Liu, G. Vitagliano, B. Rose, M. Printz, D. A. Samson, and M. Cafarella, "PalimpChat: Declarative and Interactive AI Analytics," in *Companion of the 2025 International Conference on Management of Data*, ser. SIGMOD/PODS '25. Berlin, Germany: Association for Computing Machinery, Jun. 2025, pp. 183–186. [Online]. Available: https://doi.org/10.1145/3722212.3725122

[160] H. Liu, M. Zaharia, and P. Abbeel, "RingAttention with Blockwise Transformers for Near-Infinite Context," in *Proceedings of the Twelfth International Conference on Learning Representations*, ser. ICLR '24, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., Vienna, Austria, May 2024, pp. 3992–4008. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/hash/1119587863e78451f080da2a768c4935-Abstract-Conference.html

[161] W. Liu, X. Huang, X. Zeng, X. Hao, S. Yu, D. Li, S. Wang, W. Gan, Z. Liu, Y. Yu, Z. Wang, Y. Wang, W. Ning, Y. Hou, B. Wang, C. Wu, W. Xinzhi, Y. Liu, Y. Wang, D. Tang, D. Tu, L. Shang, X. Jiang, R. Tang, D. Lian, Q. Liu, and E. Chen, "ToolACE: Winning the Points of LLM Function Calling," in *Proceedings of the Thirteenth International Conference on Learning Representations*, ser. ICLR '25, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., Singapore, Apr. 2025, pp. 41 359–41 381. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/663865ea167425c6c562cb0b6bcf76c7-Abstract-Conference.html

[162] Z. Liu, X. Xu, P. Qiao, and D. Li, "Acceleration for Deep Reinforcement Learning using Parallel and Distributed Computing: A Survey," *ACM Comput. Surv.*, vol. 57, no. 4, pp. 91:1–91:35, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3703453

[163] G. H. Loh, S. Naffziger, and K. Lepak, "Understanding Chiplets Today to Anticipate Future Integration Opportunities and Limits," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '21. Grenoble, France: IEEE Press, Feb. 2021, pp. 142–145. [Online]. Available: https://ieeexplore.ieee.org/document/9474021

[164] P. Ma, R. Ding, S. Wang, S. Han, and D. Zhang, "InsightPilot: An LLM-Empowered Automated Data Exploration System," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, ser. EMNLP '23, Y. Feng and E. Lefever, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 346–352. [Online]. Available: https://aclanthology.org/2023.emnlp-demo.31/

[165] Q. Ma, H. Zhou, T. Liu, J. Yuan, P. Liu, Y. You, and H. Yang, "Let's Reward Step by Step: Step-Level Reward Model as the Navigators for Reasoning," Oct. 2023, arXiv:2310.10080. [Online]. Available: https://arxiv.org/abs/2310.10080

[166] X. Ma, Y. Wang, Y. Wang, X. Cai, and Y. Han, "Survey on Chiplets: Interface, Interconnect and Integration Methodology," *CCF Transactions on High Performance Computing*, vol. 4, no. 1, pp. 43–52, Mar. 2022. [Online]. Available: https://link.springer.com/article/10.1007/s42514-022-00093-0

[167] S. Madden, M. Cafarella, M. Franklin, and T. Kraska, "Databases Unbound: Querying All of the World's Bytes with AI," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4546–4554, Aug. 2024. [Online]. Available: https://doi.org/10.14778/3685800.3685916

[168] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," in *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '17. Atlanta, GA, USA: IEEE Press, Jun. 2017, pp. 405–410. [Online]. Available: https://ieeexplore.ieee.org/document/7979855

[169] Z. Mei, W. Fu, K. Li, G. Wang, H. Zhang, and Y. Wu, "ReaL: Efficient RLHF Training for Large Language Models through Parameter Reallocation," in *Proceedings of the Eighth Annual Conference on Machine Learning and Systems (MLSys '25)*, ser. Proceedings of Machine Learning and Systems, M. Zaharia, G. Joshi, and Y. Lin, Eds., vol. 7, Santa Clara, CA, USA, May 2025, pp. 1–20. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2025/hash/3b3889d313ba9476c12c2d77ea66b24f-Abstract-Conference.html

[170] Y. Meng, M. Xia, and D. Chen, "SimPO: Simple Preference Optimization with a Reference-Free Reward," in *Proceedings*

of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS '24), ser. Advances in Neural Information Processing Systems, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Vancouver, Canada: Curran Associates, Dec. 2024, pp. 124 198–124 235. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/e099c1c9699814af0be873a175361713-Abstract-Conference.html

[171] Meta AI, "The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation," https://ai.meta.com/blog/llama-4-multimodal-intelligence/, Apr. 2025, [Accessed June 15, 2026].

[172] G. Mounce, J. Lyke, S. Horan, W. Powell, R. Doyle, and R. Some, "Chiplet Based Approach for Heterogeneous Processing and Packaging Architectures," in *Proceedings of the IEEE Aerospace Conference*, ser. AERO '16. Big Sky, MT, USA: IEEE Press, Mar. 2016, pp. 1–12. [Online]. Available: https://ieeexplore.ieee.org/document/7500830

[173] Y. Mroueh, "Reinforcement Learning with Verifiable Rewards: GRPO's Effective Loss, Dynamics, and Success Amplification," Oct. 2025, arXiv:2503.06639. [Online]. Available: https://arxiv.org/abs/2503.06639

[174] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A Modern Primer on Processing in Memory," in *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*, ser. Computer Architecture and Design Methodologies (CADM), M. M. S. Aly and A. Chattopadhyay, Eds. Springer Nature, Jul. 2022, pp. 171–243. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-16-7487-7_7

[175] D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, "Memory-Efficient Pipeline-Parallel DNN Training," in *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. Virtual Event: PMLR, Jul. 2021, pp. 7937–7947. [Online]. Available: https://proceedings.mlr.press/v139/narayanan21a.html

[176] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. St. Louis, MO, USA: Association for Computing Machinery, Nov. 2021, pp. 58:1–58:15. [Online]. Available: https://doi.org/10.1145/3458817.3476209

[177] S. Nayab, G. Rossolini, M. Simoni, A. Saracino, G. Buttazzo, N. Manes, and F. Giacomelli, "Concise Thoughts: Impact of Output Length on LLM Reasoning and Cost," Jan. 2025, arXiv:2407.19825. [Online]. Available: https://arxiv.org/abs/2407.19825

[178] X.-P. Nguyen, S. Pandit, A. Xu, C. Xiong, and S. Joty, "Least-Loaded Expert Parallelism: Load Balancing an Imbalanced Mixture-of-Experts," Jan. 2026, arXiv:2601.17111. [Online]. Available: https://arxiv.org/abs/2601.17111

[179] X. Ning, Z. Lin, Z. Zhou, Z. Wang, H. Yang, and Y. Wang, "Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation," in *Proceedings of the Twelfth International Conference on Learning Representations*, ser. ICLR '24, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., Vienna, Austria, May 2024, pp. 917–967. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/hash/03d7e13f0092405804f3a381ade8f3f0-Abstract-Conference.html

[180] M. Noukhovitch, S. Huang, S. Xhonneux, A. Hosseini, R. Agarwal, and A. Courville, "Asynchronous RLHF: Faster and More Efficient Off-Policy RL for Language Models," Singapore, pp. 4003–4029, Apr. 2025. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/0b99315234cc95e6ef281f9155b68832-Abstract-Conference.html

[181] NovaSky-AI, "SkyRL: A Modular Full-Stack RL Library for LLMs," https://github.com/novasky-ai/skyrl, Apr. 2026, [Accessed June 12, 2026].

[182] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, M. P. Kumar, A. See, S. Chaudhuri, G. Holland, A. Davies, S. Nowozin, P. Kohli, and M. Balog, "AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery," Jun. 2025, arXiv:2506.13131. [Online]. Available: https://arxiv.org/abs/2506.13131

[183] NVIDIA, "Context Parallelism Overview," https://docs.nvidia.com/megatron-core/developer-guide/latest/user-guide/features/context_parallel.html, 2026, [Accessed June 17, 2026].

[184] ——, "NeMo RL: Scalable Toolkit for Efficient Model Reinforcement," https://github.com/nvidia-nemo/rl, Apr. 2026, [Accessed June 12, 2026].

[185] ——, "TensorRT-LLM," https://github.com/NVIDIA/TensorRT-LLM, Apr. 2026, [Accessed June 12, 2026].

[186] OpenAI, "OpenAI o1," https://openai.com/o1/, Dec. 2024, [Accessed June 11, 2026].

[187] ——, "OpenAI o3," https://openai.com/index/introducing-o3-and-o4-mini/, Apr. 2025, [Accessed June 11, 2026].

[188] ——, "OpenAI o3 and o4-mini System Card," https://openai.com/index/o3-o4-mini-system-card/, Apr. 2025, [Accessed June 15, 2026].

[189] ——, "GPT-5.5 in ChatGPT," https://help.openai.com/en/articles/11909943-gpt-55-in-chatgpt, Jun. 2026, [Accessed June 12, 2026].

[190] ——, "Introducing GPT-5.4," https://openai.com/index/introducing-gpt-5-4/, Mar. 2026, [Accessed June 12, 2026].

[191] A. Ormazabal, C. Zheng, C. de Masson d'Autume, D. Yogatama, D. Fu, D. Ong, E. Chen, E. Lamprecht, H. Pham, I. Ong *et al.*, "Reka Core, Flash, and Edge: A Series of Powerful Multimodal Language Models," Apr. 2024, arXiv:2404.12387. [Online]. Available: https://arxiv.org/abs/2404.12387

[192] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training Language Models to Follow Instructions with Human Feedback," in *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS '22)*, ser. Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. New Orleans, LA, USA: Curran Associates, Dec. 2022, pp. 27 730–27 744. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

[193] D. Papakyriakou and I. S. Barbounakis, "Data Mining Methods: A Review," *International Journal of Computer Applications*, vol. 183, no. 48, pp. 5–19, Jan. 2022. [Online]. Available: https://ijcaonline.org/archives/volume183/number48/32253-2022921884/

[194] L. Patel, S. Jha, M. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia, "Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS," *Proc. VLDB Endow.*, vol. 18, no. 11, pp. 4171–4184, Sep. 2025. [Online]. Available: https://doi.org/10.14778/3749646.3749685

[195] S. Potluri, H. Wang, D. Bureddy, A. Singh, C. Rosales, and D. K. Panda, "Optimizing MPI Communication on Multi-GPU Systems Using CUDA Inter-Process Communication," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, ser. IPDPSW '12. Shanghai, China: IEEE Press, May 2012, pp. 1848–1857. [Online]. Available: https://ieeexplore.ieee.org/document/6270863

[196] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, "Efficient Inter-Node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs," in *Proceedings of the 42nd International Conference on Parallel Processing*, ser. ICPP '13. Lyon, France: IEEE Press, Oct. 2013, pp. 80–89. [Online]. Available: https://ieeexplore.ieee.org/document/6687341

[197] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero Bubble (Almost) Pipeline Parallelism," in *Proceedings of the Twelfth International Conference on Learning Representations*, ser. ICLR '24, B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, Eds., Vienna, Austria, May 2024, pp. 48 869–48 884. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2024/hash/d5a8e37f38a08c68162452dcba89ae9c-Abstract-Conference.html

[198] P. Qi, X. Zhou, Z. Liu, T. Pang, C. Du, M. Lin, and W. S. Lee, "Rethinking the Trust Region in LLM Reinforcement Learning," Jun. 2026, arXiv:2602.04879. [Online]. Available: https://arxiv.org/abs/2602.04879

[199] Z. Qiu, C. Li, Y. Peng, G. He, B. Yuan, and C. Wang, "TQA-Bench: Evaluating LLMs for Multi-Table Question Answering," Jun. 2026, arXiv:2411.19504. [Online]. Available: https://arxiv.org/abs/2411.19504

[200] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," in *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS '23)*, ser. Advances in Neural Information Processing Systems, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. New Orleans, LA, USA: Curran Associates, Dec. 2023, pp. 53 728–53 741. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html

[201] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. Atlanta, GA, USA: IEEE Press, Nov. 2020, pp. 20:1–20:16. [Online]. Available: https://ieeexplore.ieee.org/document/9355301

[202] S. Rando, L. Romani, A. Sampieri, L. Franco, J. Yang, Y. Kyuragi, F. Galasso, and T. Hashimoto, "LongCodeBench: Evaluating Coding LLMs at 1M Context Windows," in *Proceedings of the Second Conference on Language Modeling*, ser. COLM '25. Montreal, Canada: OpenReview, Oct. 2025, pp. 1–17. [Online]. Available: https://openreview.net/forum?id=GFPoM8Ylp8

[203] T. Ren, J. Jiang, H. Yang, W. Tian, M. Zou, G. Li, Z. Zhang, Q. Wang, S. Qin, Y. Zhao, R. Tao, H. Shao, and Y. Peng, "RiskPO: Risk-Based Policy Optimization with Verifiable Reward for LLM Post-Training," in *Proceedings of the Fourteenth International Conference on Learning Representations*, ser. ICLR '26. Rio de Janeiro, Brazil: OpenReview, Apr. 2026, pp. 1–20. [Online]. Available: https://openreview.net/forum?id=KjHB7rebQO

[204] M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret *et al.*, "Gemma 2: Improving Open Language Models at a Practical Size," Oct. 2024, arXiv:2408.00118. [Online]. Available: https://arxiv.org/abs/2408.00118

[205] M. Robeyns, M. Szummer, and L. Aitchison, "A Self-Improving Coding Agent," in *Proceedings of the Workshop on Scaling Self-Improving Foundation Models*, ser. SSI-FM '25. Singapore: OpenReview, Apr. 2025, pp. 1–18. [Online]. Available: https://openreview.net/forum?id=rShJCyLsOr

[206] A. Salemi and H. Zamani, "Evaluating Retrieval Quality in Retrieval-Augmented Generation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '24. Washington, DC, USA: Association for Computing Machinery, Jul. 2024, pp. 2395–2400. [Online]. Available: https://doi.org/10.1145/3626772.3657957

[207] P. Schmid, M. Besta, and T. Hoefler, "High-Performance Distributed RMA Locks," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '16. Kyoto, Japan: Association for Computing Machinery, Jun. 2016, pp. 19–30. [Online]. Available: https://doi.org/10.1145/2907294.2907323

[208] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347. [Online]. Available: https://arxiv.org/abs/1707.06347

[209] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '17. Cambridge, MA, USA: Association for Computing Machinery, Oct. 2017, pp. 273–287. [Online]. Available: https://doi.org/10.1145/3123939.3124544

[210] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless Computing: A Survey of Opportunities, Challenges, and Applications," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 239:1–239:32, Nov. 2022. [Online]. Available: https://doi.org/10.1145/3510611

[211] S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu, "DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing," *Proc. VLDB Endow.*, vol. 18, no. 9, pp. 3035–3048, Sep. 2025. [Online]. Available: https://doi.org/10.14778/3746405.3746426

[212] R. Shao, S. S. Li, R. Xin, S. Geng, Y. Wang, S. Oh, S. S. Du, N. Lambert, S. Min, R. Krishna *et al.*, "Spurious Rewards: Rethinking Training Signals in RLVR," Feb. 2026, arXiv:2506.10947. [Online]. Available: https://arxiv.org/abs/2506.10947

[213] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu, D. Guo *et al.*, "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models," Apr. 2024, arXiv:2402.03300. [Online]. Available: https://arxiv.org/abs/2402.03300

[214] G. Shen, Z. Wang, O. Delalleau, J. Zeng, Y. Dong, D. Egert, S. Sun, J. J. Zhang, S. Jain, A. Taghibakhshi, M. S. Ausin, A. Aithal, and O. Kuchaiev, "NeMo-Aligner: Scalable Toolkit for Efficient Model Alignment," in *Proceedings of the First Conference on Language Modeling*, ser. COLM '24. Philadelphia, PA, USA: OpenReview, Oct. 2024, pp. 1–16. [Online]. Available: https://openreview.net/forum?id=yK2eGE8QVW

[215] Z. Shen, "LLM With Tools: A Survey," Sep. 2024, arXiv:2409.18807. [Online]. Available: https://arxiv.org/abs/2409.18807

[216] G. Sheng, Y. Tong, B. Wan, W. Zhang, C. Jia, X. Wu, Y. Wu, X. Li, C. Zhang, Y. Peng, H. Lin, X. Liu, and C. Wu, "Laminar: A Scalable Asynchronous RL Post-Training Framework," Oct. 2025, arXiv:2510.12633. [Online]. Available: https://arxiv.org/abs/2510.12633

[217] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu, "HybridFlow: A Flexible and Efficient RLHF Framework," in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys '25. Rotterdam, The Netherlands: Association for Computing Machinery, Apr. 2025, pp. 1279–1297. [Online]. Available: https://doi.org/10.1145/3689031.3696075

[218] L. Shi, H. Zhang, Y. Yao, Z. Li, and H. Zhao, "Keep the Cost Down: A Review on Methods to Optimize LLM's KV-Cache Consumption," in *Proceedings of the First Conference on Language Modeling*, ser. COLM '24. Philadelphia, PA, USA: OpenReview, Oct. 2024, pp. 1–19. [Online]. Available: https://openreview.net/forum?id=8tKjqqMM5z

[219] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," Mar. 2020, arXiv:1909.08053. [Online]. Available: https://arxiv.org/abs/1909.08053

[220] A. Singh, A. Fry, A. Perelman, A. Tart, A. Ganesh, A. El-Kishky, A. McLaughlin, A. Low, A. Ostrow, A. Ananthram *et al.*, "OpenAI GPT-5 System Card," May 2026, arXiv:2601.03267. [Online]. Available: https://arxiv.org/abs/2601.03267

[221] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning," in *Proceedings of the Thirteenth International Conference on Learning Representations*, ser. ICLR '25, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., Singapore, Apr. 2025, pp. 10 131–10 165. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html

[222] A. Strausz, F. Vella, S. Di Girolamo, M. Besta, and T. Hoefler, "Asynchronous Distributed-Memory Triangle Counting and LCC with RMA Caching," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '22. Lyon, France: IEEE Press, May 2022, pp. 291–301. [Online]. Available: https://ieeexplore.ieee.org/document/9820724

[223] S. Sun, Y. Zhang, A. Bukharin, D. Mosallanezhad, J. Zeng, S. Singhal, G. Shen, A. Renduchintala, T. Konuk, Y. Dong *et al.*, "Reward-Aware Preference Optimization: A Unified Mathematical Framework for Model Alignment," Feb. 2025, arXiv:2502.00203. [Online]. Available: https://arxiv.org/abs/2502.00203

[224] F. Tajwar, G. Zeng, Y. Zhou, Y. Song, D. Arora, Y. Jiang, J. Schneider, R. Salakhutdinov, H. Feng, and A. Zanette, "Maximum Likelihood Reinforcement Learning," Feb. 2026, arXiv:2602.02710. [Online]. Available: https://arxiv.org/abs/2602.02710

[225] Z. Tan, H. Geng, X. Yu, M. Zhang, G. Wan, Y. Zhou, Q. He, X. Xue, H. Zhou, Y. Fan *et al.*, "Scaling Behaviors of LLM Reinforcement Learning Post-Training: An Empirical Study in Mathematical Reasoning," Apr. 2026, arXiv:2509.25300. [Online]. Available: https://arxiv.org/abs/2509.25300

[226] R. Teknium, J. Quesnelle, and C. Guang, "Hermes 3 Technical Report," Aug. 2024, arXiv:2408.11857. [Online]. Available: https://arxiv.org/abs/2408.11857

[227] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *The International Journal of High Performance Computing Applications*,

vol. 19, no. 1, pp. 49–66, Feb. 2005. [Online]. Available: https://journals.sagepub.com/doi/10.1177/1094342005051521

[228] M. Tomar, L. Shani, Y. Efroni, and M. Ghavamzadeh, "Mirror Descent Policy Optimization," in *Proceedings of the Tenth International Conference on Learning Representations*, ser. ICLR '22. Virtual Event: OpenReview, Apr. 2022, pp. 1–24. [Online]. Available: https://openreview.net/forum?id=aBO5SvgSt1

[229] L. Tunstall, E. E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. V. Werra, C. Fourrier, N. Habib *et al.*, "Zephyr: Direct Distillation of LM Alignment," in *Proceedings of the First Conference on Language Modeling*, ser. COLM '24. Philadelphia, PA, USA: OpenReview, Oct. 2024, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=aKkAwZB6JV

[230] vLLM Contributors, "Parallelism and Scaling," https://docs.vllm.ai/en/stable/serving/parallelism_scaling/, May 2026, [Accessed June 11, 2026].

[231] L. von Werra, Y. Belkada, L. Tunstall, E. Beeching, T. Thrush, N. Lambert, S. Huang, K. Rasul, and Q. Gallouédec, "TRL – Transformer Reinforcement Learning," https://github.com/huggingface/trl, May 2026, [Accessed June 11, 2026].

[232] B. Wang, R. Zheng, L. Chen, Y. Liu, S. Dou, C. Huang, W. Shen, S. Jin, E. Zhou, C. Shi *et al.*, "Secrets of RLHF in Large Language Models Part II: Reward Modeling," Jan. 2024, arXiv:2401.06080. [Online]. Available: https://arxiv.org/abs/2401.06080

[233] H. Wang, X. Li, D. Wang, H. Zhou, Z. Huang, Y. Yang, J. Li, and Y. Ban, "Policy Improvement Reinforcement Learning," Jun. 2026, arXiv:2604.00860. [Online]. Available: https://arxiv.org/abs/2604.00860

[234] J. Wang and Y. Chen, "A Review on Code Generation with LLMs: Application and Evaluation," in *Proceedings of the IEEE International Conference on Medical Artificial Intelligence*, ser. MedAI '23. Beijing, China: IEEE Press, Nov. 2023, pp. 284–289. [Online]. Available: https://ieeexplore.ieee.org/document/10403378

[235] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui, "Math-Shepherd: Verify and Reinforce LLMs Step-by-Step without Human Annotations," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '24, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 9426–9439. [Online]. Available: https://aclanthology.org/2024.acl-long.510/

[236] S. Wang, J. Cao, and P. S. Yu, "Deep Learning for Spatio-Temporal Data Mining: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3681–3700, Aug. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9204396

[237] S. Wang, S. Zhang, J. Zhang, R. Hu, X. Li, T. Zhang, J. Li, F. Wu, G. Wang, and E. Hovy, "Reinforcement Learning Enhanced LLMs: A Survey," Feb. 2025, arXiv:2412.10400. [Online]. Available: https://arxiv.org/abs/2412.10400

[238] T. Wang, S. Li, and W. Lu, "Self-Training with Direct Preference Optimization Improves Chain-of-Thought Reasoning," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '26, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 11 917–11 928. [Online]. Available: https://aclanthology.org/2024.acl-long.643/

[239] T. Wang, Y. Li, L. Li, Y. Chen, S. Huang, Y. Chen, P. Li, Y. Liu, and G. Chen, "SPPO: Sequence-Level PPO for Long-Horizon Reasoning Tasks," in *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '26, M. Liakata, V. P. Moreira, J. Zhang, and D. Jurgens, Eds. San Diego, CA, USA: Association for Computational Linguistics, Jul. 2026, pp. 7831–7853. [Online]. Available: https://aclanthology.org/2026.acl-long.356/

[240] W. Wang, S. Xiong, G. Chen, W. Gao, S. Guo, Y. He, J. Huang, J. Liu, Z. Li, X. Li *et al.*, "Reinforcement Learning Optimization for Large-Scale Learning: An Efficient and User-Friendly Scaling Library," Jun. 2025, arXiv:2506.06122. [Online]. Available: https://arxiv.org/abs/2506.06122

[241] W. Wang, Z. Gao, L. Chen, Z. Chen, J. Zhu, X. Zhao, Y. Liu, Y. Cao, S. Ye, X. Zhu *et al.*, "VisualPRM: An Effective Process Reward Model for Multimodal Reasoning," Mar. 2025, arXiv:2503.10291. [Online]. Available: https://arxiv.org/abs/2503.10291

[242] Z. Wang, C. Li, Y. Zhang, H. Liu, B. Wang, D. Chu, and D. Sui, "VPO: Reasoning Preferences Optimization Based on $\mathcal{V}$-Usable Information," in *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS '25)*, ser. Advances in Neural Information Processing Systems, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, Eds., vol. 38. Mexico City, Mexico: Curran Associates, Dec. 2025, pp. 171 903–171 928. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/hash/fad622800f1d8ed14ef2e917469f443a-Abstract-Conference.html

[243] X. Wen, Z. Liu, S. Zheng, S. Ye, Z. Wu, Y. Wang, Z. Xu, X. Liang, J. Li, Z. Miao, J. Bian, and M. Yang, "Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs," in *Proceedings of the Fourteenth International Conference on Learning Representations*, ser. ICLR '26. Rio de Janeiro, Brazil: OpenReview, Apr. 2026, pp. 1–34. [Online]. Available: https://openreview.net/forum?id=jGbRWwIidy

[244] L. Weng, X. Wang, J. Lu, Y. Feng, Y. Liu, H. Feng, D. Huang, and W. Chen, "InsightLens: Augmenting LLM-Powered Data Analysis with Interactive Insight Management and Navigation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 31, no. 6, pp. 3719–3732, Jun. 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10989518

[245] G. Wölflein, D. Ferber, D. Truhn, O. Arandjelovic, and J. N. Kather, "LLM Agents Making Agent Tools," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ser. ACL '25, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 26 092–26 130. [Online]. Available: https://aclanthology.org/2025.acl-long.1266/

[246] J. Wu, X. Wang, Z. Yang, J. Wu, J. Gao, B. Ding, X. Wang, and X. He, "AlphaDPO: Adaptive Reward Margin for Direct Preference Optimization," in *Proceedings of the 42nd International Conference on Machine Learning (ICML '25)*, ser. Proceedings of Machine Learning Research, A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, Eds., vol. 267. Vancouver, Canada: PMLR, Jul. 2025, pp. 67 793–67 809. [Online]. Available: https://proceedings.mlr.press/v267/wu25af.html

[247] Y. Wu, Z. Sun, S. Li, S. Welleck, and Y. Yang, "Inference Scaling Laws: An Empirical Analysis of Compute-Optimal Inference for LLM Problem-Solving," in *Proceedings of the Thirteenth International Conference on Learning Representations*, ser. ICLR '25, Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, Eds., Singapore, Apr. 2025, pp. 55 824–55 845. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/8c3caae2f725c8e2a55ecd600563d172-Abstract-Conference.html

[248] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9046288

[249] xAI, "Grok 4," https://x.ai/news/grok-4, Jul. 2025, [Accessed June 12, 2026].

[250] W. Xiao, Z. Wang, L. Gan, S. Zhao, Z. Li, R. Lei, W. He, L. A. Tuan, L. Chen, H. Jiang, Z. Zhao, and F. Wu, "A Comprehensive Survey of Direct Preference Optimization: Datasets, Theories, Variants, and Applications," Jun. 2026, arXiv:2410.15595. [Online]. Available: https://arxiv.org/abs/2410.15595

[251] Y. Xiao, Z. Zhou, F. Mao, W. Wu, S. Zhao, L. Ju, L. Liang, X. Zhang, and J. Zhou, "FlexRLHF: A Flexible Placement and Parallelism Framework for Efficient RLHF Training," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '25. Milano, Italy: IEEE Press, Jun. 2025, pp. 358–369. [Online]. Available: https://ieeexplore.ieee.org/document/11078517

[252] H. Xu, X. Mao, F.-L. Li, X. Wu, W. Chen, W. Zhang, and A. T. Luu, "Full-Step-DPO: Self-Supervised Preference Optimization with Step-Wise Rewards for Mathematical Reasoning," in *Findings of the Association for Computational Linguistics: ACL 2025*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 24 343–24 356. [Online]. Available: https://aclanthology.org/2025.findings-acl.1249/

[253] M. Xu, D. Cai, W. Yin, S. Wang, X. Jin, and X. Liu, "Resource-Efficient Algorithms and Systems of Foundation Models: A

Survey," *ACM Comput. Surv.*, vol. 57, no. 5, pp. 110:1–110:39, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3706418

[254] S. Xue, D. Qi, C. Jiang, F. Cheng, K. Chen, Z. Zhang, H. Zhang, G. Wei, W. Zhao, F. Zhou, H. Yi, S. Liu, H. Yang, and F. Chen, "Demonstration of DB-GPT: Next Generation Data Interaction System Empowered by Large Language Models," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4365–4368, Aug. 2024. [Online]. Available: https://doi.org/10.14778/3685800.3685876

[255] K. Yan, Y. Yu, Y. Yu, H. Zheng, and F. Lai, "OPPO: Accelerating PPO-Based RLHF via Pipeline Overlap," Mar. 2026, arXiv:2509.25762. [Online]. Available: https://arxiv.org/abs/2509.25762

[256] A. Yang, J. Yang, A. Ibrahim, X. Xie, B. Tang, G. Sizov, J. Park, and J. Huang, "Context Parallelism for Scalable Million-Token Inference," in *Proceedings of the Eighth Annual Conference on Machine Learning and Systems (MLSys '25)*, ser. Proceedings of Machine Learning and Systems, M. Zaharia, G. Joshi, and Y. Lin, Eds., vol. 7, Santa Clara, CA, USA, May 2025, pp. 1–16. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2025/hash/78834433edc3291f4c6cbbd2759324db-Abstract-Conference.html

[257] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 Technical Report," May 2025, arXiv:2505.09388. [Online]. Available: https://arxiv.org/abs/2505.09388

[258] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 Technical Report," Sep. 2024, arXiv:2407.10671. [Online]. Available: https://arxiv.org/abs/2407.10671

[259] F. Yao, C. Tian, J. Liu, Z. Zhang, Q. Liu, L. Jin, S. Li, X. Li, and X. Sun, "Thinking Like an Expert: Multimodal Hypergraph-of-Thought (HoT) Reasoning to Boost Foundation Modals," Aug. 2023, arXiv:2308.06207. [Online]. Available: https://arxiv.org/abs/2308.06207

[260] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," in *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS '23)*, ser. Advances in Neural Information Processing Systems, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. New Orleans, LA, USA: Curran Associates, Dec. 2023, pp. 11 809–11 822. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html

[261] Z. Yao, R. Y. Aminabadi, O. Ruwase, S. Rajbhandari, X. Wu, A. A. Awan, J. Rasley, M. Zhang, C. Li, C. Holmes, Z. Zhou, M. Wyatt, M. Smith, L. Kurilenko, H. Qin, M. Tanaka, S. Che, S. L. Song, and Y. He, "DeepSpeed-Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-Like Models at All Scales," Aug. 2023, arXiv:2308.01320. [Online]. Available: https://arxiv.org/abs/2308.01320

[262] X. Ying, W. Mengdi, C. Long, L. Lian, Z. Shixin, Z. Lei, and W. Ying, "Pipe-RLHF: A Computation Mode-Aware Parallel Framework for RLHF," *Journal of Computer Research and Development*, vol. 62, no. 6, pp. 1513–1529, Jun. 2025. [Online]. Available: https://crad.ict.ac.cn/en/article/doi/10.7544/issn1000-1239.202550127

[263] Yotta Labs, "Performance Optimization for Reinforcement Learning on AMD GPUs," https://www.yottalabs.ai/post/performance-optimization-for-reinforcement-learning-on-amd-gpus, Oct. 2025, [Accessed June 11, 2026].

[264] F. Yu, T. Liu, and K. Sun, "Optimizing Communication for Mixture-of-Experts Training with Hybrid Expert Parallel," https://developer.nvidia.com/blog/optimizing-communication-for-mixture-of-experts-training-with-hybrid-expert-parallel/, Feb. 2026, [Accessed June 17, 2026].

[265] Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, J. Liu *et al.*, "DAPO: An Open-Source LLM Reinforcement Learning System at Scale," in *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS '25)*, ser. Advances in Neural Information Processing Systems, D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, Eds., vol. 38. San Diego, CA, USA: Curran Associates, Dec. 2025, pp. 113 222–113 244. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html

[266] R. Yuan, M. Khandoga, and V. K. Sankarapu, "Beyond KL Divergence: Policy Optimization with Flexible Bregman Divergences for LLM Reasoning," Feb. 2026, arXiv:2602.04380. [Online]. Available: https://arxiv.org/abs/2602.04380

[267] Y. Yue, "ColossalChat: An Open-Source Solution for Cloning ChatGPT with a Complete RLHF Pipeline," https://medium.com/pytorch/5edf08fb538b, Mar. 2023, [Accessed June 11, 2026].

[268] Y. Yue, Y. Yuan, Q. Yu, X. Zuo, R. Zhu, W. Xu, J. Chen, C. Wang, T. Fan, Z. Du *et al.*, "VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks," Apr. 2025, arXiv:2504.05118. [Online]. Available: https://arxiv.org/abs/2504.05118

[269] S. Zeighami, Y. Lin, S. Shankar, and A. Parameswaran, "LLM-Powered Proactive Data Systems," *Bulletin of the Technical Committee on Data Engineering*, vol. 49, no. 1, pp. 90–103, Mar. 2025. [Online]. Available: http://sites.computer.org/debull/A25mar/p90.pdf

[270] L. Zeng, L. Zhong, L. Zhao, T. Wei, L. Yang, J. He, C. Cheng, R. Hu, Y. Liu, S. Yan, H. Fang, and Y. Zhou, "Skywork-Math: Data Scaling Laws for Mathematical Reasoning in Large Language Models–The Story Goes On," Jul. 2024. [Online]. Available: https://arxiv.org/abs/2407.08348

[271] H. Zhang, P. Wang, S. Diao, Y. Lin, R. Pan, H. Dong, D. Zhang, P. Molchanov, and T. Zhang, "Entropy-Regularized Process Reward Model," *Transactions on Machine Learning Research*, pp. 1–22, Jun. 2025. [Online]. Available: https://openreview.net/forum?id=cSxDH7N3x9

[272] J. Zhang, H. Zhang, R. Chakravarti, Y. Hu, P. Ng, A. Katsifodimos, H. Rangwala, G. Karypis, and A. Halevy, "CoddLLM: Empowering Large Language Models for Data Analytics," Feb. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.00329

[273] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, and J. Lin, "The Lessons of Developing Process Reward Models in Mathematical Reasoning," in *Findings of the Association for Computational Linguistics: ACL 2025*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 10 495–10 516. [Online]. Available: https://aclanthology.org/2025.findings-acl.547/

[274] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, Jan. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9039675

[275] W. Zhao, P. Aggarwal, S. Saha, A. Celikyilmaz, J. Weston, and I. Kulikov, "The Majority Is Not Always Right: RL Training for Solution Aggregation," Sep. 2025, arXiv:2509.06870. [Online]. Available: https://arxiv.org/abs/2509.06870

[276] X. Zhao, X. Zhou, and G. Li, "Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4481–4484, Aug. 2024. [Online]. Available: https://doi.org/10.14778/3685800.3685905

[277] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3848–3860, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611569

[278] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "SGLang: Efficient Execution of Structured Language Model Programs," in *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS '24)*, ser. Advances in Neural Information Processing Systems, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Vancouver, Canada: Curran Associates, Dec. 2024, pp. 62 557–62 583. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/fd0705710bf01b88a60a3d479ea341d9-Abstract-Conference.html

[279] R. Zheng, S. Dou, S. Gao, Y. Hua, W. Shen, B. Wang, Y. Liu, S. Jin, Q. Liu, Y. Zhou, L. Xiong, L. Chen, Z. Xi, N. Xu, W. Lai, M. Zhu, C. Chang, Z. Yin, R. Weng, W. Cheng, H. Huang, T. Sun, H. Yan, T. Gui, Q. Zhang, X. Qiu, and X. Huang, "Secrets of RLHF in Large Language Models Part I: PPO," Jul. 2023, arXiv:2307.04964. [Online]. Available: https://arxiv.org/abs/2307.04964

[280] H. Zhong, Z. Shan, G. Feng, W. Xiong, X. Cheng, L. Zhao,

D. He, J. Bian, and L. Wang, "DPO Meets PPO: Reinforced Token Optimization for RLHF," in *Proceedings of the 42nd International Conference on Machine Learning (ICML '25)*, ser. Proceedings of Machine Learning Research, A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, Eds., vol. 267. Vancouver, Canada: PMLR, Jul. 2025, pp. 78 498–78 521. [Online]. Available: https://proceedings.mlr.press/v267/zhong25b.html

[281] Y. Zhong, Z. Zhang, X. Song, H. Hu, C. Jin, B. Wu, N. Chen, Y. Chen, Y. Zhou, C. Wan, H. Zhou, Y. Jiang, Y. Zhu, and D. Jiang, "StreamRL: Scalable, Heterogeneous, and Elastic RL for LLMs with Disaggregated Stream Generation," Apr. 2025, arXiv:2504.15930. [Online]. Available: https://arxiv.org/abs/2504.15930

[282] Y. Zhong, Z. Zhang, B. Wu, S. Liu, Y. Chen, C. Wan, H. Hu, L. Xia, R. Ming, Y. Zhu, and X. Jin, "Optimizing RLHF Training for Large Language Models with Stage Fusion," in *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '25. Philadelphia, PA, USA: USENIX Association, Apr. 2025, pp. 489–503. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/zhong

[283] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[284] P. Zhou, C. Liu, J. Ren, X. Zhou, Y. Xie, M. Cao, Z. Rao, Y.-L. Huang, D. Chong, J. Liu, J. B. Kim, S. Wang, R. C.-W. Wong, and S. Kim, "When Large Vision Language Models Meet Multimodal Sequential Recommendation: An Empirical Study," in *Proceedings of the ACM on Web Conference*, ser. WWW '25. Sydney, Australia: Association for Computing Machinery, Apr. 2025, pp. 275–292. [Online]. Available: https://doi.org/10.1145/3696410.3714764

[285] B. Zhu, E. Frick, T. Wu, H. Zhu, K. Ganesan, W.-L. Chiang, J. Zhang, and J. Jiao, "Starling-7B: Improving Helpfulness and Harmlessness with RLAIF," in *Proceedings of the First Conference on Language Modeling*, ser. COLM '24. Philadelphia, PA, USA: OpenReview, Oct. 2024, pp. 1–26. [Online]. Available: https://openreview.net/forum?id=GqDntYTTbk

[286] H. Zhu, Y. Ren, Y. Li, M. Lin, L. Yang, X. Liu, X. Zhen, H. Liu, and B. Zhang, "Unbiased Dynamic Pruning for Efficient Group-Based Policy Optimization," Mar. 2026, arXiv:2603.04135. [Online]. Available: https://arxiv.org/abs/2603.04135

[287] Z. Zhu, C. Xie, X. Lv, and slime Contributors, "slime: An LLM Post-Training Framework for RL Scaling," https://github.com/THUDM/slime, May 2026, [Accessed June 15, 2026].

APPENDIX A
RLM PIPELINE: FUNCTIONAL DESCRIPTION

In the RLM pipeline, each stage can be understood as a function acting on batches, with clearly defined inputs and outputs. We now describe these stages in more detail.

A.1 Generation Stage

The Generation stage takes as input a batch of prompts $X = \{x_1, \ldots, x_B\} \subset \mathcal{X}$ and the behavior policy $\pi_{\theta_{\text{old}}}$, which either comes from the previous RL iteration, or – for the 1st iteration – is the base model. Here, $\mathcal{X}$ is the space of input prompts. For each input prompt $x_b \in \mathcal{X}$, the policy model $\pi_\theta(y \mid x_b)$ generates a set of K candidate completions $\text{Gen}_\theta(x_b) = \left\{y_b^{(i)}\right\}_{i=1}^{K}$. The i-th candidate response to a prompt x_b is denoted as $y_b^{(i)} \in \mathcal{Y}$ and it is a sequence of T tokens $\left(y_{b,1}^{(i)}, \ldots, y_{b,T}^{(i)}\right)$, where $y_{b,t}^{(i)}$ represents a token at time-step t; $\mathcal{Y}$ is the space of token sequences. The output is thus the rollout batch $\mathcal{B}_{\text{roll}} = \left\{\left(x_b, y_b^{(i)}\right)\right\}_{b,i}$. Along with the sampled outputs, PPO-like methods often also compute per-token actor log-probabilities $\log \pi_\theta\left(y_{b,t}^{(i)} \mid x_b, y_{b,<t}^{(i)}\right)$ for each step $t \in \{1, \ldots, T\}$, for use in the training stage.

A.2 Assessment Stage

The Assessment stage takes the rollout batch $\mathcal{B}_{\text{roll}}$ and evaluates it with the auxiliary models required by the algorithm. In PPO-like methods, this usually includes the reward model R_φ, reference policy π_{ref}, and critic V_ψ that are used to transform Generation's output into numerical learning signals. The most important such signals are **sequence-level scalar rewards** $r_b^{(i)}$: a single quality score assigned to each complete sequence $y_b^{(i)}$, typically computed by the frozen reward model $R_\varphi\left(x_b, y_b^{(i)}\right)$. These scores quantify alignment with human preferences, such as helpfulness or safety. Moreover, the stage also produces **token-level training targets** $A_b^{(i)} = \left(A_{b,1}^{(i)} A_{b,2}^{(i)} \ldots A_{b,T}^{(i)}\right)$; each such target is a dense vector of values for each token position $t \in \{1, \ldots, T\}$. These values, commonly referred to as *advantages*, enable granular credit assignment by comparing the empirical return $\hat{G}_{b,t}^{(i)}$ against the estimate return by the critic model $V_\psi\left(x_b, y_{b,<t}^{(i)}\right)$. They are constructed from returns/rewards and, for PPO-like actor–critic methods, and critic value estimates, e.g., $A_{b,t}^{(i)} = \hat{G}_{b,t}^{(i)} - v_{b,t,i}$. In critic-free methods such as GRPO, no V_ψ pass is performed; advantages are computed from group-normalized rewards instead. Finally, the Assessment stage also involves computing **reference log-probabilities** $\log \pi_{\text{ref}}\left(y_{b,t}^{(i)} \mid x_b, y_{b,<t}^{(i)}\right)$.

A.3 Training Stage

The *Training stage* takes the rollout batch and the learning signals from Assessment: rewards, reference log-probabilities, stored sample-time actor log-probabilities, and, when applicable, advantages and value targets. It updates the trainable model components required by the algorithm. In PPO, the actor uses policy-ratio terms such as $\pi_\theta\left(y_{b,t}^{(i)} \mid x_b, y_{b,<t}^{(i)}\right) / \pi_{\theta_{\text{old}}}\left(y_{b,t}^{(i)} \mid x_b, y_{b,<t}^{(i)}\right)$, while the critic is updated with a value-regression loss. In GRPO, only the actor is updated because no learned critic is used. In DPO, Training operates on preference pairs rather than online rollouts and updates only the policy using teacher-forced likelihoods of preferred and dispreferred responses. Thus, Training outputs the next policy $\pi_{\theta'}$, and updated critic parameters ψ' only for algorithms that train a critic.

A.4 Iterative Loop

This process is repeated in an iterative loop: after the policy update, the new parameters $\theta' \leftarrow \theta$ are used in the next Generation stage. The full loop therefore evolves as $\theta \rightarrow \text{Gen}_\theta(X) \rightarrow \text{Assess}(X, Y) \rightarrow \text{Train}(\cdot) \rightarrow \theta' \rightarrow \ldots$ and continues until convergence or until a predetermined number of iterations is reached.

APPENDIX B
DETAILS OF MATHEMATICAL DERIVATIONS

We offer details of mathematical derivations. To simplify equations, we use $C_f^{\text{tok}} \equiv C_f$, $C_b^{\text{tok}} \equiv C_b$, and $C_{\text{gen}}^{\text{roll}} \equiv C_{\text{gen}}$.

B.1 Results for Building Blocks

We now derive the model-specific building blocks used in Table 3 for forward and backward passes in terms of architectural parameters such as the number of layers L, hidden size d, feed-forward width d_{ff}, vocabulary size V, and sequence lengths S, T. We parameterize these values with a given model M in question, i.e., hidden size d_M, feed-forward width $d_{M,\text{ff}}$, and L_M layers.

B.1.1 Forward-Pass FLOPs

We begin with a standard dense Transformer layer. For a model M, one layer contains **four dense attention projections**: $W_Q, W_K, W_V, W_O \in \mathbb{R}^{d_M \times d_M}$, contributing $4d_M^2$ parameters, and **two FFN projections**, $W_{\text{up}} \in \mathbb{R}^{d_M \times d_{M,\text{ff}}}$ and $W_{\text{down}} \in \mathbb{R}^{d_{M,\text{ff}} \times d_M}$, contributing $2d_M d_{M,\text{ff}}$ parameters. Thus, one layer contains

$$4d_M^2 + 2d_M d_{M,\text{ff}}$$

trainable projection parameters.

Under the standard FLOP accounting for matrix multiplication, each parameter contributes approximately two FLOPs per processed token in the forward pass (one multiply and one add). Hence, the parameter-dependent forward FLOPs per token per layer are

$$2\big(4d_M^2 + 2d_M d_{M,\text{ff}}\big).$$

Across L_M layers, this becomes

$$2L_M\big(4d_M^2 + 2d_M d_{M,\text{ff}}\big).$$

In addition, self-attention performs parameter-free token-mixing operations, namely score computation and value aggregation. Under the same coarse cost model used throughout the paper, these contribute

$$2L_M(S+T)d_M$$

FLOPs per token on a sequence of total length $S+T$.

Therefore, the base forward-pass cost per token for model M is

$$C_f(M) = 2L_M\big(4d_M^2 + 2d_M d_{M,\text{ff}} + (S+T)d_M\big),$$

up to model-specific output heads.

B.1.1.1 **Policy and reference model**: We parameterize $M \in \{\pi, \text{ref}\}$. For the policy model π_θ, one additionally computes logits over the vocabulary, i.e. an unembedding/output projection of size $d_\pi \to V$. Under the same multiply–add accounting, this contributes

$$2Vd_\pi$$

FLOPs per token. Hence

$$C_f(\pi_\theta) = 2L_\pi\big(4d_\pi^2 + 2d_\pi d_{\text{ff}}^\pi + (S+T)d_\pi\big) + 2Vd_\pi.$$

Because the reference model π_{ref} is architecturally identical to the actor,

$$C_f(\pi_{\text{ref}}) = C_f(\pi_\theta).$$

B.1.1.2 **Reward model**: Now, $M = R$. The reward model R_φ processes the full sequence but produces a single scalar score from the final hidden state. Its transformer backbone therefore costs

$$2L_R\big(4d_R^2 + 2d_R d_{\text{ff}}^R + (S+T)d_R\big).$$

The scalar reward head is a projection $d_R \to 1$ applied once per sequence, i.e. $2d_R$ FLOPs per sequence, or equivalently

$$\frac{2d_R}{S+T}$$

FLOPs per token when normalized by sequence length. Thus,

$$C_f(R_\varphi) = 2L_R\big(4d_R^2 + 2d_R d_{\text{ff}}^R + (S+T)d_R\big) + \frac{2d_R}{S+T}.$$

B.1.1.3 **Value model / critic**: Finally, $M = V$. The critic V_ψ outputs one scalar value per token. Its transformer backbone contributes

$$2L_V\big(4d_V^2 + 2d_V d_{\text{ff}}^V + (S+T)d_V\big),$$

and its value head $d_V \to 1$ is applied at every token position, contributing

$$2d_V$$

FLOPs per token. Hence

$$C_f(V_\psi) = 2L_V\big(4d_V^2 + 2d_V d_{\text{ff}}^V + (S+T)d_V\big) + 2d_V.$$

B.1.2 Backward-pass FLOPs

For dense Transformer training, a standard approximation is that the backward pass costs about twice the forward pass, because gradients must be propagated both with respect to activations and with respect to weights. Accordingly, for any trainable model M we use

$$C_b(M) \approx 2\,C_f(M).$$

This is the approximation used throughout the paper when deriving training-stage costs.

B.1.3 Autoregressive generation: prefill and decode

For policy generation, it is important to distinguish prefill from decode.

B.1.3.1 **Prefill**: Given a prompt of length S, the prefill pass processes the prompt once, initializes the KV cache, and produces the logits needed to begin generation. Therefore, its cost is simply the forward cost evaluated at sequence length S:

$$C_{\text{prefill}}(\pi_\theta; S) = 2L_\pi S\big(4d_\pi^2 + 2d_\pi d_{\text{ff}}^\pi + Sd_\pi\big) + 2Vd_\pi \\ L_\pi\left[O(S^2) + O(Sd_\pi)\right].$$

B.1.3.2 **Decode**: After prefill, the model generates tokens autoregressively. At decode step t, the current token attends to a context of length $S+t$, consisting of the prompt plus the t previously generated tokens. Hence the cost of the full decode phase is

$$C_{\text{dec}}(\pi_\theta; S, T) = \sum_{t=1}^{T-1}\left[2L_\pi\big(4d_\pi^2 + 2d_\pi d_{\text{ff}}^\pi + (S+t)d_\pi\big) + 2Vd_\pi\right].$$

Using

$$\sum_{t=1}^{T-1}(S+t) = (T-1)S + \frac{(T-1)T}{2},$$

this can be written as

$$C_{\mathrm{dec}}(\pi_\theta; S, T) = 2(T-1)L_\pi\big(4d_\pi^2 + 2d_\pi d_{\mathrm{ff}}^\pi\big) + 4L_\pi d_\pi\left((T-1)S + \frac{(T-1)T}{2}\right) + 2(T-1)Vd_\pi.$$

B.1.3.3 **Total generation cost**: The total generation cost is therefore

$$C_{\mathrm{gen}}(\pi_\theta; S, T) = C_{\mathrm{prefill}}(\pi_\theta; S) + C_{\mathrm{dec}}(\pi_\theta; S, T).$$

This is the quantity that should be used in the most detailed generation accounting. For more compact asymptotic comparisons, one may further summarize it as

$$C_{\mathrm{gen}}(\pi_\theta; S, T) = O\big((S+T)\, C_f(\pi_\theta; S+T)\big),$$

but this coarser form should be understood as an upper-level abstraction of the explicit prefill+decode decomposition above.

B.1.4 Depth

We now derive the depth terms used in Table 3. Here, depth means the critical-path length under unbounded parallelism.

B.1.4.1 **Single forward pass**: Within one Transformer layer, the dense projections in the FFN contribute reduction depth

$$\log d_M + \log d_{M,\mathrm{ff}}.$$

For the multi-head attention block, the depth comes from the sequential dependencies among the projection, attention, aggregation, and output-projection subcomputations. The query, key, and value projections are independent and can be computed in parallel; therefore, they contribute only $\log d_M$ to the critical path. Once Q and K are available, the attention-score computation within each head reduces over the per-head hidden dimension d_M/h_M, contributing

$$\log(d_M/h_M) = \log d_M - \log h_M.$$

The subsequent weighted aggregation of values reduces over the sequence length, contributing $\log(S+T)$. Finally, the attention output projection maps the concatenated heads back to the hidden dimension and contributes another $\log d_M$. Thus, suppressing constant factors, the multi-head attention depth is

$$\log d_M + \log(d_M/h_M) + \log(S+T) + \log d_M = 3\log d_M + \log(S+T) - \log h_M.$$

Since layers are sequential, the total forward depth scales linearly with L_M. In the simplified cost model used in Table 3, this is summarized as

$$D_f(M) = O\big(L_M[\log d_M + \log(S+T)]\big),$$

plus model-specific output-head terms.

For the **policy/reference model**, the additional vocabulary projection contributes one more logarithmic reduction, yielding

$$D_f(\pi_\theta) = O\big(L_\pi[\log d_\pi + \log(S+T)] + \log d_\pi\big),$$
$$D_f(\pi_{\mathrm{ref}}) = D_f(\pi_\theta).$$

For the **reward** and **value models**, the scalar head contributes an additional $\log d$ term:

$$D_f(R_\varphi) = O\big(L_R[\log d_R + \log(S+T)] + \log d_R\big),$$
$$D_f(V_\psi) = O\big(L_V[\log d_V + \log(S+T)] + \log d_V\big).$$

B.1.4.2 **Tensor-parallel forward depth**: If tensor parallelism of degree P_t shards the hidden dimension, the corresponding reduction depth decreases accordingly, giving

$$D_f^{\mathrm{TP}}(M) = O\big(L_M[\log(d_M/P_t) + \log(S+T)] + \log(d_M/P_t)\big).$$

B.1.4.3 **Backward depth**: Under the same coarse-grained model, the backward pass has a different dependency structure to that of the forward pass. Namely, for any product of activations with model weights, one must consider computing gradients with respect to both the activations and the weights. Computing the gradients with respect to activations has the same asymptotic dependency structure as in the forward pass. Computing the gradients with respect to model weights, however, has to accumulate gradients across the whole batch. Thus

$$D_b(M) = D_f(M) + O(\log(B(S+T))) = O(L(\log d_M + \log(S+T)) + \log(B(S+T))).$$

As gradients with respect to weights do not need to be backpropagated to the previous layer, $\log(B(S+T))$ does not have to be multiplied with L.

B.1.4.4 **Generation depth**: Autoregressive generation has two components: one prefill pass plus a sequence of sequential decode steps. Therefore,

$$D_{\mathrm{gen}}(\pi_\theta; S, T) = D_f(\pi_\theta; S) + \sum_{t=1}^{T-1} D_f(\pi_\theta; S+t).$$

In compact asymptotic form, this is

$$D_{\mathrm{gen}}(\pi_\theta; S, T) = O\big(T \cdot D_f(\pi_\theta; S+T)\big),$$

which makes explicit that the dominant sequential dependence comes from the T decode steps.

B.1.5 Parameter counts

We finally derive the parameter-count and memory expressions used in Table 3.

B.1.5.1 **Policy/reference model**: Ignoring biases and layer-norm parameters, the actor contains

$$|\pi_\theta| = L_\pi(4d_\pi^2 + 2d_\pi d_{\mathrm{ff}}^\pi) + Vd_\pi$$

parameters: the first term comes from the stacked Transformer blocks, and the second from token embeddings / output vocabulary projection. Since the reference model shares the same architecture,

$$|\pi_{\mathrm{ref}}| = |\pi_\theta|.$$

B.1.5.2 **Shared actor–critic backbone**: For a shared actor–critic model with backbone width d_{ac} and L_{ac} layers, plus a policy head of size Vd_{ac} and a scalar value head of size d_{ac}, we obtain

$$|\pi_{\mathrm{ac}}| = L_{\mathrm{ac}}(4d_{\mathrm{ac}}^2 + 2d_{\mathrm{ac}}d_{\mathrm{ff}}^{\mathrm{ac}}) + Vd_{\mathrm{ac}} + d_{\mathrm{ac}}.$$

B.1.5.3 **Reward model**: The reward model consists of a Transformer backbone plus a scalar reward head:

$$|R_\varphi| = L_R(4d_R^2 + 2d_R d_{\text{ff}}^R) + Vd_R + d_R.$$

B.1.5.4 **Value model / critic**: Similarly, the value model consists of a Transformer backbone plus a scalar head:

$$|V_\psi| = L_V(4d_V^2 + 2d_V d_{\text{ff}}^V) + Vd_V + d_V.$$

B.1.5.5 **KV-cache memory**: During generation, each layer stores both keys and values for every processed token. For one rollout of total length $S+T$, this yields

$$M_{\text{KV}} = 2(S+T)L_\pi d_\pi,$$

where the factor 2 accounts for keys and values. If all BK rollouts are generated concurrently, the corresponding peak KV-cache memory is

$$BK \cdot M_{\text{KV}}.$$

If fewer rollouts are generated concurrently, the peak memory decreases proportionally, at the cost of reduced concurrency.

B.1.5.6 **Optimizer state**: For trainable models, optimizer-state memory is approximated by a factor of four times the parameter count, corresponding to parameters, gradients, and Adam first and second moments.

B.1.5.7 **Training activations and inference buffers**: For a trainable model M, backpropagation requires intermediate activations from the forward pass. Under the leading-order activation model used throughout the paper, we retain one hidden-state-sized contribution per layer and processed token:

$$M_{\text{Act}}(M) = \Theta(L_M d_M)$$

per token. Hence, for a batch of Q sequences of length $S+T$,

$$M_{\text{Act,total}}(M) = \Theta(Q(S+T)L_M d_M)\,.$$

This expression suppresses constant-factor storage for Q/K/V tensors, FFN intermediates, normalization and residual temporaries, and other implementation-specific buffers, and assumes no activation checkpointing.

For a frozen forward-only model, intermediate layer buffers need not be retained for backpropagation and can be reused across layers. Assuming the attention matrix is not fully materialized, the leading-order inference buffer is

$$M_{\text{Inf}}(M) = \Theta(d_M)$$

per processed token, or

$$M_{\text{Inf,total}}(M) = \Theta(Q(S+T)d_M)$$

for Q sequences. Model parameters and autoregressive KV-cache storage are accounted for separately.

B.2 Results for RL-LLM Frameworks

We start with derivations for results in Section 2.5.

B.2.1 Online Frameworks

Online methods such as PPO and GRPO repeatedly execute the full Generation → Assessment → Training loop, so their per-iteration cost is the sum of these three stages.

B.2.1.1 **Generation**: Generation is dominated by autoregressive sampling with the current policy π_θ. For each of the B prompts, the policy generates K candidate responses, yielding BK rollouts in total. The cost of one rollout should be decomposed into a *prefill* pass over the prompt of length S and a sequence of T *decode* steps:

$$C_{\text{gen}}(\pi_\theta) = C_{\text{prefill}}(\pi_\theta; S) + \sum_{t=1}^{T-1} C_{\text{decode}}(\pi_\theta; S+t).$$

Here, the prefill initializes the KV cache from the prompt, while decode step t produces the next token while attending to a context of length $S+t$. Thus, the total generation work over the whole batch is

$$W_{\text{gen}} = \Theta\big(BK \cdot C_{\text{gen}}(\pi_\theta)\big).$$

The critical path is determined by the autoregressive dependence across generated tokens: token $t+1$ cannot be produced before token t has been generated. Therefore, the generation depth scales linearly in T:

$$D_{\text{gen}} = \Theta\big(T \cdot D_f(\pi_\theta)\big).$$

This sequential dependence makes generation the primary wall-clock bottleneck in online RL-LLM pipelines.

B.2.1.2 **Assessment**: After generation, the completed prompt–response pairs are evaluated by auxiliary models such as the reward model R_φ, the critic V_ψ, and the reference policy π_{ref}. For PPO, all three are used; for GRPO, the critic is omitted. Each model processes the full sampled sequences in teacher-forced, non-autoregressive mode, so the work per model scales as

$$W_{\text{assess}}(M) = \Theta\big(BK(S+T)\,C_f(M)\big).$$

Unlike generation, this stage does *not* incur a factor-T sequential dependence across output tokens. Its critical path is therefore the depth of a single forward pass through the corresponding model:

$$D_{\text{assess}}(M) = \Theta\big(D_f(M)\big).$$

If the assessment models are placed on disjoint device groups and executed concurrently, the stage depth is the maximum of their forward-pass depths; if they are co-located and executed sequentially, these depths add.

B.2.1.3 **Training**: In the training stage, the policy and, for actor–critic methods, the critic are updated by backpropagation over the sampled sequences. Since training is performed in teacher-forced mode on the complete prompt–response sequences, the work per updated model and per optimization epoch scales as

$$W_{\text{train}}(M) = \Theta\big(BK(S+T)\,C_{tr}(M)\big).$$

Again, there is no autoregressive factor-T dependence across tokens in the critical path; instead, the depth is that of a single backward pass:

$$D_{\text{train}}(M) = \Theta\big(D_{tr}(M)\big).$$

Thus, PPO incurs updates for both π_θ and V_ψ, whereas GRPO updates only π_θ.

B.2.2 Offline Frameworks

Offline frameworks such as DPO decouple generation from optimization. Instead of sampling fresh rollouts inside the training loop, they optimize the policy π_θ on a static dataset of preference pairs (x, y_w, y_l). As a result, the inner loop contains only batched forward and backward passes over fixed data.

For a batch of B_{pairs} preference pairs, the forward work is

$$W_{\text{forward}} = \Theta\big(B_{\text{pairs}}(S+T)\,[C_f(\pi_\theta) + C_f(\pi_{\text{ref}})]\big),$$

while the backward work is

$$W_{\text{backward}} = \Theta\big(B_{\text{pairs}}(S+T)\,C_b(\pi_\theta)\big).$$

The critical path is therefore the depth of a single batched forward/backward evaluation rather than an autoregressive decode chain:

$$D_{\text{DPO}} = \Theta\big(\max\{D_f(\pi_\theta), D_f(\pi_{\text{ref}}), D_b(\pi_\theta)\}\big).$$

B.3 Results for Intra-Model Parallelism

We now derive the results for Section 3.7 (Tables 5, 6, 7, and 8). The goal is to expose how standard intra-model parallelism strategies transform the work, depth, and memory terms derived in Appendix B.1.

B.3.1 Baseline Cost Model

Appendix B.1 derives the forward and backward costs of Transformer-based models in terms of layer count, hidden dimension, FFN width, vocabulary size, and sequence length. For the present intra-model analysis, we consider a dense Transformer training iteration with L layers, hidden dimension d, FFN dimension d_{ff}, vocabulary size V, prompt length S, response length T, and batch size B. The complete teacher-forced training sequence therefore has length $S+T$. Since Tables 5–8 analyze a generic Transformer training step rather than a role-specific actor, critic, reward, or reference model, we ignore small model-specific head differences and use the common dense Transformer body.

From Appendix B.1, the forward-pass cost per token of the dense Transformer body is

$$C_f = 2L\left(4d^2 + 2dd_{\text{ff}} + (S+T)d\right),$$

where $4d^2$ comes from the query, key, value, and output projections, $2dd_{\text{ff}}$ from the two FFN projections, and $(S+T)d$ from the sequence-dependent attention score/value operations. Using the same approximation as in Table 3,

$$C_b \approx 2C_f.$$

Thus, the forward-plus-backward work per token is

$$C_f + C_b \approx 3C_f = 6L\left(4d^2 + 2dd_{\text{ff}} + (S+T)d\right).$$

For $B(S+T)$ processed tokens, the baseline work is

$$W_{\text{base}} = 6B(S+T)L\left(4d^2 + 2dd_{\text{ff}} + (S+T)d\right),$$

which gives the asymptotic summary

$$W_{\text{base}} = O\big(B(S+T)L(d^2 + (S+T)d)\big)$$

in the parameter-dominated regime.

We next refine the depth expression. A single forward Transformer layer has reduction depth

$$4\log d + \log d_{\text{ff}} + \log(S+T) - \log h,$$

where h is the number of attention heads. Hence,

$$D_f = L\left(4\log d + \log d_{\text{ff}} + \log(S+T) - \log h\right).$$

The activation-gradient path in the backward pass has a comparable layerwise dependency structure. However, trainable projections also require parameter gradients. For a projection $Y = XW$, with the batch and sequence dimensions flattened into $B(S+T)$ rows,

$$\nabla_X \mathcal{L} = \nabla_Y \mathcal{L}\, W^\top, \qquad \nabla_W \mathcal{L} = X^\top \nabla_Y \mathcal{L}.$$

The second product accumulates contributions to each shared parameter over the $B(S+T)$ batch-token positions and therefore introduces a reduction of depth

$$\log(B(S+T)).$$

These parameter-gradient reductions branch from the activation-gradient backward path and can overlap with propagation through preceding layers; therefore they are not multiplied by L. We conservatively account for this additional critical-path contribution as

$$D_b = O(L[\log d + \log(S+T)] + \log(B(S+T))).$$

Under the explicit reduction-depth model used in Tables 7–8, we therefore use

$$\begin{aligned} D_{\text{base}} &= 2L\left(4\log d + \log d_{\text{ff}} + \log(S+T) - \log h\right) \\ &\quad + \log(B(S+T)) \\ &= 2L\log\left(\frac{d^4 d_{\text{ff}}(S+T)}{h}\right) + \log(B(S+T)). \end{aligned}$$

The corresponding asymptotic form is

$$D_{\text{base}} = O(L[\log d + \log(S+T)] + \log(B(S+T))).$$

Finally, the memory model follows the same parameter-counting convention as Appendix B.1. The dominant model-state terms are parameters, gradients, and Adam first and second moments, giving a factor of four over the parameter count:

$$4L(4d^2 + 2dd_{\text{ff}}) + 4Vd = L(16d^2 + 8dd_{\text{ff}}) + 4Vd.$$

Stored activations contribute

$$B(S+T)Ld.$$

Therefore,

$$M_{\text{base}} = L(16d^2 + 8dd_{\text{ff}}) + 4Vd + B(S+T)Ld,$$

or asymptotically

$$M_{\text{base}} = O\big(Ld^2 + Vd + B(S+T)Ld\big).$$

The remaining derivations apply each intra-model parallelism strategy as a partitioning transformation of these baseline work, depth, and memory terms.

B.3.2 Data Parallelism

Data parallelism of degree P_d partitions the batch dimension. Each rank processes B/P_d sequences but stores a full model replica. Therefore, the per-rank work is

$$\begin{aligned} W_{\mathrm{DP,rank}} &= \frac{1}{P_d} W_{\mathrm{base}} \\ &= 6\frac{B}{P_d}(S+T)L\left(4d^2 + 2dd_{\mathrm{ff}} + (S+T)d\right), \end{aligned}$$

while the global work remains

$$W_{\mathrm{DP,global}} = W_{\mathrm{base}}.$$

The layerwise forward/backward dependency is unchanged. Locally, however, each rank forms parameter gradients from only $\frac{B}{P_d}(S+T)$ batch-token positions. Hence,

$$D_{\mathrm{DP,rank}} = 2L\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{h}\right) + \log\left(\frac{B}{P_d}(S+T)\right).$$

The globally aggregated parameter gradient still depends on all $B(S+T)$ positions, so, ignoring communication latency while retaining the logical gradient-reduction dependency,

$$D_{\mathrm{DP,global}} = D_{\mathrm{base}}.$$

The per-rank activation memory is reduced by P_d, but model-state memory is replicated:

$$M_{\mathrm{DP,rank}} = L(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd + \frac{B}{P_d}(S+T)Ld.$$

At the global level, the activation term sums back to $B(S+T)Ld$, while the model-state term is replicated P_d times:

$$M_{\mathrm{DP,global}} = P_d\big[L(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd\big] + B(S+T)Ld.$$

Thus, DP reduces per-rank work and activation memory, while the full logical model update retains the global batch-gradient dependency.

B.3.3 Pipeline Parallelism

Pipeline parallelism of degree P_p partitions the layer dimension. Each pipeline stage stores and computes approximately L/P_p layers. Applying

$$L \mapsto \frac{L}{P_p}$$

to the baseline per-rank formulas gives

$$W_{\mathrm{PP,rank}} = 6B(S+T)\frac{L}{P_p}\left(4d^2 + 2dd_{\mathrm{ff}} + (S+T)d\right),$$

and

$$D_{\mathrm{PP,rank}} = 2\frac{L}{P_p}\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{h}\right) + \log(B(S+T)).$$

The layer-dependent memory terms are similarly divided across pipeline stages:

$$M_{\mathrm{PP,rank}} = \frac{L}{P_p}(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd + B(S+T)\frac{L}{P_p}d.$$

The vocabulary term $4Vd$ is written separately because embeddings and output heads are commonly pinned to boundary stages rather than evenly partitioned across all stages.

Globally, pipeline parallelism preserves total work and total model-state memory up to boundary-stage effects:

$$W_{\mathrm{PP,global}} = W_{\mathrm{base}}, \qquad M_{\mathrm{PP,global}} \approx M_{\mathrm{base}}.$$

For global depth, a microbatch still traverses all P_p stages and hence all L layers. Therefore,

$$D_{\mathrm{PP,global}} = D_{\mathrm{base}}.$$

Realized runtime additionally depends on the microbatch schedule, pipeline fill/drain bubbles, and whether one counts the latency of a single microbatch or steady-state throughput. Those hardware-schedule effects are outside the present work–depth abstraction.

B.3.4 Tensor Parallelism

Tensor parallelism of degree P_t partitions hidden-dimensional operators within each layer. In the simplified work–depth model, this shards the dominant GEMM work, model-state memory, and activation memory by P_t. Thus,

$$W_{\mathrm{TP,rank}} = 6B(S+T)L\left(\frac{4d^2 + 2dd_{\mathrm{ff}} + (S+T)d}{P_t}\right),$$

and

$$M_{\mathrm{TP,rank}} = \frac{L(16d^2 + 8dd_{\mathrm{ff}})}{P_t} + \frac{4Vd}{P_t} + \frac{B(S+T)Ld}{P_t}.$$

The hidden-dimensional reductions are correspondingly shortened on each tensor shard, while the parameter-gradient accumulation still spans all $B(S+T)$ batch-token positions. Under the explicit model used in the tables,

$$D_{\mathrm{TP}} = 2L\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{P_t^2 h}\right) + \log(B(S+T)).$$

Asymptotically,

$$D_{\mathrm{TP}} = O\big(L[\log(d/P_t) + \log(S+T)] + \log(B(S+T))\big).$$

Globally, TP preserves total work and total memory order:

$$W_{\mathrm{TP,global}} = W_{\mathrm{base}}, \qquad M_{\mathrm{TP,global}} \approx M_{\mathrm{base}},$$

ignoring communication buffers and collective overheads. Thus, TP primarily reduces per-device pressure and hidden-dimensional critical-path reductions, at the cost of layer-wise communication not modeled in these tables.

B.3.5 Context Parallelism

Context parallelism of degree P_c partitions the sequence dimension. Each rank owns a local query/context shard of length $(S+T)/P_c$. Each local query, however, still depends on keys and values from the global context of length $S+T$. Thus, CP partitions sequence-local work and activations without replacing the global attention-reduction length $S+T$ by $(S+T)/P_c$.

The per-rank work is

$$W_{\mathrm{CP,rank}} = 6B\frac{S+T}{P_c}L\left(4d^2 + 2dd_{\mathrm{ff}} + (S+T)d\right).$$

The per-rank memory is

$$M_{\mathrm{CP,rank}} = L(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd + B\frac{S+T}{P_c}Ld.$$

Under the computation-only depth abstraction, the attention part retains the global $S+T$ reduction dependency. The local parameter-gradient reduction is over $B(S+T)/P_c$ token instances, giving

$$D_{\mathrm{CP,rank}} = 2L\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{h}\right) + \log\left(\frac{B(S+T)}{P_c}\right).$$

Globally, the partial parameter gradients span all sequence partitions, so

$$D_{\mathrm{CP,global}} = D_{\mathrm{base}}.$$

Distributed attention additionally introduces communication-round dependencies, which are outside the present FLOP-based depth abstraction.

Globally, model parameters are replicated across context partitions, while the activation term sums back to the original order:

$$W_{\mathrm{CP,global}} = W_{\mathrm{base}},$$

$$M_{\mathrm{CP,global}} = P_c\big[L(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd\big] + B(S+T)Ld.$$

Thus, CP is primarily a sequence-work, memory, and long-context feasibility technique; it does not reduce the global exact-attention reduction from $S+T$ to $(S+T)/P_c$.

B.3.6 Expert Parallelism

Expert parallelism applies to MoE layers. Suppose the model has E experts, E_a active experts per token, expert hidden dimension d_e, and expert-parallel degree P_e.

For the isolated EP results in Tables 5–8, we assume that EP is the only partitioning mechanism: the non-expert Transformer computation is replicated across the P_e ranks, while expert FFN parameters and routed expert computation are partitioned across them. Thus, no additional DP, TP, PP, or CP partitioning is assumed outside the expert FFN.

The per-rank work is therefore

$$W_{\mathrm{EP,rank}} = 6B(S+T)L\left(4d^2 + \frac{2E_a dd_e}{P_e} + (S+T)d\right).$$

Because the dense/shared computation is replicated, while expert computation is partitioned, global executed work is

$$W_{\mathrm{EP,global}} = 6B(S+T)L\left(P_e[4d^2 + (S+T)d] + 2E_a dd_e\right).$$

Since experts are selected inside the same layer position, EP does not remove the dense Transformer critical path. The dense parameter-gradient reduction also spans all $B(S+T)$ token instances. Thus,

$$D_{\mathrm{EP}} = 2L\log\left(\frac{d^4 d_e(S+T)}{h}\right) + \log(B(S+T)),$$

or asymptotically,

$$D_{\mathrm{EP}} = O(L[\log d + \log(S+T)] + \log(B(S+T))).$$

The per-rank memory is

$$M_{\mathrm{EP,rank}} = L\left(16d^2 + \frac{8Edd_e}{P_e}\right) + 4Vd + B(S+T)Ld.$$

Globally, the dense model state and dense activations are replicated across the P_e ranks, while expert parameters sum across expert partitions:

$$M_{\mathrm{EP,global}} = P_e\left(16Ld^2 + 4Vd + B(S+T)Ld\right) + 8LEdd_e,$$

up to the same optimizer-state convention used in the baseline memory model.

B.3.7 3D Parallelism

3D parallelism combines data, pipeline, and tensor parallelism with degrees (P_d, P_p, P_t). It applies the substitutions

$$B \mapsto \frac{B}{P_d}, \qquad L \mapsto \frac{L}{P_p}, \qquad d\text{-sharded GEMM terms} \mapsto \frac{1}{P_t}$$

to the corresponding baseline terms. Therefore, the per-rank work is

$$W_{\mathrm{3D,rank}} = 6\frac{B}{P_d}(S+T)\frac{L}{P_p}\left(\frac{4d^2 + 2dd_{\mathrm{ff}} + (S+T)d}{P_t}\right).$$

The per-rank depth combines pipeline and tensor reductions, while the local parameter-gradient reduction spans the $\frac{B}{P_d}(S+T)$ batch-token positions assigned to the data-parallel replica:

$$D_{\mathrm{3D,rank}} = 2\frac{L}{P_p}\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{P_t^2 h}\right) + \log\left(\frac{B}{P_d}(S+T)\right),$$

or asymptotically,

$$D_{\mathrm{3D,rank}} = O\left(\frac{L}{P_p}[\log(d/P_t) + \log(S+T)] + \log\left(\frac{B}{P_d}(S+T)\right)\right).$$

The memory expression combines layer partitioning, tensor partitioning, and batch partitioning:

$$M_{\mathrm{3D,rank}} = \frac{L(16d^2 + 8dd_{\mathrm{ff}})}{P_p P_t} + \frac{4Vd}{P_t} + \frac{B(S+T)Ld}{P_d P_p P_t}.$$

Globally, the dominant work remains the baseline work, while model-state memory is replicated across data-parallel groups:

$$W_{\mathrm{3D,global}} = W_{\mathrm{base}},$$

$$M_{\mathrm{3D,global}} = P_d\left[L(16d^2 + 8dd_{\mathrm{ff}}) + 4Vd\right] + B(S+T)Ld.$$

The global depth restores the full L-layer pipeline path and the full batch-gradient dependency, while retaining the tensor-parallel hidden reduction:

$$D_{\mathrm{3D,global}} = 2L\log\left(\frac{d^4 d_{\mathrm{ff}}(S+T)}{P_t^2 h}\right) + \log(B(S+T)),$$

or

$$D_{\mathrm{3D,global}} = O\big(L[\log(d/P_t) + \log(S+T)] + \log(B(S+T))\big).$$

This explains the main role of 3D parallelism in the tables: it gives strong per-device memory reduction because it simultaneously partitions batch, layers, and hidden-dimensional computation. Its realized runtime, however, depends on communication, pipeline scheduling, and microbatching, which are deliberately outside the work–depth–memory abstraction used here.

B.3.8 5D Parallelism

We finally consider a combined configuration using data, pipeline, tensor, context, and expert parallelism with degrees $(P_d, P_p, P_t, P_c, P_e)$, where

$$P_d P_p P_t P_c P_e = N.$$

An important distinction from the isolated EP analysis above is required. The isolated EP row is intentionally a conceptual abstraction designed to expose the effect of expert partitioning alone: only expert FFNs are partitioned across the P_e ranks, while the same shared dense Transformer computation is executed on the same token set by every EP rank. Consequently, its global work contains P_e copies of the shared dense arithmetic. This should not be interpreted as the execution strategy necessarily used in a practical multidimensional MoE system.

For the 5D configuration, we instead model a practically motivated coupled execution in which the ranks participating in the EP dimension also own disjoint *source-token* shards for the shared Transformer path. Expert parallelism itself determines where expert parameters reside; the additional source-token partition specifies where tokens reside before expert dispatch. These are distinct notions. Formally, if $\mathcal{U}$ denotes the $B(S+T)$ batch-token positions of the global training invocation, then we assume

$$\mathcal{U} = \bigsqcup_{r_d=1}^{P_d} \bigsqcup_{r_c=1}^{P_c} \bigsqcup_{r_e=1}^{P_e} \mathcal{U}_{r_d,r_c,r_e},$$

with approximately balanced partitions

$$|\mathcal{U}_{r_d,r_c,r_e}| \approx \frac{B(S+T)}{P_d P_c P_e}.$$

Thus, a shared dense parameter may be replicated across the EP dimension, but each source token is processed by the shared path on only one EP rank. After routing, token representations are redistributed by the expert-parallel all-to-all to the ranks hosting the selected experts, and expert outputs are subsequently returned to the corresponding source-token ranks. We assume balanced routing, so the resulting token–expert assignments are approximately uniformly distributed across the P_e expert ranks. Tensor parallelism is applied to both shared dense and expert GEMMs.

Under these assumptions, the per-rank work is

$$W_{\text{5D,rank}} = 6\frac{B}{P_d P_e}\frac{S+T}{P_c}\frac{L}{P_p}\left(\frac{4d^2 + 2E_a d d_e + (S+T)d}{P_t}\right).$$

The factors have distinct origins: P_d partitions the batch, P_c partitions source sequence positions, P_e further partitions source-token ownership across the expert-parallel ranks, P_p partitions layers, and P_t partitions intra-layer matrix operations. The $1/P_e$ factor is therefore applied to the source-token workload rather than separately to the expert term; under balanced routing, each EP rank already receives approximately $1/P_e$ of the global token–expert assignments.

Summing executed arithmetic across all $N = P_d P_p P_t P_c P_e$ ranks gives

$$W_{\text{5D,global}} = 6B(S+T)L\left(4d^2 + 2E_a d d_e + (S+T)d\right).$$

In particular, the shared dense term is not multiplied by P_e, unlike in the isolated EP abstraction. The reason is that the execution mapping assigns disjoint source-token subsets to the EP ranks, so the same dense token computation is not redundantly executed on every EP rank. This illustrates the distinction between *parameter replication* and *arithmetic replication*: shared dense parameters may remain replicated across the EP dimension even though the corresponding FLOPs are evaluated on disjoint token subsets.

The per-rank depth is

$$D_{\text{5D,rank}} = 2\frac{L}{P_p}\log\left(\frac{d^4 d_e (S+T)}{P_t^2 h}\right) + \log\left(\frac{B(S+T)}{P_d P_e P_c}\right).$$

The first term combines pipeline and tensor partitioning of the layerwise critical path while retaining the global attention span $S+T$. The second term is the local parameter-gradient reduction over the source batch-token positions assigned to the rank. As in the other tables, communication, including the expert dispatch/combine all-to-all and distributed-attention collectives, is outside the arithmetic-depth abstraction.

At the global level, the complete invocation traverses all L pipeline layers and its parameter gradients depend on all $B(S+T)$ batch-token positions. Hence,

$$D_{\text{5D,global}} = 2L\log\left(\frac{d^4 d_e (S+T)}{P_t^2 h}\right) + \log(B(S+T)).$$

The corresponding per-rank memory is

$$M_{\text{5D,rank}} = \frac{16Ld^2}{P_p P_t} + \frac{8LEdd_e}{P_p P_t P_e} + \frac{4Vd}{P_t} + \frac{B(S+T)Ld}{P_d P_e P_c P_p P_t}.$$

Shared dense model state is partitioned by PP and TP but remains replicated across the DP, CP, and EP dimensions. Expert model state is additionally partitioned across P_e, while the leading-order activation term follows the combined source-token, layer, and tensor partitioning assumed above. Consequently, aggregate global memory is

$$\begin{aligned} M_{\text{5D,global}} =& P_d P_c P_e \left(16Ld^2 + 4Vd\right) \\ &+ 8P_d P_c LEdd_e + B(S+T)Ld, \end{aligned}$$

up to the same boundary-stage and embedding-placement conventions used for PP and TP.

Isolated EP provides a clean limiting case in which only expert FFNs are partitioned, making the cost of replicated shared computation explicit. The 5D configuration instead captures a more practically relevant multidimensional MoE execution in which expert placement is combined with source-token distribution and other intra-model parallelism dimensions. As a result, shared dense arithmetic is partitioned rather than replicated across the EP dimension, whereas replicated shared *state* can still contribute a factor P_e to global memory.

B.4 Results for Inter-Model Parallelism

We provide derivations for results in Section 4.6 and in Table 9. The analysis uses the building blocks from Table 3: forward costs $C_f(\cdot)$, backward costs $C_b(\cdot)$, autoregressive generation cost $C_{\text{gen}}(\cdot)$, forward/backward depths $D_f(\cdot)$, $D_b(\cdot)$, and generation depth $D_{\text{gen}}(\cdot)$. Communication, kernel efficiency, and scheduling overheads are outside the work–depth abstraction; they are represented only indirectly through memory buffers such as M_{reshard} and shadow copies M_{sh}.

B.4.1 Stage Primitives

For a PPO-style iteration with B prompts, K rollouts per prompt, prompt length S, and response length T, Generation invokes the actor autoregressively. Using the prefill–decode decomposition from Table 3,

$$\begin{aligned} W_G &= BK\, C_{\mathrm{gen}}(\pi_\theta), \\ D_G &= D_{\mathrm{gen}}(\pi_\theta), \\ M_G &= |\pi_\theta| + BK\, M_{\mathrm{KV}}. \end{aligned}$$

This assumes the rollout-time actor log-probabilities are stored during generation. If an implementation recomputes old actor log-probabilities in a teacher-forced pass, an additional $BK(S+T)C_f(\pi_\theta)$ work term and $D_f(\pi_\theta)$ depth term should be added to Assessment or Training, depending on where the recomputation is performed.

Assessment consists of teacher-forced forward passes through π_{ref}, R_φ, and V_ψ:

$$W_A = BK(S+T)[C_f(\pi_{\mathrm{ref}}) + C_f(R_\varphi) + C_f(V_\psi)].$$

If these models are co-located and executed sequentially, the depth is

$$D_A^{\Sigma} = D_f(\pi_{\mathrm{ref}}) + D_f(R_\varphi) + D_f(V_\psi).$$

If they are placed on disjoint device groups and executed concurrently, the depth becomes

$$D_A^{\max} = \max\{D_f(\pi_{\mathrm{ref}}), D_f(R_\varphi), D_f(V_\psi)\}.$$

The corresponding co-located memory footprint is

$$M_A^{\Sigma} = |\pi_{\mathrm{ref}}| + |R_\varphi| + |V_\psi| + BK(S+T)M_{\mathrm{Inf}}.$$

Training updates the actor and critic. For a trainable model M, we define

$$\begin{aligned} C_{\mathrm{tr}}(M) &:= C_f(M) + C_b(M) \approx 3C_f(M), \\ D_{\mathrm{tr}}(M) &:= D_f(M) + D_b(M), \end{aligned}$$

since a parameter update requires a forward pass followed by backpropagation. Thus,

$$W_T = BK(S+T)[C_{\mathrm{tr}}(\pi_\theta) + C_{\mathrm{tr}}(V_\psi)].$$

Sequential co-located training has depth

$$D_T^{\Sigma} = D_{\mathrm{tr}}(\pi_\theta) + D_{\mathrm{tr}}(V_\psi),$$

whereas disaggregated actor/critic training has depth

$$D_T^{\max} = \max\{D_{\mathrm{tr}}(\pi_\theta), D_{\mathrm{tr}}(V_\psi)\}.$$

Under the Adam memory model without activation checkpointing, the co-located training footprint is

$$M_T^{\Sigma} = 4(|\pi_\theta| + |V_\psi|) + BK(S+T)(L_\pi d_\pi + L_V d_V).$$

For disaggregated training, the peak per device group is

$$\begin{aligned} M_T^{\max} = \max\{&4|\pi_\theta| + BK(S+T)L_\pi d_\pi, \\ &4|V_\psi| + BK(S+T)L_V d_V\}. \end{aligned}$$

B.4.2 Baseline Co-Located Execution

The baseline uses separate actor and critic models and a fully co-located device group. All stages execute sequentially. Therefore,

$$\begin{aligned} W_{\mathrm{base}} &= W_G + W_A + W_T, \\ D_{\mathrm{base}} &= D_G + D_A^{\Sigma} + D_T^{\Sigma}, \end{aligned}$$

and

$$M_{\mathrm{base}} = \max\{M_G, M_A^{\Sigma}, M_T^{\Sigma}\}.$$

The maximum appears because the stages are temporally multiplexed on the same device group: the peak is the largest stage footprint, not the sum over all stages. If an implementation keeps all models resident simultaneously, then the persistent part of the memory expression should instead sum the resident model states.

B.4.3 Shared Actor–Critic

With a shared actor–critic model π_{ac}, the actor and value heads share a Transformer backbone. Generation uses π_{ac} as the policy:

$$W_G^{\mathrm{ac}} = BK\, C_{\mathrm{gen}}(\pi_{\mathrm{ac}}), \qquad D_G^{\mathrm{ac}} = D_{\mathrm{gen}}(\pi_{\mathrm{ac}}).$$

Assessment no longer requires an independent critic model; instead, value estimates are produced by the shared model:

$$W_A^{\mathrm{ac}} = BK(S+T)[C_f(\pi_{\mathrm{ref}}) + C_f(R_\varphi) + C_f(\pi_{\mathrm{ac}})].$$

Training performs one forward pass followed by backpropagation through the shared backbone:

$$W_T^{\mathrm{ac}} = BK(S+T)C_{\mathrm{tr}}(\pi_{\mathrm{ac}}).$$

Thus, shared actor–critic is the only inter-model configuration in Table 9 that reduces global work relative to the separate-backbone PPO baseline. Its training memory becomes

$$M_T^{\mathrm{ac}} = 4|\pi_{\mathrm{ac}}| + BK(S+T)L_{\mathrm{ac}}d_{\mathrm{ac}},$$

up to the small policy/value head activations. The benefit is largest when $|V_\psi|$ is comparable to $|\pi_\theta|$; it is smaller when the critic is already much smaller than the actor.

B.4.4 Disaggregated Placement

Disaggregated placement assigns actor, reference, reward, and critic operators to separate device groups. It does not change total work:

$$W_{\mathrm{disagg}} = W_G + W_A + W_T.$$

It changes depth by replacing independent sequential subcomputations with parallel fork–join regions:

$$D_{\mathrm{disagg}} = D_G + D_A^{\max} + D_T^{\max}.$$

It also changes memory from co-resident memory to per-group memory:

$$M_{\mathrm{disagg}} = \max\{M_G, M_{\pi_{\mathrm{ref}}}, M_{R_\varphi}, M_{V_\psi}, M_{T,\pi}, M_{T,V}\}.$$

This captures the main memory advantage of disaggregation: reward-model devices need not hold actor or critic state, and actor-training devices need not hold frozen reward/reference models. Disaggregation alone does not remove the Generation $\rightarrow$ Assessment $\rightarrow$ Training dependency; it only enables concurrency where the dataflow graph has independent branches.

B.4.5 Hybrid Execution and Resharding

Hybrid execution allows the same model to use different intra-model layouts in different stages. Let D_G^{inf} and M_G^{inf} denote Generation depth and memory under an inference-oriented layout, and let D_T^{train} and M_T^{train} denote Training depth and memory under a training-oriented layout such as ZeRO/FSDP or 3D parallelism. The FLOP work is unchanged:

$$W_{\text{hyb}} = W_G + W_A + W_T,$$

but the stage depths become layout-specific:

$$D_{\text{hyb}} = D_G^{\text{inf}} + D_A^{\text{inf},\Sigma} + D_T^{\text{train}},$$

assuming the same device group is temporally multiplexed and therefore provides no inter-model concurrency by itself.

The cost of hybrid execution is resharding. If two invocations of the same model use incompatible layouts, weights or optimizer state must be redistributed. This does not change FLOP work in the work–depth table, but it affects realized runtime and may require an additional transient buffer:

$$M_{\text{hyb}} = \max\{M_G^{\text{inf}}, M_A^{\text{inf}}, M_T^{\text{train}}\} + M_{\text{reshard}}.$$

Hybrid execution is useful only when the per-stage gains from specialized layouts exceed the resharding and orchestration costs.

B.4.6 Stage Fusion

Stage fusion preserves total work:

$$W_{\text{fusion}} = W_G + W_A + W_T.$$

Its effect is on depth. Inter-stage fusion streams completed generations into Assessment, so the fused Generation–Assessment depth is

$$D_{G/A}^{\text{fused}} = \max\{D_G, D_A^{\max}\}$$

under ideal overlap. Training remains downstream, so

$$D_{\text{fusion}} = \max\{D_G, D_A^{\max}\} + D_T^{\max}.$$

When generation dominates assessment, this simplifies to

$$D_{\text{fusion}} \approx D_G + D_T^{\max}.$$

Intra-stage fusion similarly replaces sequential actor/critic training pipelines with a max term when the two pipelines can be overlapped:

$$D_T^{\Sigma} \to D_T^{\max}.$$

The memory footprint may increase because multiple stage fragments are live simultaneously:

$$M_{\text{fusion}} = \max\{M_{G/A}^{\text{fused}}, M_T^{\max}\}.$$

Here $M_{G/A}^{\text{fused}}$ includes the live generation KV cache, the assessment model states, and any queues or communication buffers needed to stream completed samples.

B.4.7 Asynchronous Execution

Bounded asynchrony overlaps generation/assessment of one iteration with training of another. Total work is unchanged:

$$W_{\text{async}} = W_G + W_A + W_T.$$

The steady-state recurrence depth becomes

$$D_{\text{async}} = \max\{D_G + D_A^{\max}, D_T^{\max}\},$$

assuming disaggregated generation/assessment and training groups. If assessment is also fused with generation, then the first term can be replaced by $\max\{D_G, D_A^{\max}\}$.

Asynchrony requires parameter snapshots. If a trainable model is read by a stale consumer on a disjoint device group, the consumer needs a stable inference-side copy while the training side updates another copy. We denote this additional persistent copy by

$$M_{\text{sh}}(m).$$

For actor asynchrony, $M_{\text{sh}}(\pi_\theta)$ is typically a BF16, possibly quantized, inference copy of the actor. Thus,

$$M_{\text{async}} = \max\{M_{G/A}^{\text{async}} + M_{\text{sh}}(\pi_\theta), M_T^{\max}\}.$$

This exposes the memory trade-off: asynchrony reduces depth by overlapping iterations, but it may require roughly one extra parameter copy for each asynchronously consumed trainable model. This is not necessarily a full $2\times$ increase in total memory, because optimizer states remain on the training side and the shadow copy may be lower precision or sharded, but it must be accounted for.

B.4.8 Combined Configuration

The combined configuration composes the strongest ingredients: stage-specific intra-model sharding, disaggregated model placement, stage fusion, and bounded asynchrony. The total FLOP work is still

$$W_{\text{comb}} = W_G + W_A + W_T,$$

unless the model structure is also changed, for example by using a shared actor–critic backbone. With ideal fusion and bounded asynchrony, the recurrence depth is

$$D_{\text{comb}} = \max\{D_G^{\text{TP}}, D_T^{\text{shard}}\},$$

where D_G^{TP} denotes actor generation depth under an inference-oriented sharded layout, and D_T^{shard} denotes the depth of the sharded training layout. A more conservative expression keeps the unhidden assessment tail:

$$D_{\text{comb}} = \max\{\max(D_G^{\text{TP}}, D_A^{\max}), D_T^{\text{shard}}\}.$$

The memory expression is

$$M_{\text{comb}} = \max\{M_G^{\text{TP}} + M_{\text{sh}}(\pi_\theta), M_{\pi_{\text{ref}}}, M_{R_\varphi}, M_{V_\psi}, M_T^{\text{shard}}\}.$$

Thus, the combined strategy gives the smallest idealized depth and peak per-device-group memory in Table 9, but only under sufficient hardware, careful scheduling, and acceptable staleness.