Analysis

When Pretraining, SFT and RL Share One Likelihood Objective: Recipe Engineering Collapses into Data-Mixture Design

HX Research

HX Research · Deep Analysis · 3 November 2026

Foundation-model training has long been organized as staged pretraining, midtraining, supervised fine-tuning (SFT) and reinforcement learning (RL), with algorithms, data and infrastructure split accordingly. Recent work shows that suitable RL objectives can be rewritten as maximum-likelihood fitting to model-generated target distributions, so that they share one likelihood training rule with pretraining, midtraining and SFT; at the same time, pretraining gains of weight normalization over weight decay are driven mainly by the induced effective learning rate (ELR) trajectory rather than by normalization itself. We argue that staged pipelines are being rewritten as "a single likelihood objective plus data mixture and ELR trajectories," and that enterprises should manage reproducible mixtures, ELR schedules and recipe versions—not nominal stage labels. We provide a joint formalization, a three-layer scheduling framework, route comparisons, a recipe-diagnosis algorithm, reproducible illustrative calculations, and five forward-looking judgments with falsifiers.

Keywords: unified likelihood training; data mixture; effective learning rate; MaxRL; AIR schedule; midtraining; recipe engineering

Introduction

For a decade, the training narrative for foundation models has been dominated by "stages": maximum-likelihood pretraining on web-scale text[1–3], midtraining on higher-quality or more downstream-like corpora[4,5], then SFT on human demonstrations, and finally RL alignment with preferences or verifiable rewards[6–8]. Scaling laws describe compute–data–parameter tradeoffs[9,10]; Transformers and subsequent open and closed bases standardized pretraining itself[11–15]; InstructGPT, preference modeling and proximal policy optimization wrote post-training as a separate pipeline[6,16–18]; direct preference optimization further collapsed preference learning into a classification loss[19]. Stage labels ease division of labour and audit, but they also freeze the habit that "changing stage means changing algorithm, trainer and team."

Two nearly simultaneous threads are unsettling that habit. First, Song et al. derive suitable RL objectives as maximum-likelihood fitting to model-generated targets, placing pretraining, midtraining, SFT and RL under one likelihood algorithm; the lifecycle becomes which data to train on, in what proportions and when, with staging as a special case of the mixture schedule (as reported)[20,21]. The foundation is Maximum Likelihood Reinforcement Learning (MaxRL): under terminal binary feedback, expected-reward RL is only a first-order approximation to likelihood, and MaxRL interpolates between them with sampling compute (as reported)[22]. Second, Yu et al. show that weight-normalization gains over weight decay are driven mainly by the induced ELR trajectory; matching ELR largely collapses the loss dynamics, so schedules should target ELR rather than nominal LR, motivating the AIR schedule (as reported)[23]. An independent study likewise reports "ELR collapse": when ELR is matched, loss trajectories approximately coincide across learning rates and parameter norms (as reported)[24].

Three gaps remain. First, industry still treats "switching on RL" or "doing SFT" as algorithm changes, without a shared language for static corpora and model-generated data in one loss. Second, data mixture and LR schedules are often tuned separately, even though evidence that they should be co-designed is growing[25,26]. Third, counter-evidence is under-cited: verifiable-reward RL may not elicit reasoning modes beyond the base model[27]; post-training incurs a "sharpening tax"[28]; norm control is not a free lunch[29,30]; and unification does not by itself remove safety gates or staged audit needs.

We do not claim a unified view will replace every staged process overnight; a likelier path is to pilot same-trainer mixing on verifiable domain models, then make ELR logs standard in pretraining reports, and only then rewrite organizational KPIs from stage KPIs to recipe KPIs—the horizon order also reflected in Table 5.

Thesis. Staged pipelines are being rewritten as a single likelihood objective plus data-mixture / ELR trajectories; what enterprises must manage are mixtures, ELR and reproducible recipes—not nominal stage labels. Contributions:

1. A joint formalization of unified likelihood training and ELR (Eqs. (1)–(6)) with a notation table, casting staging as a special mixture path and nominal LR as a visible

projection of ELR.
2. A three-layer scheduling framework—data sources, mixture weights, ELR—with mechanism analysis, route comparison and counter-evidence.
3. A recipe diagnosis and ranking algorithm (Algorithm 1), reproducible illustrations, and five forward-looking judgments with time scales, evidence, falsifiers and enterprise implications.

The rest of the paper is organized as follows. Section 2 formalizes unified likelihood, mixtures, ELR, AIR-like schedules, recipe state and pass@k; Section 3 compresses the design space into three layers; Section 4 analyses mechanisms and routes, including post-training objectives and recipe artifacts; Section 5 gives the diagnosis algorithm; Section 6 provides calculations; Sections 7–10 cover outlook, common misconceptions, limitations and conclusions. Time-pressed readers can start with Tables 2–3, Algorithm 1 and Table 5, then return to equations and counter-evidence.

Problem Formalization and Notation

A Shared Likelihood Objective

Let the policy (autoregressive LM) be π_θ. Pretraining, midtraining and SFT already share a maximum-likelihood (cross-entropy) form; they differ mainly in data: pretraining and midtraining fit corpus distributions (empty prompts), while SFT fits instruction–demonstration pairs[6,31,32]. For binary terminal rewards, let success probability $p_\theta(x)=E_{y\sim\pi_\theta(\cdot|x)}[r(x,y)]$ and the success-conditioned distribution $\pi_\theta^+(y|x)=\pi_\theta(y|x,r=1)$. MaxRL shows that its update can be written as maximum-likelihood fitting to the current policy's success-conditioned distribution (as reported)[20,22]:

$$L(\theta;\rho\circ\pi_{\theta_t}{}^{+}) = -E_{x\sim\rho}\,E_{y\sim\pi_{\theta_t}{}^{+}(\cdot|x)}[\log \pi_\theta(y|x)] \tag{1}$$

Eq. (1) returns RL to the same likelihood-fitting form as pretraining. The generator need not equal the optimized model: for any fixed generator ν, fitting its success-conditioned distribution ν^+ still uses the same likelihood loss (as reported)[20].

Source Mixtures and Staging as a Special Case

Each source s at iteration t is described by a prompt distribution $\rho_{s,t}$ and a target $\mu_{s,t}$. With mixture weights $\lambda_{s,t}\geq 0$ summing to one, the joint training distribution is

$$q_t = \textstyle\sum_s \lambda_{s,t}\,(\rho_{s,t}\circ\mu_{s,t}),\qquad L(\theta;q_t) = -E_{(x,y)\sim q_t}[\log \pi_\theta(y|x)] \tag{2}$$

Eq. (2) is the core of unified training: the learning rule is fixed; what changes is the data distribution. Conventional staging places all weight on one source at a time; unified training allows overlapping sources and smooth weight trajectories (as reported)[20]. In practice, sources share one weighted token cross-entropy: unit weights on corpus tokens for pretraining/midtraining, prompt masks for SFT, and response weights A tied to pass rate for RL (as reported)[20,33].

Effective Learning Rate

For a parameter block of norm r and nominal learning rate η, define

$$\eta_{eff} = \eta / r \tag{3}$$

Under weight normalization r is fixed, so η_{eff} tracks η; under weight decay r evolves, so the same nominal schedule induces a different ELR path. Matching ELR largely collapses WN and WD loss dynamics (as reported)[23,24]. The object to schedule is therefore $\eta_{eff}(t)$, not the $\eta(t)$ in a config file.

AIR-like Decay and Late-Stage Dominance

Over a finite horizon K with prescribed peak and terminal LRs, a theoretically motivated $1/\sqrt{t}$ decay can be affinely mapped to the AIR (Affine Inverse-square-Root) schedule. As reported, on 128B-token MoE pretraining at matched cumulative LR, AIR lowers terminal training loss by about 0.01 versus cosine; stage-swap experiments attribute most of the gain to late ELR scheduling (as reported)[23]. Schematically,

$$\eta(t) = \eta_{term} + (\eta_{peak} - \eta_{term})\cdot\varphi(t),\quad \text{with } \varphi \text{ an affinely transformed } 1/\sqrt{t} \text{ shape} \tag{4}$$

Eq. (4) stresses that shape, peak, terminal value and cumulative-LR budget jointly determine the path; comparing "cosine versus linear" without fixing cumulative LR is not meaningful.

An Identifiable Recipe State

For enterprise recipe management, define the recipe state at time t as

$$R_t = (\{\lambda_{s,t}\}_s,\ \eta_{eff}(t),\ G_t) \tag{5}$$

where G_t is the generator configuration (on-policy or not, sample count, reward definition). Runs that share a stage name but differ in R_t are not comparable; runs that differ in stage name but share similar R_t paths should be expected to share similar loss dynamics—the operational corollary of ELR collapse and unified likelihood.

Capability Retention and Sharpening

Post-training often raises pass@1 while reducing diversity or large-k coverage. With per-task success probability p,

$$pass@k = 1-(1-p)^k \tag{6}$$

If training mainly raises p on already reachable tasks (sharpening) without expanding {p>0}, small-k gains are large and large-k gains limited—consistent with reports that RL may not expand base-model reasoning modes and with the sharpening tax (as reported)[22,27,28]. Table 1 summarizes notation.

Table 1 | Main notation

Symbol	Meaning	Where
π_θ, p_θ, $\pi_\theta{}^+$	Policy, success probability, success-conditioned distribution	Eq. (1)
$\lambda_{s,t}$, q_t	Source mixture weights, joint training distribution	Eq. (2)
η, r, η_{eff}	Nominal LR, block norm, effective LR	Eq. (3)
η_{peak}, η_{term}, φ	Peak/terminal LR, decay shape	Eq. (4)
R_t, G_t	Recipe state, generator config	Eq. (5)
p, pass@k	Single-try success, success within k tries	Eq. (6)

Analysis Framework: Three Scheduling Layers

We split design under the unified view into three layers, each acting on different variables in Eqs. (1)–(6).

The layers couple: raising RL weight without adjusting ELR can look like "unstable RL"; aligning ELR while using conflicting sources looks like forgetting. Looking back at scaling laws, they answer how parameters and tokens trade under a single pretraining likelihood[9,10,23]; this article asks which quantities become first-class when the objective remains likelihood but sources and effective step sizes change over time. The two are compatible: Chinchilla-style accounting still applies to static sources, but generated tokens and reward evaluation must enter the budget, and η_{eff} trajectories must sit beside token counts as conditioning variables. BERT and T5 already showed that pretraining objective form shapes transfer[31,32]; unified training shows post-training objectives can return to the same likelihood family. History is not a simple loop: it reseparates objective form from data schedule. Once separated, organizations can iterate data quality and optimization dynamics independently instead of packaging every change as an opaque "generation upgrade."

The next section expands each layer with mechanisms and route comparisons.

Mechanism Analysis and Route Comparison

Data-source Layer: Static Corpora, Demonstrations and Verifiable Generations

The data-source layer decides what the model is fitting. Pretraining corpora supply broad linguistic and world knowledge[1,3,14]; midtraining bridges from generic web text toward code, math or domain text, with reported gains largest for domains far from general pretraining and better forgetting mitigation than specialized continued pretraining alone[4]. Surveys systematize midtraining as connecting pretraining and post-training distributions[5]; agentic midtraining recipes try to inject actionable priors before post-training[34]. SFT supplies instruction–demonstration pairs; RL sources supply rewarded model generations, linking on verifiable tasks to pure-RL reasoning incentives such as DeepSeek-R1[8,35,36].

Unified training's key mechanism is coexistence under one weighted likelihood: generators write asynchronously into a buffer; the trainer samples static data and RL generations by target token fractions (as reported)[20,33]. MaxRL clarifies why casting RL as likelihood is not mere notation: standard RL optimizes the first-order pass@1 term, while maximum likelihood corresponds to a harmonic combination of pass@k; increasing samples improves the MaxRL objective itself, whereas in REINFORCE it mainly reduces variance (as reported)[22]. Group-relative methods such as DeepSeekMath's GRPO remain important engineering baselines, but objective choice changes the pass@1–pass@k tradeoff[22,36].

Counter-evidence and boundaries. First, not all feedback is binary and verifiable: open dialogue still relies on preference models or AI feedback[7,16], and likelihoodization needs extra construction. Second, self-generated midtraining data can help later RL, but quality is capped by the generator[37]. Third, cross-behavior policy optimization shows offline demonstrations and on/off-policy rollouts can share a regression target—unification has multiple mathematical paths, not only MaxRL[38].

Mixture-weight Layer: Proportions and Timing Replace Stage Gates

The mixture-weight layer decides how much to learn when. DoReMi estimates domain weights with a small proxy via group DRO and resamples for a larger model; as reported, on The Pile it improves average few-shot accuracy by about 6.5 points over default domain weights and reaches baseline accuracy with about 2.6× fewer steps[39]. Bivariate mixture laws, mixture surveys and loss-reweighting elevate "mixture" from heuristics to optimizable objects[26,40,41]. Midtraining work also finds strong interaction between introduction time and mixture weight: early introduction tolerates higher specialized weights; late introduction cannot be compensated by simply upweighting (as reported)[4]. Unified training mixes reasoning corpora into late pretraining and continues them during RL; as reported, it improves accuracy–coverage tradeoffs at matched compute and yields a stronger continual-learning start[20].

This explains fragility of step recipes that "SFT then RL": they are λ jumps. Continual pretraining and post-alignment forgetting warn that abrupt distribution shifts hurt retained skills[42,43]; retention work stresses on-policy data for mitigating forgetting[44]. Reports that off-policy merging can beat on-policy self-distillation for continual learning further widen the mixture design space (as reported)[45]. Open recipes such as Nemotron and AceReason still showcase SFT–RL synergy, but under the unified view they are particular λ trajectories rather than different algorithmic species[46,47].

Counter-evidence: stage gates retain non-optimization reasons—safety review, data authorization, procurement and accountability often require nameable stages. Unified training demotes stages to mixture presets; it does not cancel compliance reporting. It requires reporting R_t, not merely "we are in SFT."

Table 2 | Three layers: variables, evidence and failure modes

Layer	Main variables	Representative evidence	Typical failures
Data-source layer	ρ_s, μ_s, reward definition	Midtraining bridge; DoReMi; mixture laws	Ill-defined sources; reward hacking
Mixture-weight layer	$\lambda_{s,t}$ trajectory	Unified training; timing–weight interaction	Step changes causing forgetting; late upweighting ineffective
ELR layer	$\eta_{\text{eff}}(t)$	AIR; ELR collapse; WSD cooldown	Tuning only nominal LR; early/late mismatch

ELR Layer: Schedule ELR, Not Only Nominal LR

The ELR layer decides how hard updates push. Matching interventions show that replaying norms or rescaling LR largely collapses WN–WD loss gaps; AIR, under matched cumulative LR, beats cosine and other shapes, with late scheduling explaining most of the gap to a comparator (as reported)[23]. Liu et al.'s ELR collapse reproduces "ratio-dominated loss dynamics" across optimizers, architectures and scales (as reported)[24].

WSD (Warmup-Stable-Decay) shows another staged function of schedules: a high LR advances along a "river valley," while decay reduces oscillation and reveals progress (as reported)[48,49]. Watching only nominal LR misjudges normalization when switching WN/WD—the opening paradox in the AIR paper is that under the same nominal schedule, WN eventually leads on cosine but trails throughout on WSD (as reported)[23]. SOLAR, decay-free checkpoint merging and schedule-free methods expand the design space[50–52]; terminal-shrinkage averaging and warmup-duration scaling remind us of schedule–estimator and early-sacrifice vs long-term interactions[53,54]. Conclusions that data schedules and LR schedules should be co-designed explicitly bind the mixture and ELR layers[25].

Counter-evidence and boundaries. "Hyperball is not a free lunch" and MuonGS's relaxed norm control show that norm-control gains depend on implementation details[29,30]. AIR's alignment theory rests on idealized assumptions; quantitative links to LM loss remain open (as reported)[23]. Applying the same ELR trajectory to continual training, SFT and RL still needs systematic tests.

Post-training Objectives: Likelihood-ization, Preferences and Sharpening Tax

Inside post-training, objective choice should enter the same language. DPO collapses RLHF into a classification loss[19]; PPO-style RLHF remains an important production baseline[6,17]. Mechanistic work asks what RL changes in representations[55]; Yue et al. find that verifiable-reward RL advantages can reverse at large k in favor of the base model (as reported)[27]. Sharpening Tax names the diversity cost of post-training[28]. MaxRL offers a likelihood path that can mitigate diversity collapse via a pass@k view (as reported)[22]. For unverifiable tasks, native-reasoning routes still explore training without verifiers[56].

LoRA and QLoRA change the trainable subspace, not the requirement to record recipe state[57,58]. Optimizers from Adam/AdamW to later preconditioners change update geometry and should still map to reportable ELR or equivalent coordinates[59–63].

Engineering Meaning: From Stage Narratives to Recipe Artifacts

Eqs. (2) and (5) rewrite three inventories. First, data inventories with authorization flags. Second, trainer inventories: async generation, shared buffers and weighted losses are the minimal unified-training components[33]; audit may still render a "stage view," but that view should be projected from λ trajectories. Third, optimizer inventories: AdamW, norm control and Muon-like methods are allowed, but reports need ELR or norm trajectories[23,30,60].

For domain LLMs, midtraining is often the highest-leverage yet least versioned segment. As reported, midtraining gains grow with distributional distance from general pretraining, and late introduction cannot be fixed by upweighting alone[4]. Agentic midtraining that injects tool-use and multi-step priors before post-training[34] is the same class of problem as online sample return and offline retraining: when a sample enters q_t matters more than which stage label it wears.

Compute accounting changes too. Chinchilla-style compute-optimal discussion assumes a single pretraining objective[9,10]; once RL generations consume training-token budgets, effective compute should count static tokens + generated tokens + reward evaluation. Unified training's controlled experiments compare accuracy and coverage under fixed total budgets (as reported)[20].

Open-weight reports (DeepSeek-V3, Qwen3, Nemotron 3) describe multi-stage or cascaded post-training but rarely publish reproducible λ series and ELR trajectories[14,15,46]. AceReason-style SFT–RL synergy[47] should be read as a specific mixture schedule. Surveys of replication after R1 show that reasoning-model reproduction hinges on data and reward detail[35].

Infrastructure requirements concentrate on buffer, weighting and asynchrony rather than abandoning stage scripts. Teams may keep stage names as milestones if milestones bind to R_t snapshots. Replication experience for DeepSeek-R1-like systems already points to data and reward detail rather than to a mysterious algorithm name[8,35].

When Not to Force Unification

Forcing unification fails in three modes. First, unverifiable rewards with high label noise: DPO/RLHF pipelines may remain preferable even if rewritable as some likelihood or classification objective[7,19]. Second, authorization cut by stage. Third, insufficient organizational capability: without norm logs, reward versions and retention sets, unification only builds a larger black box.

Thus we argue not for abolishing stages, but for demoting them to "mixture presets + gate labels." An analogy to agent execution layers helps: unified likelihood sets how training rules and data schedules are expressed without automatically raising data-quality caps. DoReMi-style domain reweighting assumes in-domain data are trainable[39]; midtraining bridges assume measurable relative positions of specialized and general corpora[4]. Recipe engineering does not replace data engineering.

Method: Recipe Diagnosis and Ranking

We compress the analysis into an executable procedure. Algorithm 1 aims, on a fixed base model and evaluation suite, to classify whether an intervention mainly changed mixture weights, ELR or generator configuration; to estimate effects on loss, pass@1, pass@k and forgetting; and to emit a reproducible recipe version.

Three design intents. First, step 1 forces R_t logging, matching Eq. (5). Second, step 3 surfaces confounding: many industrial "RL framework swaps" also change data, LR and sampling. Third, step 6 treats recipes as artifacts alongside weights and prompts. Experimental norms since OpenAI Gym emphasize reproducible environments and rewards[64]; foundation-model training needs the same for sources and ELR.

Illustrative Calculations

Unless noted, numbers below are illustrative to show how Eqs. (1)–(6) interact; scripts and seeds are in src/figs.py.

Mixture weights. Three sources: pretraining P, midtraining M, RL generations R. A staged recipe uses only P on 0–0.7 of budget, only M on 0.7–0.85, only R on 0.85–1.0. A unified recipe uses λ=(0.55,0.25,0.20) on 0.5–1.0 while matching staged "effective token exposure." Figure 1 shows step versus smooth λ paths.

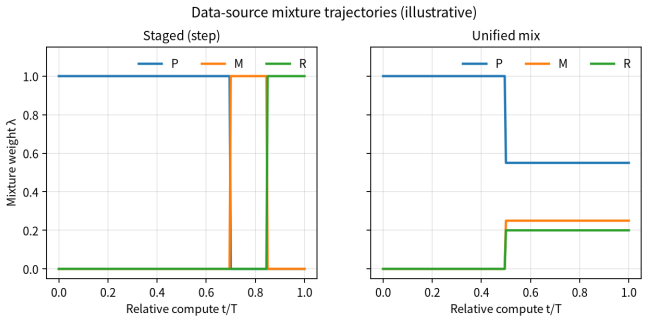


Fig. 1 | Illustrative data-source mixture-weight trajectories. Staging is a step function; unified training allows overlapping sources. Illustrative data, HX Research Team.

Effective learning rate. Suppose under weight decay a block norm grows linearly from 1.0 to 1.8 while nominal LR follows a cosine. Then $\eta_{eff}(t)=\eta(t)/r(t)$ decays faster than the nominal cosine. With weight normalization at r=1.0, $\eta_{eff}=\eta$. Figure 2 plots nominal cosine, ELR under WD, and ELR under WN.

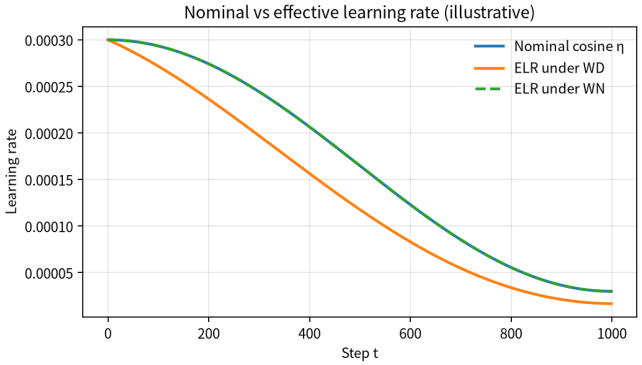


Fig. 2 | Nominal versus effective learning rate. Rising norms under weight decay pull ELR below the nominal cosine; under weight normalization they coincide. Illustrative data, HX Research Team.

Sharpening versus coverage. Simulate 1,000 tasks: 70% reachable with p∼Beta(2,3); 30% unreachable. "Sharpening post-training" raises reachable p by 50% (capped at 1) without changing coverage; "expansion" raises reachable p by 10% and moves 12% of unreachable tasks to p=0.2. Figure 3: sharpening pass@1 is about 0.421 versus expansion's 0.342; at k=32 expansion reaches about 0.821 versus sharpening's 0.703 (illustrative). This matches the qualitative MaxRL / RLVR-boundary picture[22,27].

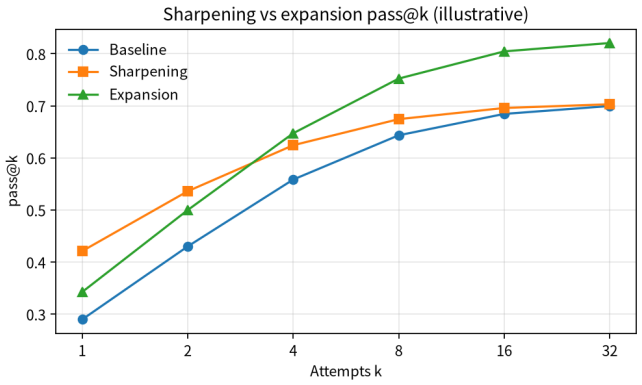


Fig. 3 | Illustrative sharpening vs expansion pass@k. Sharpening leads at small k; expansion overtakes at large k. Illustrative data, fixed seed, HX Research Team.

Table 4 is an illustrative ranking, not a claim about any real system. It conveys that mixture- and ELR-layer gains can stack, and that reporting only pass@1 overrates sharpening and underrates expansion.

Illustration: ELR Gaps under Matched Cumulative LR

To show the danger of comparing only nominal curves, take peak 3×10^{-4}, terminal 3×10^{-5}, T=10^4 steps, cosine

Table 3 | Training-route comparison (figures as reported by papers/project pages; not independently reproduced)

Route	Mechanism	Main gains	Main costs/risks	When suitable	When not
Strict staging	λ steps; algorithms may change	Clear audit; easy teaming	Switch forgetting; duplicated infra	Compliance-mandated gates	Smooth capability transfer needed
Unified likelihood mix	Multi-source under Eq. (2)	Accuracy/coverage at matched compute; continual start	Large eng change; hard rewards	Verifiable rewards + mixable static data	Sparse unverifiable rewards
DoReMi-like domain reweight	Proxy estimates λ	Few-shot and step efficiency	Proxy–large model transfer error	Multi-domain pretraining	Unstable domain definitions
Nominal LR cosine/WSD	Tune $\eta(t)$	Easy to implement	ELR incomparable across WN/WD	Known stable norm dynamics	Cross norm-control comparisons
ELR/AIR-oriented schedules	Tune $\eta_{\text{eff}}(t)$	Loss and robustness; late-stage dominance	Need norm monitoring; idealized theory	Large-scale pretraining	Norm statistics unavailable
MaxRL objective	Approximate likelihood, not only pass@1	pass@k / test-time scaling	More sampling compute	Binary verifiable tasks	Open-ended preference tasks

Algorithm 1 | Unified Recipe Diagnosis and Ranking (URDR)

Input: Base model M_0; source set S (static and verifiable generations); candidate recipes $\{R^{(0)},\ldots,R^{(m)}\}$ with $R^{(0)}$ current; compute budget B; eval set T (retention + target); tries K

Output: Per-recipe loss/downstream metrics, Δpass@1, Δpass@K, forgetting score, ELR-trajectory summary, and multiplicity-corrected ranking

1: For each recipe, log full R_t: $\lambda_{s,t}$, per-block norms and $\eta_{\text{eff}}(t)$, generator G_t; forbid stage-name-only logs
2: Train within budget B; record EMA train loss and eval loss with fixed seeds; estimate pass@1 and pass@K on T (Eq. (6))
3: Label interventions: λ-path changes → mixture layer; fixed λ with η_{eff} changes → ELR layer; only G changes → generator layer; simultaneous changes → confounded (require single-factor follow-ups)
4: Forgetting score = retention-set drop vs M_0; coverage proxy = fraction with estimated pass > 0
5: Pairwise task-level tests across recipes with multiplicity correction; keep only significant improvements
6: Export reproducible artifacts: data-manifest hashes, λ series, ELR series, reward definitions and code versions; assign a semantic recipe version

Complexity and applicability: O(m) trainings or equivalent slices; K in 8–32 to separate sharpening from expansion; requires reproducible rewards on verifiable tasks; not for open generation without objective rewards unless a preference protocol is added

Table 4 | Illustrative recipe snapshots (same compute-budget endpoint)

Recipe	Terminal λ(P,M,R)	ELR policy	pass@1	pass@32	Retention relative drop
Staged steps	(0,0,1)	Nominal cosine	0.40	0.69	0.06
Unified mix	(0.55,0.25,0.20)	Nominal cosine	0.43	0.73	0.03
Unified mix + ELR align	(0.55,0.25,0.20)	Matched AIR-shaped ELR	0.45	0.75	0.03

nominal schedule, and $r(t)=1+0.8t/T$ under weight decay. Summing nominal LR yields cumulative LR $\Sigma\eta$; summing ELR yields $\Sigma\eta_{\text{eff}}$. Under these illustrative parameters, $\Sigma\eta_{\text{eff}}/\Sigma\eta\approx0.72$—about 28% less effective update mass than the nominal ledger. Another method using weight normalization and claiming the "same cosine" has $\Sigma\eta_{\text{eff}}=\Sigma\eta$, so the comparison was never equivalent. The AIR paper's insistence on matching cumulative LR before comparing shapes closes this loophole (as reported)[23].

Illustration: Mixture-layer Buffering of Forgetting

Suppose retention-set loss rise scales with the size of a target-domain KL jump. A staged recipe jumps λ_R from 0 to 1 entering the RL segment; a unified recipe keeps $\lambda_R=0.2$ with $\lambda_P=0.55$, a smaller jump. The illustrative retention drops 0.06 vs 0.03 in Table 4 match that buffering intuition and do not replace measurement. Continual-pretraining and post-alignment forgetting studies independently motivate smooth mixtures[42,43].

Table 5 | Five forward-looking judgments (views)

#	Judgment	Horizon	Main evidence	Falsifier	Enterprise implication
1	Mainstream trainers will default to mixing static data and RL generations in one trainer; stage APIs become mixture presets	12–24 months	Unified training system/code[20,33]; MaxRL[22]	By end-2028 major open stacks still force separate SFT/RL processes with no shared loss	Require recipe-level mix APIs in build-or-buy, not only stage scripts
2	Pretraining reports will routinely publish ELR or norm trajectories; nominal LR alone will be treated as incomparable	12–18 months	AIR; ELR collapse[23,24]	Top-venue/industry reports still show only nominal LR and compare loss across WN/WD directly	Force per-block norm and η_{eff} logging on internal platforms
3	Midtraining becomes a default distributional bridge; timing and weights enter standard recipe templates	6–18 months	Midtraining bridge and survey[4,5]; agentic midtraining[34]	Controlled ablations show skipping midtraining has no material code/math loss	Line-item midtraining data budgets and acceptance tests
4	Post-training acceptance reports pass@1 and large-k pass@k/diversity; sharpening tax enters standard regressions	6–12 months	RLVR bound; sharpening tax; MaxRL[22,27,28]	Major benchmarks and customer acceptance still watch only pass@1	Add coverage/diversity clauses to contracts and eval templates
5	Enterprise train-while-serving platforms treat versioned recipes (mixture + ELR + reward) as core assets, not one-off weight files	12–24 months	Eq. (5); continual-learning start[20]; forgetting literature[42]	Production still cannot reproduce yesterday's model and no one treats recipes as artifacts	Put recipes into model registries and change management

Forward-looking Judgments and Research Agenda

The following are HX Research Team views, not facts.

Research agenda: (1) extend AIR/ELR theory into SFT/RL mix segments; (2) build estimable likelihood-proxy objectives for unverifiable tasks; (3) define cross-lab recipe exchange formats (λ series, ELR series, reward specs); (4) quantify how unified mixing affects safety gates and data-authorization audits so "algorithmic unity" does not become "governance blank."

Where Common Industry Views May Be Wrong

First, "turning on RL changes the algorithm." Under MaxRL/unified training, suitable RL updates are likelihood fitting; what changes are sources and weights, not necessarily the learning rule[20,22].

Second, "weight normalization always beats weight decay." Comparisons without matched ELR often debit schedule differences to normalization[23,24].

Third, "mixture and LR can be tuned separately then glued." Evidence for joint design is growing[25,26].

Fourth, "pass@1 up means capability expansion." It may be sharpening; check large k and retention[27,28].

Fifth, "unified training cancels stages." Unification turns stages into mixture presets; compliance and safety may still need staged gates and audit narratives.

Sixth, "LoRA/QLoRA means recipes no longer matter." Adapters shrink the update subspace; λ and ELR still decide what is learned[57,58].

In practice, a minimal five-line recipe sheet suffices: source and authorization hashes; piecewise λ(t); $\eta_{\text{eff}}(t)$ or norm-log pointer; reward and generator config; eval protocol (pass@1, pass@K, retention). Only with all five can one answer where a gain came from. For multi-tenant or multi-industry models, the sheet should also mark non-mixable sources. Unified likelihood does not mean any data may enter one loss; it means that once admitted, data are treated with one learning rule and explicit weights. Governance still decides admission—the same control idea as platforms where "the model proposes, the system decides"[65].

For evaluation governance, fix at least four curves: train loss, retention, target pass@1, target pass@K. Only when all four move may one claim "capability expansion"; if only pass@1 rises while pass@K and retention stagnate or fall, default to sharpening. The rule does not require a unified trainer, yet fits the unified view[22,27,28].

For vendor management, add three RFP items: same-loss mixing of static and generated data; exportable norm/ELR logs; recipe versions and data hashes. Missing items need not block purchase but should raise risk ratings and reproduction cost. If Deep Hope S1's train-while-serving claim is to be checkable, these are the interfaces—not marketing sentences[66].

If trainees can only repeat "we should do unified training" but cannot state this week's λ and η_{eff} changes, knowledge stopped at slogans. If they can say which gains came from the mixture layer, which from ELR, and which were only sharpening, recipe language has entered daily decisions.

Limitations

First, key evidence for unified training and AIR is recent preprint or early publication work; figures are as reported by authors and not independently reproduced here. Second, Eq. (1) emphasizes binary verifiable rewards; continuous rewards and preferences need extra constructions (e.g., TailRL-style extensions discussed in the unified-training paper)[20]. Third, mixture weights, norm paths and pass distributions in the illustrations are assumptions for mechanism, not measurements. Fourth, open and industrial recipes (DeepSeek, Qwen, Nemotron, etc.) vary in disclosure; we cite only what technical reports state and do not infer unpublished mixture numbers[14,15,46]. Fifth, product mentions are based solely on Great Unicorn Intelligence's public site descriptions of Deep Hope S1 and Harness; they are not performance promises and not customer measurements.

Conclusion

Return to the thesis. When pretraining, midtraining, SFT and suitable RL can share one likelihood objective, recipe engineering's center of gravity shifts from nominal stage labels to data mixtures and ELR trajectories. The source layer sets what is fitted; the mixture layer sets proportions and timing; the ELR layer sets update strength; together they form a versionable recipe state R_t. MaxRL and unified training unify algorithm and system; AIR and ELR collapse unify optimization dynamics; counter-evidence reminds us of sharpening taxes, forgetting, norm-control boundaries, and governance needs for stage gates.

In Great Unicorn Intelligence's product context, this judgment aligns with Deep Hope S1's train-while-serving direction: as described on the site, the platform places statistical/ML/DL and LLM deploy-and-train on one system and emphasizes an inference–training loop; the Enterprise Agent Platform · Harness places the model inside twelve components including a governance layer, stressing auditability and controlled updates[65,66]. For enterprises, transferable assets are likelier to be versioned mixtures, ELR trajectories and reward specs than a one-off weight snapshot.

Takeaways:

1. Read training via Eq. (2): the learning rule can stay fixed; what changes is q_t.
2. Read optimization via Eqs. (3)–(4): align ELR for comparison and transfer, do not paste only nominal LR curves.
3. Manage via Eq. (5): write recipe state R_t into the model registry.
4. Accept via Eq. (6): report pass@1 and large k; watch for sharpening tax.
5. Staging may remain as audit presets, but should not be mistaken for the algorithmic boundary itself.

Data availability

This article produces no new experimental data. Cited figures come from the listed references, project pages and the company site; illustrative calculations are generated by scripts under the stated assumptions.

Code availability

Scripts for the illustrative calculations and Figs. 1–3 are in src/figs.py (Python, fixed random seed), provided with this package for verification.

Competing interests

The authors are employees of Great Unicorn Intelligence Technology (Shenzhen) Co., Ltd. Deep Hope S1 and the Enterprise Agent Platform · Harness mentioned here are the company's public products.

References

1. Brown, T. et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems* **33** (2020); https://arxiv.org/abs/2005.14165.
2. Touvron, H. et al. LLaMA: Open and Efficient Foundation Language Models. Preprint at https://arxiv.org/abs/2302.13971 (2023).
3. Chowdhery, A. et al. PaLM: Scaling Language Modeling with Pathways. Preprint at https://arxiv.org/abs/2204.02311 (2022).
4. Liu, E., Neubig, G. & Xiong, C. Midtraining Bridges Pretraining and Posttraining Distributions. Preprint at https://arxiv.org/abs/2510.14865 (2025).
5. Mo, K. et al. Mid-Training of Large Language Models: A Survey. Preprint at https://arxiv.org/abs/2510.06826 (2025).
6. Ouyang, L. et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems* **35** (2022); https://arxiv.org/abs/2203.02155.
7. Bai, Y. et al. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. Preprint at https://arxiv.org/abs/2204.05862 (2022).
8. DeepSeek-AI et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *Preprint / journal ref:* Nature volume 645, pages 633-638 (2025); Preprint at https://arxiv.org/abs/2501.12948 (2025).
9. Kaplan, J. et al. Scaling Laws for Neural Language Models. Preprint at https://arxiv.org/abs/2001.08361 (2020).
10. Hoffmann, J. et al. Training Compute-Optimal Large Language Models. Preprint at https://arxiv.org/abs/2203.15556 (2022).
11. Vaswani, A. et al. Attention Is All You Need. In *Advances in Neural Information Processing Systems* **30** (2017); https://arxiv.org/abs/1706.03762.

12. Touvron, H. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. Preprint at https://arxiv.org/abs/2307.09288 (2023).
13. OpenAI. GPT-4 Technical Report. Preprint at https://arxiv.org/abs/2303.08774 (2023).
14. DeepSeek-AI et al. DeepSeek-V3 Technical Report. Preprint at https://arxiv.org/abs/2412.19437 (2024).
15. Yang, A. et al. Qwen3 Technical Report. Preprint at https://arxiv.org/abs/2505.09388 (2025).
16. Stiennon, N. et al. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems* **33** (2020); https://arxiv.org/abs/2009.01325.
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal Policy Optimization Algorithms. Preprint at https://arxiv.org/abs/1707.06347 (2017).
18. Ziegler, D. M. et al. Fine-Tuning Language Models from Human Preferences. Preprint at https://arxiv.org/abs/1909.08593 (2019).
19. Rafailov, R. et al. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems* **36** (2023); https://arxiv.org/abs/2305.18290.
20. Song, Y. et al. Training Foundation Models with a Single Algorithm. Preprint at https://arxiv.org/abs/2610.40564 (2026).
21. Song, Y. et al. Unified Training — Training Foundation Models with a Single Algorithm (project page). https://yudasong.github.io/unified-training/ (2026).
22. Tajwar, F. et al. Maximum Likelihood Reinforcement Learning. Preprint at https://arxiv.org/abs/2602.02710 (2026).
23. Yu, Q., Ma, X., Bin, X. & He, D. Your LR Schedule May Not Be Principled: Effective Learning Rate Scheduling in Weight-Normalized Training. Preprint at https://arxiv.org/abs/2610.40362 (2026).
24. Liu, Z. et al. Effective Learning Rate Governs Loss Dynamics in Language Model Pretraining. Preprint at https://arxiv.org/abs/2608.24814 (2026).
25. Wen, H., Luo, K., Yan, T., Chen, W. & Lyu, K. Data Schedules and Learning Rate Schedules Should Be Designed Together. Preprint at https://arxiv.org/abs/2610.29348 (2026).
26. Li, J., Zhang, H., Xu, J., Cai, T. & Zhang, J. Beyond Data Schedule: Fast LLM Pretraining on Data Mixtures by Loss-Reweighting. Preprint at https://arxiv.org/abs/2610.32141 (2026).
27. Yue, Y. et al. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?. *Preprint / journal ref:* NeurIPS 2025 Oral; ICML 2025 AI4MATH workshop best paper; Preprint at https://arxiv.org/abs/2504.13837 (2025).
28. Oh, C. et al. Sharpening Tax in Post-Training. Preprint at https://arxiv.org/abs/2610.01509 (2026).
29. Xiao, Y. et al. Hyperball May Not Be a Free Lunch. Preprint at https://arxiv.org/abs/2607.22444 (2026).
30. Glentis, A., Li, D. & Hong, M. MuonGS: Pretraining LLMs with Relaxed Weight-Norm Control. Preprint at https://arxiv.org/abs/2610.37647 (2026).
31. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT* 4171–4186 (2019); https://arxiv.org/abs/1810.04805.
32. Raffel, C. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* **21**, 1–67 (2020); https://arxiv.org/abs/1910.10683.
33. Song, Y. et al. unified-training (code repository). GitHub, https://github.com/yudasong/unified-training (2026).
34. Singhal, S. et al. Priming Base Models for Agentic Post-Training: An Open Midtraining Recipe. Preprint at https://arxiv.org/abs/2610.22019 (2026).
35. Liu, Y. et al. 100 Days After DeepSeek-R1: A Survey on Replication Studies and More Directions for Reasoning Language Models. Preprint at https://arxiv.org/abs/2505.00551 (2025).
36. Shao, Z. et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. Preprint at https://arxiv.org/abs/2402.03300 (2024).
37. RRV, A. et al. Mid-Training with Self-Generated Data Improves Reinforcement Learning in Language Models. Preprint at https://arxiv.org/abs/2605.08472 (2026).
38. Cheng, D. et al. Cross-Behavior Policy Optimization: Unified Learning from Offline Demonstrations and On- and Off-Policy Rollouts via a Shared Regression Target. Preprint at https://arxiv.org/abs/2610.12830 (2026).
39. Xie, S. M. et al. DoReMi: Optimizing Data Mixtures Speeds Up Language Model Pretraining. Preprint at https://arxiv.org/abs/2305.10429 (2023).
40. Ge, C., Ma, Z., Chen, D., Li, Y. & Ding, B. BiMix: A Bivariate Data Mixing Law for Language Model Pretraining. Preprint at https://arxiv.org/abs/2405.14908 (2024).
41. Chen, Z., Miao, Y., Supryadi & Xiong, D. Data Mixing for Large Language Models Pretraining: A Survey and Outlook. *Preprint / journal ref:* Data Intelligence 8 (2026), Art. No. 2026r01; Preprint at https://arxiv.org/abs/2604.16380 (2026).
42. Li, C. & Lee, H. Examining Forgetting in Continual Pre-training of Aligned Large Language Models. Preprint at https://arxiv.org/abs/2401.03129 (2024).
43. Ibrahim, A. et al. Simple and Scalable Strategies to Continually Pre-train Large Language Models. Preprint at https://arxiv.org/abs/2403.08763 (2024).
44. Chen, H., Razin, N., Narasimhan, K. & Chen, D. Retaining by Doing: The Role of On-Policy Data in Mitigating Forgetting. *Preprint / journal ref:* Proceedings of the 43rd International Conference on Machine Learning (ICML), 2026; Preprint at https://arxiv.org/abs/2510.18874 (2025).
45. Wu, C. H., Zhang, T. & Raghunathan, A. Off-Policy Merging Beats On-Policy Self-Distillation for Continual Learning. Preprint at https://arxiv.org/abs/2610.05872 (2026).
46. NVIDIA et al. NVIDIA Nemotron 3: Efficient and Open Intelligence. Preprint at https://arxiv.org/abs/2512.20856 (2025).
47. Liu, Z. et al. AceReason-Nemotron 1.1: Advancing Math and Code Reasoning through SFT and RL Synergy. Preprint at https://arxiv.org/abs/2506.13284 (2025).
48. Wen, K. et al. Understanding Warmup-Stable-Decay Learning Rates: A River Valley Loss Landscape Perspective. Preprint at https://arxiv.org/abs/2410.05192 (2024).
49. Dremov, A., Hägele, A., Kosson, A. & Jaggi, M. Training Dynamics of the Cooldown Stage in Warmup-Stable-Decay Learning Rate Scheduler. *Preprint / journal ref:* Transactions on Machine Learning Research (TMLR), 2025; Preprint at https://arxiv.org/abs/2508.01483 (2025).
50. Shang, Q. et al. SOLAR: A State-Driven Online Learning Rate Scheduler for LLM Pretraining. Preprint at https://arxiv.org/abs/2609.34681 (2026).
51. Tian, C. et al. WSM: Decay-Free Learning Rate Schedule via Checkpoint Merging for LLM Pre-training. Preprint at https://arxiv.org/abs/2507.17634 (2025).
52. Defazio, A. ScheduleFree+: Scaling Learning-Rate-Free & Schedule-Free Learning to Large Language Models. Preprint at https://arxiv.org/abs/2605.19095 (2026).
53. Ousherovitch, A. & Wang, Y. Terminal Shrinkage Averaging Reveals a Schedule-Estimator Interaction in LLM Pretraining. Preprint at https://arxiv.org/abs/2609.25482 (2026).
54. Topollai, K. & Choromanska, A. Balancing Early Performance Sacrifices with Long-Term Gains: Scaling Learning-Rate Warmup Duration Across Training Horizons. Preprint at https://arxiv.org/abs/2609.33041 (2026).
55. Du, L., Tang, Y., Duan, X. & Liu, D. What Does an LLM Learn from Reinforcement Learning? A Mechanistic Interpretability Perspective with Fixed-SAE Track. Preprint at https://arxiv.org/abs/2609.15064 (2026).
56. Wang, Y., Liu, Z., Li, X., Lu, C. & Yang, C. Native Reasoning Models: Training Language Models to Reason on Unverifiable Data. Preprint at https://arxiv.org/abs/2602.11549 (2026).
57. Hu, E. J. et al. LoRA: Low-Rank Adaptation of Large Language

Models. In *Proc. International Conference on Learning Representations* (2022); https://arxiv.org/abs/2106.09685.

58. Dettmers, T., Pagnoni, A., Holtzman, A. & Zettlemoyer, L. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems* **36** (2023); https://arxiv.org/abs/2305.14314.
59. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. In *Proc. International Conference on Learning Representations* (2015); https://arxiv.org/abs/1412.6980.
60. Loshchilov, I. & Hutter, F. Decoupled Weight Decay Regularization. In *Proc. International Conference on Learning Representations* (2019); https://arxiv.org/abs/1711.05101.
61. Ioffe, S. & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. 32nd International Conference on Machine Learning* (PMLR, 2015); https://arxiv.org/abs/1502.03167.
62. Salimans, T. & Kingma, D. P. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems* **29** (2016); https://arxiv.org/abs/1602.07868.
63. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. Preprint at https://arxiv.org/abs/1512.03385 (2015).
64. Brockman, G. et al. OpenAI Gym. Preprint at https://arxiv.org/abs/1606.01540 (2016).
65. 弘犀智能（Great Unicorn Intelligence）. 企业智能体平台 · Harness（Enterprise Agent Platform · Harness）. https://www.hxallai.com/en/platform/agent/ (2026).
66. 弘犀智能（Great Unicorn Intelligence）. Deep Hope S1 · 推训一体大模型平台（Deep Hope S1 · unified inference–training LLM platform）. https://www.hxallai.com/en/platform/deep-hope/ (2026).

Copyright Notice: © 2026 Great Unicorn Intelligence Technology (Shenzhen) Co., Ltd. All rights reserved. Any reposting, quotation in commentary, or adaptation must credit the authors (Great Unicorn Intelligence Research Team) and cite the source (Great Unicorn Intelligence, https://www.hxallai.com).