xinxu

SII-GAIR

No Second Run: Does RSI Stop at Frontier Pre-training?

SII & Xinxu AI

Abstract

Pre-training is where every foundation model begins; bringing recursive self-improvement (RSI) there would let AI improve the process that creates AI, and could change how the entire industry builds foundation models. Yet public RSI work has so far targeted only post-training, which starts from an existing base model, runs trials costing GPU-hours, and returns verifiable scores. RSI for pre-training breaks all three conditions at orders-of-magnitude higher cost: a run consumes hundreds of thousands of GPU-hours over months and cannot be repeated; its objective, post-training headroom, cannot be read off any single score; and what must improve is not a model but the whole production & organization system, from data pipelines and training infrastructure to the teams that run them. To our knowledge, **we are the first in the world to apply RSI to an entire frontier pre-training run from scratch**, and to report it in detail. RSI does not stop at frontier pre-training, but it changes form: since no end-to-end loop can close over a single unrepeatable run, we decompose the *pre-training loop* into smaller loops that can, each repairing a step the full run breaks.

The result is 801-30B-A3B-Base, a 30B-A3B MoE model trained from scratch on 11.8T tokens in three months by a newly formed team. We release it as a ***base* model**: full post-training is left to a subsequent release, and the lightweight SFT reported here (6B tokens, no RL) only probes post-training headroom. As a base model, 801-30B-A3B outperforms Qwen3-30B-A3B and Nemotron-3.5-Lightning-30B-A3B, trailing only the larger Qwen3.5-35B-A3B, and keeps this position under identical SFT; with lightweight SFT alone, it achieves the highest average on four agentic benchmarks against fully post-trained models, including Qwen3.6-35B-A3B. Post-training RSI starts where this work ends: it presupposes a strong base model, exactly what pre-training must produce. Bringing RSI into pre-training thus acts on the foundation every later stage builds on. We will open-source 801-30B-A3B-Base and the lightweight SFT data, a ready-made resource for evaluating the post-training headroom of base models, at `https://github.com/GAIR-NLP/801-30B`.

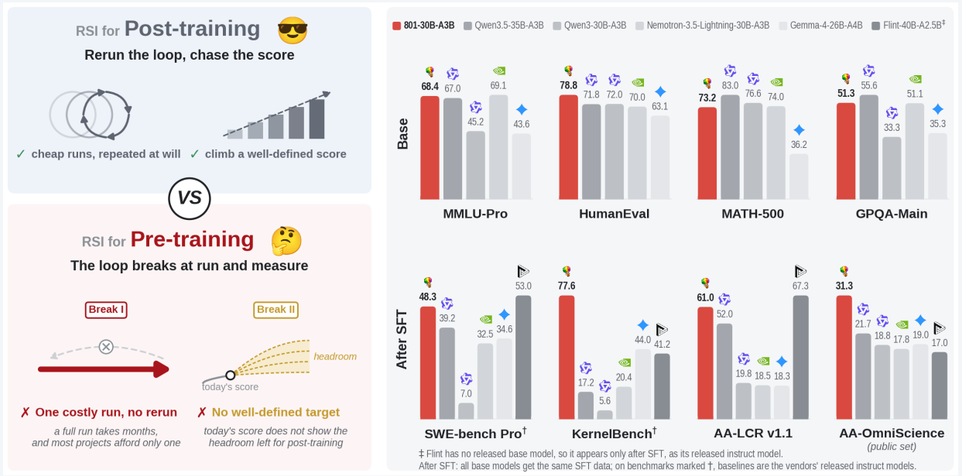


Figure 1: *Left:* recursive self-improvement (RSI) takes a different form in pre-training than in post-training. *Right:* 801-30B-A3B (red) against open baselines on base-model benchmarks (top) and after supervised fine-tuning (bottom).

We present 801 (pronounced eight-oh-one), named after the classroom where it was built.

没有第二次完整训练：递归自我改进止步于前沿预训练吗？

SII-GAIR | SII & XinxuAI

摘要

预训练是一切基础模型的起点；将递归自我改进（RSI）引入这一阶段，就能让 AI 改进创造 AI 本身的过程，并可能改变整个行业构建基础模型的方式。然而，公开的 RSI 研究迄今仅着眼于后训练：从已有基础模型出发，以 GPU 小时为单位开展试验，并得到可验证的分数。预训练中的 RSI 打破了这三个条件，成本还高出数个数量级：一次训练历时数月，消耗数十万 GPU 小时，无法重复；其目标是后训练潜力，不能从任何单一分数中读出；需要改进的也不是某个模型，而是涵盖数据流水线、训练基础设施及其运营团队的整套生产与组织系统。据我们所知，我们是全球首个将 RSI 应用于从零开始的完整前沿预训练过程、并详细报告这一实践的团队。RSI 并未止步于前沿预训练，而是改变了形式：既然一次不可重复的完整训练无法构成端到端闭环，我们就将预训练拆成能够闭合的较小循环，分别修复完整训练所破坏的循环环节。

成果是 801-30B-A3B-Base：一个新组建的团队用三个月，从零开始在 11.8T token 上训练出总参数 30B、激活参数 3B 的 MoE 模型。我们将其作为基础模型发布，完整后训练留待后续版本；本报告中的轻量 SFT（6B token，无 RL）仅用于探测后训练潜力。作为基础模型，801-30B-A3B 超过 Qwen3-30B-A3B 和 Nemotron-3.5-Lightning-30B-A3B，仅落后于更大的 Qwen3.5-35B-A3B；采用相同 SFT 后仍保持这一排名。仅经过轻量 SFT，它在四项智能体基准上的平均分就超过了所比较的完整后训练模型，包括 Qwen3.6-35B-A3B。

后训练 RSI 从本研究结束的地方起步：它预设已经存在一个强大的基础模型，而这恰恰是预训练必须产出的成果。因此，将 RSI 引入预训练，就是改进所有后续阶段赖以建立的根基。我们将开源 801-30B-A3B-Base 和轻量 SFT 数据；后者可直接用于评估基础模型的后训练潜力。地址：https://github.com/GAIR-NLP/801-30B。

图 1：左图展示 RSI 在预训练与后训练中的不同形式。右图将 801-30B-A3B（红色）与开放基线模型对比：上方为基础模型基准，下方为监督微调后的结果。

我们将模型命名为 801（读作 eight-oh-one），以纪念构建它的那间教室。

译文说明：每个对照页左侧保留原始页面，右侧提供中文翻译。图表数值、代码、模型及基准名称保留原样；参考文献保留原始书目信息。B 表示十亿，T 表示万亿；headroom 统一译为“后训练潜力”。

1 Introduction

Recursive self-improvement (RSI), in which a system turns its own experience and feedback into persistent changes that improve its capability, has moved quickly from idea to practice. Agents now run the loop *propose* → *run* → *measure* → *update* on neural architectures (Xu et al., 2026b), GPU kernels (Yu et al., 2026; Fu et al., 2026a), agent harnesses (Xia et al., 2026), and, closest to model building, on post-training itself (NeoHorse Team et al., 2026; Rank et al., 2026; Huang et al., 2026). Where these loops touch pre-training, they do so through small-scale proxies: data-curation and data-selection strategies validated on small models (Mi et al., 2026; Xu et al., 2026c; Meng et al., 2026), or single-GPU training runs measured in minutes (Karpathy, 2026; Guo et al., 2026). These settings share three conditions: they start from an existing base model or a toy-scale one, each trial costs GPU-hours or less, and each returns a verifiable score. Pre-training is where every foundation model begins, and it satisfies none of the three. Bringing RSI there would let AI improve the very process that creates AI; yet, to our knowledge, no public work has attempted it on a full frontier pre-training run.

RSI for pre-training is not a harder instance of RSI for post-training but a different regime (Table 1). **First**, it breaks *run*. A single run consumes hundreds of thousands of GPU-hours over months, and most projects can afford only one at full length. An ablation that isolates a single cause costs GPU-weeks, so most hypotheses are never tested end to end. **Second**, it breaks *measure*. The goal of pre-training is not the highest score today but headroom for post-training. A run that chases the metric can quietly borrow against post-training, for example by annealing in supervised fine-tuning (SFT) data too early, or by collapsing output diversity at the expense of the headroom that reinforcement learning (RL) later relies on (Yue et al., 2025). Such an objective cannot be read off any single score, and if even a crisply defined benchmark gets overfit by self-improvement loops (Xia et al., 2026), a target this diffuse offers far more ways to be hacked than to be met. **Third**, what must improve is not a model but a production system and the organization that runs it. Before a single token is trained, the data must be built: in our run, six pipelines with more than 30 processing stages turned raw web pages, documents, and code into 1,053 data subsets. The training stack evolved through more than 130 infrastructure code versions and 106 hyperparameter fields. Behind these artifacts stands an organization: data, infrastructure, training, and evaluation teams, each holding a different part of the evidence, with the hardest problems falling between them. A load-balance anomaly, for instance, surfaced in training metrics but traced to an infrastructure change interacting with optimizer settings (Section 3); no single team could have closed that loop alone. Improving pre-training therefore means improving how the organization divides work, shares evidence, and remembers what it has learned. Post-training RSI starts where this system ends: it presupposes a strong base model, which is exactly what the system must produce.

Small-scale proxies and scaling laws seem to offer a fast loop, but a cheap loop cannot answer every expensive question. Seemingly minor fitting choices shift the fitted exponent (Besiroglu et al., 2024; Weng, 2026), small differences at fitting scale grow large at trillion-token scale, and a ladder of proxy runs assumes that scale is the only thing changing (Weng, 2026). Once a question couples architecture, recipe, infrastructure, and multi-source data mixtures, the proxy study needed to settle it becomes as expensive as the run it was meant to de-risk. We therefore use our two pilot runs to surface and eliminate problems, not to extrapolate scores.

Table 1: RSI for Post-training vs. RSI for frontier pre-training operate in different regimes.

	RSI for post-training	**RSI for frontier pre-training (ours)**
Starting point	An existing base model	Random initialization
Cost per trial	GPU-hours (e.g., 10 H100-hours (Rank et al., 2026))	Hundreds of thousands of GPU-hours for one run
Full-length trials	Tens to hundreds	One
Signal at *measure*	Verifiable task score	Post-training headroom, not readable from any single score
Data	A fixed dataset or environment	Built from raw sources: 6 pipelines, >30 stages, 1,053 subsets, 10.5T unique tokens
Infrastructure	A training script	>130 code versions, 106 hyperparameter fields
What improves	A model or a harness	A production system and the organization that runs it
Who is in the loop	One agent or researcher	Data, infrastructure, training, and evaluation teams with their agents

1 引言

递归自我改进（RSI）指系统将自身经验和反馈转化为持久改变，从而增强自身能力。这一概念正迅速走向实践。智能体已经在神经网络架构（Xu 等，2026b）、GPU 内核（Yu 等，2026；Fu 等，2026a）、智能体运行框架（Xia 等，2026）以及最接近模型构建的后训练本身（NeoHorse Team 等，2026；Rank 等，2026；Huang 等，2026）上运行“提出→执行→测量→更新”的循环。这些循环涉及预训练时，采用的都是小规模代理：在小模型上验证数据整理和选择策略（Mi 等，2026；Xu 等，2026c；Meng 等，2026），或开展以分钟计的单 GPU 训练（Karpathy，2026；Guo 等，2026）。这些场景有三个共同条件：从已有基础模型或玩具规模模型出发，每次试验成本不超过若干 GPU 小时，并能得到可验证的分数。预训练是一切基础模型的起点，却不满足任何一个条件。将 RSI 引入其中，意味着让 AI 改进创造 AI 本身的过程；但据我们所知，尚无公开研究在完整的前沿预训练中尝试过这一点。

预训练 RSI 并非后训练 RSI 的更困难版本，而是另一种运行范式（表 1）。首先，它破坏了“执行”环节：一次训练历时数月，消耗数十万 GPU 小时，多数项目只负担得起一次完整训练。隔离单个原因的消融实验也需要数 GPU 周，因此大多数假设从未得到端到端验证。其次，它破坏了“测量”环节：预训练追求的不是今天最高的分数，而是后训练潜力。追逐指标可能悄悄透支这种潜力，例如过早用 SFT 数据退火，或牺牲 RL 所需的潜力来压缩输出多样性（Yue 等，2025）。这一目标无法由单一分数反映。即使定义明确的基准也会被自我改进循环过拟合（Xia 等，2026），如此分散的目标就更容易被钻空子，而非真正实现。

第三，需要改进的是生产系统及其运营组织，而非模型本身。训练第一个 token 之前，必须先构建数据：本项目用六条流水线、三十多个处理阶段，将网页、文档和代码转成 1,053 个数据子集。训练系统历经超过 130 个基础设施代码版本，包含 106 个超参数字段。背后还有数据、基础设施、训练和评估团队，各自掌握不同证据，最困难的问题往往落在团队交界处。例如，某个负载均衡异常表现在训练指标中，根源却是基础设施变更与优化器设置的相互作用（第 3 节）；任何单一团队都无法独自闭合这一循环。因此，改进预训练也意味着改进组织分工、证据共享和经验记忆。后训练 RSI 从这个系统结束的地方开始：它依赖一个强大的基础模型，而这正是该系统的产物。

小规模代理和缩放定律看似提供了快速循环，但廉价循环无法回答所有昂贵的问题。细小的拟合选择会改变拟合指数（Besiroglu 等，2024；Weng，2026）；拟合尺度上的微小差异会在万亿 token 规模放大；一组逐级扩大的代理训练还假设变化的只有规模（Weng，2026）。一旦问题耦合架构、配方、基础设施和多源数据混合，所需代理研究就可能与它试图降低风险的完整训练一样昂贵。因此，我们用两次试训暴露并排除问题，而不是外推分数。

表 1：后训练 RSI / 前沿预训练 RSI（本文）

起点：已有基础模型 / 随机初始化。

单次成本：GPU 小时（如 10 个 H100 小时）/ 数十万 GPU 小时。

完整试验次数：数十至数百 / 一次。

测量信号：可验证的任务分数 / 无法由单一分数读取的后训练潜力。

数据：固定数据集或环境 / 从原始来源构建，6 条流水线、超过 30 个阶段、1,053 个子集、10.5T 唯一 token。

基础设施：训练脚本 / 超过 130 个代码版本、106 个超参数字段。

改进对象：模型或运行框架 / 生产系统及其运营组织。

参与者：一名智能体或研究者 / 数据、基础设施、训练、评估团队及其智能体。

If the end-to-end loop cannot be rerun, where do agents earn their place? The run leaves behind a vast context: internal metrics, hyperparameter changes, data-mixture edits, infrastructure diffs, and dozens of evaluations on every intermediate checkpoint, far more than any team can read, let alone remember. The characteristic waste in this regime is not a bad experiment but re-investigating a question settled weeks earlier, because no one linked the symptom to its class of cause. The bottleneck thus shifts from compute to institutional memory: what was tried, what failed and why, and which signals co-occurred. Agents earn their place by recalling and attributing across evidence no human will finish reading, and, wherever a fast signal can be built, by doing the work itself: building data-pipeline components, crawling the web, and generating agentic training data at a breadth no team could manage by hand.

Our answer to the question in the title is that RSI does not stop at frontier pre-training, but it changes form. Since no end-to-end loop can close over a single unrepeatable run, we decompose pre-training into smaller loops that can close, each repairing a step the full run breaks. *Debugging loops* relocate *run* onto the infrastructure and data stacks, which can be rerun cheaply; the *headroom loop* repairs *measure* for the run as a whole; and *experience-to-metric loops* build a *measure* where none existed. A shared *context infrastructure* serves as the organization's memory: it integrates evidence across teams and preserves negative results and their reasons, which is where the *update* step persists and the recursion lives (Figure 2).

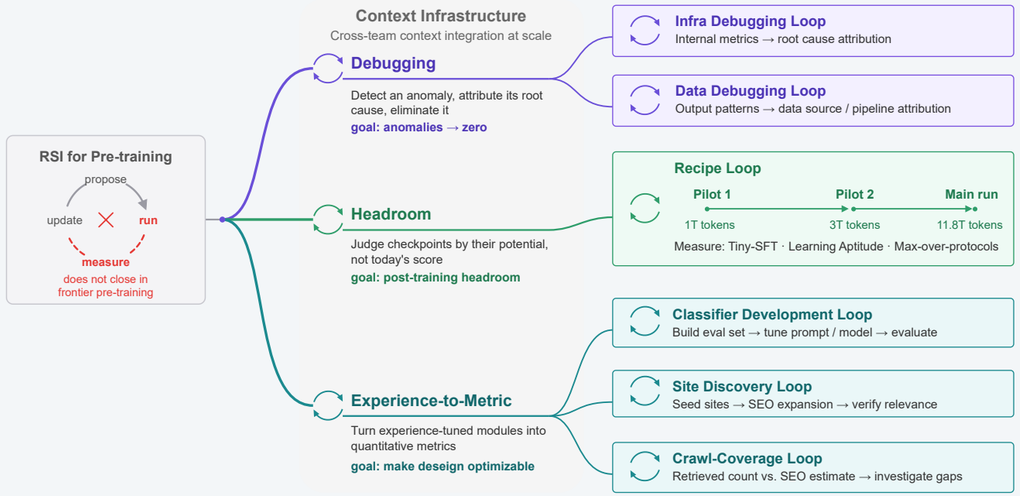


Figure 2: The standard RSI loop (*propose* → *run* → *measure* → *update*) does not close in frontier pre-training: a run takes months and cannot be repeated, and today's score does not reflect the headroom a checkpoint leaves for post-training. We propose three types of loops, instantiated as six practical loops. Debugging loops comprise the infra debugging loop and the data debugging loop, which trace internal-metric and output anomalies to their causes. The headroom loop is instantiated as the recipe loop, using headroom measurements to guide the whole pre-training process. Experience-to-metric loops comprise the classifier development loop, the site discovery loop, and the crawl-coverage loop, turning experience-based judgment into measurable feedback. All six loops share a context infrastructure that integrates evidence across teams at scale.

We built these loops while training 801-30B-A3B, a 30B-A3B Mixture-of-Experts (MoE) (Shazeer et al., 2017) model trained from scratch on 11.8T tokens in three months by a newly formed team. To our knowledge, this is the first public account of RSI applied to an entire frontier pre-training run. Our contributions are:

- *(a) Debugging loops (Section 3).* An *infra debugging loop* and a *data debugging loop*, driven by internal-metric and output anomalies, respectively. Across two pilots (stopped at 1T and 3T tokens) and the main run, agents localized causes spanning infrastructure, data, and recipe, from a load-balance penalty interacting with optimizer state to low-diversity synthetic data, and resolved five cross-stack diagnoses in about one month. Together with data and recipe revisions, these fixes raised the mean score over 22 benchmarks (Appendix B) at a matched 1T-token budget from 47.78 (initial pilot) to 51.96 (revised pilot) to 56.83 (main run).
- *(b) A headroom loop (Section 4).* Because base scores do not preserve post-training rankings, we judge checkpoints by what they are *worth* after adaptation, using tiny-SFT, a learning aptitude curve, and a max-over-protocols

如果端到端循环无法重跑，智能体的价值在哪里？训练留下海量上下文：内部指标、超参数变化、数据混合调整、基础设施差异，以及每个中间检查点的数十项评估，远超任何团队能够读完、更不用说记住的范围。这种范式下，典型浪费不是做了一个坏实验，而是因为没人将症状关联到其原因类别，又去调查数周前已经解决的问题。因此，瓶颈从算力转向组织记忆：尝试过什么、什么失败了及其原因、哪些信号同时出现。智能体的价值在于跨越人类无法读完的证据进行检索与归因；凡是能构建快速反馈信号的地方，它们还能直接开展工作，例如构建数据流水线组件、抓取网页、生成智能体训练数据，其覆盖广度远超人工团队。

对标题问题，我们的回答是：RSI 并未止步于前沿预训练，而是改变了形式。一次不可重复的训练无法形成端到端闭环，于是我们将其拆成能闭合的较小循环，分别修复被完整训练破坏的环节。调试循环将“执行”转移到可廉价重跑的基础设施和数据系统；潜力循环修复整次训练的“测量”；经验量化循环则为原本没有测量信号的地方建立测量。共享上下文基础设施充当组织记忆，整合跨团队证据，保存负面结果及其原因；“更新”由此持久化，递归性也由此产生（图 2）。

图 2：标准 RSI 循环“提出→执行→测量→更新”无法在前沿预训练中闭合：训练历时数月且不可重复，当前分数又无法反映后训练潜力。本文提出三类循环，落实为六个实际循环，均共享可大规模整合跨团队证据的上下文基础设施。

调试类：基础设施调试循环，内部指标→根因归因；数据调试循环，输出模式→数据来源或流水线归因。目标是发现、归因和消除异常，使异常趋近于零。

潜力类：训练配方循环，以 1T token 试训、3T token 试训和 11.8T token 主训练为节点，通过 Tiny-SFT、学习能力及跨评估协议取最大值来衡量检查点，追求后训练潜力。

经验量化类：分类器开发循环（建评估集→调提示词或模型→评估）；站点发现循环（种子站点→SEO 扩展→验证相关性）；抓取覆盖循环（抓取数量与 SEO 估计比较→调查缺口）。目标是将经验判断变为定量反馈，使设计可优化。

我们在训练 801-30B-A3B 的过程中建立了这些循环。这个总参数 30B、激活参数 3B 的混合专家（MoE）模型（Shazeer 等，2017）由新团队在三个月内从零开始训练，使用 11.8T token。据我们所知，这是首份将 RSI 应用于整个前沿预训练过程的公开报告。贡献如下：

(a) 调试循环（第 3 节）。分别由内部指标异常和输出异常驱动的基础设施与数据调试循环。在两次分别止于 1T、3T token 的试训及主训练中，智能体定位了跨基础设施、数据和配方的根因，包括负载均衡惩罚与优化器状态的相互作用、低多样性合成数据等，约一个月完成五项跨系统诊断。结合数据和配方修订，相同 1T token 预算下，22 项基准（附录 B）的平均分由 47.78 提升至 51.96，再提升至 56.83。

(b) 潜力循环（第 4 节）。基础模型分数无法保持后训练后的排名，因此我们按适应后的价值判断检查点，使用 Tiny-SFT、学习能力曲线和跨协议最大值来估计能力上限。这些测量能区分基础评估的表面停滞与适应后仍在持续的提升。

estimate of the capability ceiling. These measurements separate apparent stagnation in base evaluation from continued gains after adaptation.

- *(c) Experience-to-metric loops (Section 5).* Classifier development, site discovery, and crawl coverage, in which agents turn judgment once made by feel into recorded, quantitatively evaluated workflows. We built six data pipelines from scratch this way.
- *(d) Agentic data at pre-training scale (Section 6).* Agents cross from consuming training to producing it: 220K executable pull requests rolled out across nine harnesses yielded more than 100B tokens of verified agentic trajectories (134B agentic tokens in total), used in the long-context stage of pre-training.

What we release and how to compare it. We release 801-30B-A3B-Base as a *base* model. Full post-training is left to a subsequent release; the lightweight SFT reported here (6B tokens, no RL) serves only to probe post-training headroom. We therefore compare at three levels, from the fairest to the least favorable to us (Section 8). (1) *Base against base*: 801-30B-A3B outperforms Qwen3-30B-A3B (Yang et al., 2025a) and Nemotron-3.5-Lightning-30B-A3B (NVIDIA et al., 2026), trailing only the larger Qwen3.5-35B-A3B (Qwen Team, 2026a). (2) *Identical SFT*: when every base model is fine-tuned on the same SFT data, it keeps this position. (3) *Lightweight SFT against full post-training*: even so, it achieves the highest average across four agentic benchmarks among the fully post-trained models we compare, including Qwen3.6-35B-A3B (Qwen Team, 2026b). The third comparison pits a few billion SFT tokens against complete vendor post-training pipelines; we report it as a reference point for headroom, not as a like-for-like comparison. We will open-source 801-30B-A3B-Base and the lightweight SFT data, a ready-made resource for evaluating the post-training headroom of base models, at `https://github.com/GAIR-NLP/801-30B`.

2 Adapting RSI to Frontier Pre-Training

Recursive self-improvement (RSI) is the idea that a system turns its own experience and feedback into persistent changes that improve its capability and, in the strongest form, the process of improvement itself. The engine behind it is a closed loop, exemplified by AlphaEvolve (Novikov et al., 2025) and ASI-Evolve (Xu et al., 2026c). We write that loop as *propose* → *run* → *measure* → *update* and use it as the operational definition of RSI throughout this report. In this report, the improving system is the pre-training project itself, comprising its agents, researchers, and infrastructure, rather than the model being trained.

The four steps are worth defining precisely. *Propose* forms a candidate change or hypothesis, the role played by the generator in generate-critique-revise schemes (Madaan et al., 2023) and by the proposer in proposer-solver self-play (Zhao et al., 2025). *Run* realizes that candidate to produce evidence, by training, rolling out, or executing it. *Measure* scores the evidence against a signal that stands in for human judgment or the desired target. Chen et al. (2026) arrange candidate signals into a verification hierarchy, from formal verifiers at the strong end to a model's intrinsic self-assessment at the weak end. *Update* folds the outcome back into a persistent change, keeping what scored higher and discarding the rest, and in the recursive case it also revises the loop machinery that produced the change. This persistence is what makes the loop recursive rather than a one-off search.

Two preconditions decide whether this loop improves anything. The first is cheap, repeatable iteration: gains compound only when a trial is cheap enough that *run* and *update* can cycle many times. The second is a reliable signal at *measure*, aligned with the true goal. Without an external signal there is no reliable improvement ; with a weak or misaligned one, the loop degenerates through self-confirming errors, model and diversity collapse (Shumailov et al., 2024), or frame lock-in on a well-posed but wrong objective (Chen et al., 2026).

As argued in Section 1, frontier pre-training breaks both preconditions: a full-length run cannot be repeated, and its true objective, post-training headroom, is too diffuse to serve as a trustworthy *measure*. The system that must improve is, moreover, an organization rather than a model. The standard end-to-end loop therefore does not transfer to this regime. We keep the loop, but change where it closes.

Three types of loops. Rather than casting pre-training as one score-driven loop, we close three types of smaller loops around it, with six concrete instances (Figure 2). Table 2 shows how each type instantiates the four steps.

- *Debugging loops* relocate *run* onto the infrastructure and data-pipeline stacks, where a diagnosis can be rerun cheaply. They are driven by anomalies rather than by a score to maximize: *measure* asks whether the anomaly disappears, and each ruled-out cause is recorded as a negative result. The *infra debugging loop* traces unusual internal metrics to infrastructure code or training-recipe settings, and the *data debugging loop* traces unusual output patterns to a specific data source or processing stage (Section 3).
- The *headroom loop* repairs *measure* for the run as a whole. Its signal is post-training headroom, estimated through tiny-SFT, learning aptitude, and max-over-protocols evaluation rather than read off a base-model

(c) 经验量化循环（第 5 节）。在分类器开发、站点发现、抓取覆盖三个循环中，智能体将凭经验做出的判断转化为有记录、可定量评估的工作流。我们以这种方式从零构建了六条数据流水线。

(d) 预训练规模的智能体数据（第 6 节）。智能体从训练成果的使用者变成训练数据的生产者：22 万个可执行拉取请求在九种运行框架中展开，产生超过 100B token 的已验证智能体轨迹；智能体 token 总量为 134B，用于预训练的长上下文阶段。

发布内容与比较方式。我们发布基础模型 801-30B-A3B-Base。完整后训练留待后续版本；此处轻量 SFT（6B token、无 RL）仅探测后训练潜力。因此第 8 节按从最公平到最不利于我们的顺序开展三层比较：(1) 基础模型对基础模型：本模型超过 Qwen3-30B-A3B（Yang 等，2025a）和 Nemotron-3.5-Lightning-30B-A3B（NVIDIA 等，2026），仅落后于更大的 Qwen3.5-35B-A3B（Qwen Team，2026a）。(2) 相同 SFT：所有基础模型在同一 SFT 数据上微调后，排名保持不变。(3) 轻量 SFT 对完整后训练：即便如此，本模型仍在四项智能体基准上获得最高平均分，比较对象包括 Qwen3.6-35B-A3B（Qwen Team，2026b）。第三层用数十亿 SFT token 对比厂商完整后训练流程，只用于展示潜力，不是同条件比较。基础模型与轻量 SFT 数据将在 https://github.com/GAIR-NLP/801-30B 开源。

2 让 RSI 适应前沿预训练

RSI 的核心是将系统自身经验与反馈转化为提升能力的持久改变；最强形式还包括改进“改进过程”本身。其引擎是 AlphaEvolve（Novikov 等，2025）和 ASI-Evolve（Xu 等，2026c）所展示的闭环。本文以“提出→执行→测量→更新”为 RSI 的操作性定义。改进的系统是由智能体、研究者和基础设施组成的预训练项目本身，而不是正在训练的模型。

四个步骤须精确定义。“提出”生成候选改动或假设，对应生成—批评—修订方案中的生成器（Madaan 等，2023），以及提出者—求解者自博弈中的提出者（Zhao 等，2025）。“执行”通过训练、轨迹展开或代码运行实现候选项并产生证据。“测量”依据替代人类判断或期望目标的信号评分。Chen 等（2026）将信号按验证强度排序：从最强的形式验证器到最弱的模型内在自评。“更新”将结果纳入持久改变，保留高分方案、丢弃其余方案；在递归情形中，还会修改产生改动的循环机制。正是这种持久性使其成为递归循环，而非一次性搜索。

循环能否带来改进取决于两个前提。其一，迭代廉价且可重复，只有“执行”和“更新”能够多次循环，收益才会累积。其二，“测量”有与真实目标一致的可靠信号。缺少外部信号就没有可靠改进；信号薄弱或错位则会导致自我强化的错误、模型及多样性坍缩（Shumailov 等，2024），或锁定在定义清晰但方向错误的目标上（Chen 等，2026）。

前沿预训练打破这两个前提：完整训练无法重复，后训练潜力这一真实目标又过于分散，难以可靠测量。此外，须改进的是组织而非模型。标准端到端循环因而不能直接移植。我们保留循环，但改变其闭合位置。

三类循环。我们围绕预训练建立三类较小循环，共六个具体实例（图 2），而非将预训练视为一个分数驱动的大循环。表 2 展示其四步骤。

调试循环将“执行”迁移到基础设施和数据流水线，便于廉价复验诊断。它由异常驱动，而非追求分数最大化：“测量”判断异常是否消失，并将每个被排除的原因记为负面结果。基础设施调试追溯内部指标异常到代码或配方；数据调试追溯输出异常到具体来源或处理阶段（第 3 节）。

潜力循环修复整次训练的“测量”：用 Tiny-SFT、学习能力与跨协议最大值估计后训练潜力，而非直接读取基础模型分数。

Table 2: How each loop type instantiates the four RSI steps. Debugging and experience-to-metric loops close quickly on cheap components; the headroom loop closes only a few times per project, because its *run* is the pre-training run itself.

Loop type	Propose	Run	Measure	Update
Debugging	A suspected cause: code version, configuration, or data subset	Targeted diagnosis on the infra or data stack; cheap and repeatable	Does the anomaly disappear?	Apply the fix; record ruled-out causes
Headroom	A recipe change, or whether to continue training	Pilots and main-run checkpoints; costly and not repeatable	Tiny-SFT, learning aptitude, max-over-protocols	Revise the recipe; judge training progress
Experience-to-metric	A new prompt, model, or crawler version	Component-level evaluation; cheap	Task-specific evaluation set; SEO page-count estimate	Deploy, or iterate again

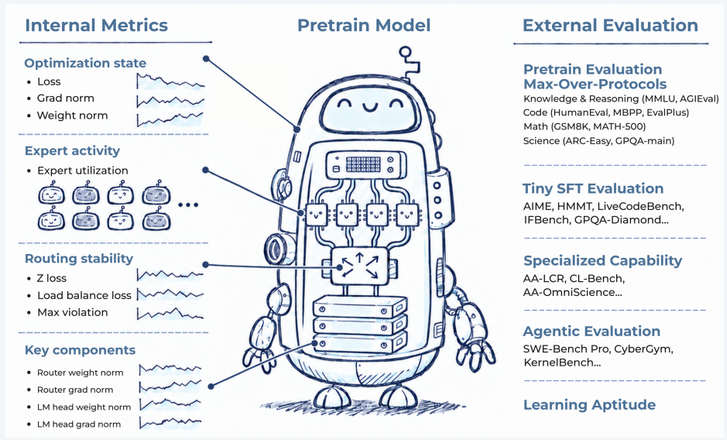


Figure 3: Overview of internal training metrics and external evaluation protocols for pre-training monitoring.

score. It is instantiated as the *recipe loop*, which uses these measurements to revise the recipe from the two pilots, stopped at 1T and 3T tokens, to main pre-training, and to monitor progress within the main run (Section 4). Because its *run* is the costly run itself, it closes only a few times per project.

- *Experience-to-metric loops* build a *measure* where none existed. Steps such as building data-pipeline components and acquiring web data were historically tuned by hand; we give each an explicit evaluation, such as a task-specific evaluation set or an SEO page-count estimate, so that revisions can be compared. This type is instantiated as the *classifier development loop*, the *site discovery loop*, and the *crawl-coverage loop* (Section 5).

Context infrastructure. These loops operate on different parts of pre-training, but all require evidence that spans teams: an anomaly may connect a model's output to a data-processing decision made weeks earlier, headroom measurements must be read alongside recipes and checkpoint histories, and data workflows must retain the evaluations behind each revision. We built a *context infrastructure* to integrate this evidence (internal metrics such as per-layer loss, per-module gradient norms, and expert load-balance statistics, together with hyperparameter changes, data-mixture edits, infrastructure diffs, and checkpoint evaluations), produced by agents and by researchers who separately own the data, infrastructure, training, and evaluation stacks, at a scale no team can read unaided (Table 1). It records not only what was tried but what failed and why, so that negative results become persistent knowledge. This is where the *update* step persists and where the recursion in our loops lives: each new proposal starts from what earlier iterations have ruled out, and agents direct their effort to the next unresolved bottleneck instead of rediscovering old ones.

表 2：各类循环如何落实 RSI 的四个步骤

调试循环：提出可疑原因（代码版本、配置或数据子集）；在基础设施或数据系统上执行廉价、可重复的定向诊断；测量异常是否消失；应用修复并记录排除的原因。

潜力循环：提出配方改动或是否继续训练的判断；执行试训及主训练检查点评估，成本高且不可重复；用 Tiny-SFT、学习能力、跨协议最大值测量；更新配方并判断训练进展。

经验量化循环：提出新提示词、模型或爬虫版本；执行廉价的组件级评估；用任务专属评估集、SEO 页面数估计等测量；部署方案或继续迭代。

调试及经验量化循环在廉价组件上快速闭合；潜力循环的“执行”是预训练本身，因此每个项目只能闭合少数几次。

图 3：用于预训练监控的内部训练指标和外部评估协议概览。内部指标包括：优化状态（损失、梯度范数、权重范数）、专家活动（专家利用率）、路由稳定性（z-loss、负载均衡损失、最大违背值）和关键组件（路由器权重/梯度范数、语言模型输出头权重/梯度范数）。外部评估包括跨协议最大值预训练评估、Tiny-SFT 评估、专项能力评估、智能体评估和学习能力。

潜力循环的具体形式是配方循环：使用这些测量，从分别止于 1T 和 3T token 的两次试训逐步修订到主预训练配方，并监测主训练内部的进展（第 4 节）。由于“执行”就是昂贵的训练本身，每个项目只能闭合数次。

经验量化循环为原本缺乏测量的环节建立“测量”。数据流水线组件构建与网页数据获取等步骤过去依靠人工调节；我们为其配置明确评估，例如任务专属评估集或 SEO 页面数估计，使每次修订能够相互比较。具体包括分类器开发、站点发现和抓取覆盖循环（第 5 节）。

上下文基础设施。这些循环作用于预训练的不同部分，但都需要跨团队证据。一个异常可能将模型输出与数周前的数据处理决策关联起来；潜力测量必须结合配方和检查点历史解读；数据工作流则须保存每次修订背后的评估。

为此，我们建立上下文基础设施，整合智能体以及分别负责数据、基础设施、训练、评估系统的研究者产生的证据，包括逐层损失、逐模块梯度范数、专家负载均衡统计等内部指标，以及超参数变化、数据混合调整、基础设施差异和检查点评估。证据规模之大，任何团队都无法独自读完（表 1）。系统不仅记录试过什么，也记录失败项及其原因，使负面结果成为持久知识。

“更新”在这里得以持久化，循环的递归性也在这里体现：每个新提案从此前已排除的可能性出发，智能体将精力投向下一个尚未解决的瓶颈，避免反复发现旧问题。



Figure 4: Two pilots and the main pre-training in a three-month period. *Top*: main configuration changes; matching colors link each iteration to its evaluation curves below. *Bottom*: initial pilot (warm brown) and revised pilot (gray-blue) stopped at 1T and 3T tokens, respectively; main pre-training (dark teal) is shown through Stage1-1 at 7T tokens. The five subset scores are means over their constituent benchmarks, and Overall is the mean over all shared benchmarks.

From pilots to main pre-training. Together, these loops supported revisions of the data and training recipe across two pilots and main pre-training (Figure 4). The initial pilot stopped at 1T tokens, the revised pilot at 3T, and the main run used approximately 11.8T tokens; the figure shows main pre-training through Stage1-1 at 7T. These revisions brought together debugging, headroom assessment, and quantitatively evaluated data workflows.

At 1T tokens the mean score over the 22 common benchmarks rises from 47.78 to 51.96 to 56.83, and at 3T tokens main pre-training reaches 59.94 against the revised pilot's 56.48. Because data and recipe changed together, these gains reflect the combined iterations supported by the three types of loops rather than any single fix. The following sections describe how each type contributed to this process.

3 Debugging Loops

This section shows the two debugging loops operating on the context infrastructure as we introduced in Section 2. (1) The *infra debugging loop* traces unusual internal metrics to infrastructure code or training-recipe settings. (2) The *data debugging loop* traces unusual output patterns to a specific data source or processing stage (Figure 5; Table 2 maps both loops to the four RSI steps). Unlike conventional RSI approaches, which improve performance by repeatedly training and comparing candidate models, these loops operate within a single pre-training run. Rather than restarting training to evaluate new candidate models, they continuously identify, attribute, and eliminate problems as training progresses, improving the ongoing run itself. We conclude with the five cross-stack diagnoses produced by these loops (Table 3).

图 4：三个月内的两次试训与主预训练。

上方列出关键配置变化，颜色与下方评估曲线对应。

迭代 1，初始试训：初始数据混合与配方；学习率 2.7e-3，负载均衡损失系数 1e-2；路由器使用 Muon。

迭代 2，修订试训：数据混合不变、配方修订；学习率 5e-4，负载均衡损失系数 1e-4；路由器改用 AdamW。

迭代 3，主预训练：更新数据来源和混合；从局部负载均衡改为全局负载均衡；归一化参数与偏置不施加权重衰减。

下方：初始试训（暖棕）和修订试训（灰蓝）分别止于 1T、3T token；主预训练（深青）显示到 Stage1-1 结束，即 7T token。五类子分数分别是常识、推理、代码、数学、科学类内部基准的平均值；Overall 是全部共享基准的平均值。横轴为训练 token 数（T），纵轴为分数（%）。

从试训到主预训练。这些循环共同支撑了两次试训和主训练中的数据与配方修订（图 4）。初始试训在 1T token 停止，修订试训在 3T 停止，主训练使用约 11.8T token；图中仅展示主训练的 Stage1-1，直到 7T。这些修订结合了调试、潜力评估和定量评估的数据工作流。

在 1T token 时，22 项共有基准的平均分依次由 47.78 升至 51.96、56.83；在 3T 时，主训练达到 59.94，而修订试训为 56.48。由于数据和配方同时变化，这些收益反映三类循环支持的综合迭代，而不能归因于某一个修复。后续各节将说明每类循环如何发挥作用。

3 调试循环

本节展示第 2 节所述的上下文基础设施之上运行的两个调试循环：(1) 基础设施调试循环，将异常内部指标追溯至基础设施代码或训练配方设置；(2) 数据调试循环，将异常输出模式追溯至具体数据来源或处理阶段（图 5；四个 RSI 步骤的映射见表 2）。

传统 RSI 方法通过反复训练、比较候选模型来提高性能，而这些循环在单次预训练内部运作。它们不通过重启训练来评估新模型，而是随着训练推进，持续识别、归因、消除问题，改进当前训练本身。最后我们总结由这些循环完成的五项跨系统诊断（表 3）。

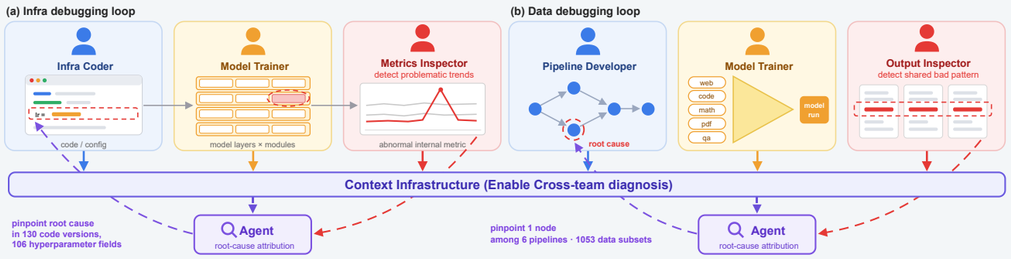


Figure 5: The debugging loops are instantiated as **(a)** the infra debugging loop and **(b)** the data debugging loop.

3.1 Infra Debugging Loop

The infra debugging loop uses internal training metrics as the starting point for identifying and resolving anomalies during pre-training. By linking these signals to infrastructure code and training-recipe configurations through the context infrastructure, agents can trace unexpected model behavior to its underlying causes, propose targeted fixes, and verify their effects within the ongoing training run. We illustrate this process through the anomalies observed in the two pilot runs, following four steps: detect, attribute, fix, and verify.

Detect. During the two pilot runs, agents scanning internal metrics through the context infrastructure identified two contrasting patterns in MoE load-balancing indicators (Figure 6). In the initial pilot, both aggregate and layer-wise MaxVio (Wang et al., 2024a) declined to low levels under a strong balancing penalty, raising concerns about potential over-balancing. The revised pilot revealed a different pattern: layers 15 and 18 developed large, persistent MaxVio violations, while the aggregate remained within a typical range. This discrepancy showed that aggregate monitoring alone could obscure severe layer-specific imbalance. Layer 48, the final layer, was also elevated, but we do not treat this alone as evidence of failure, since its function is closely tied to the output space. Together, these two patterns illustrate why monitoring MoE load balancing requires examining both aggregate trends and layer-specific deviations.

Attribute. To identify the source of these anomalies, agents used the context infrastructure to correlate metric trajectories with infrastructure code commits and optimizer configurations. By searching across more than 130 infrastructure code versions and 106 hyperparameter fields, they narrowed the cause to a change in the load-balance penalty implementation and its interaction with optimizer state updates for certain parameter groups. This process established an attribution chain from training stagnation to internal-metric anomalies, and from those anomalies to specific infrastructure and configuration changes. Rather than treating unusual metrics as isolated symptoms, agents traced them across the training stack to identify their underlying causes.

Fix. Based on this diagnosis, we first revised the pilot configuration by lowering the load-balancing (Fedus et al., 2022) coefficient from 10^{-2} to 10^{-4}, reducing the learning rate from 2.7×10^{-3} to 5×10^{-4}, and switching the optimizer for router parameters from Muon (Jordan et al., 2024; Liu et al., 2025) to AdamW (Loshchilov and Hutter, 2019). These settings were carried into main pre-training, with two further adjustments: computing the load-balance loss over the global batch rather than per micro-batch (Qiu et al., 2025), and excluding normalization and bias parameters from weight decay. Because several factors changed together, the comparison does not isolate the contribution of any single adjustment. Together, these changes constituted the revised training configuration used in main pre-training.

Verify. Subsequent training shows that the revised configuration mitigates layer-wise load imbalance and suggests that internal metrics may reveal potential training problems earlier than downstream evaluation alone. (1) First, the revised configuration substantially mitigated the layer-specific imbalance observed in the pilot. At a matched training budget of 3T tokens, layer-15 MaxVio had already exceeded 14 in the revised pilot, whereas it was around 5 in main pre-training, with pronounced escalation occurring only at a later stage (Figure 7). This indicates that the revised configuration substantially alleviated the earlier anomaly, although it did not eliminate it entirely. (2) Second, the internal metrics provided a more granular view of training dynamics than downstream evaluation alone. During main pre-training, layer-15 MaxVio rose sharply over approximately 2.2–6T tokens (the shaded interval in Figure 7). Over the same interval, downstream evaluation gains gradually diminished. After approximately 6T tokens, layer-15 MaxVio remained at a high plateau, while downstream evaluation gains within Stage1-1 had largely flattened. This temporal correspondence suggests a possible association between layer-wise load imbalance and training saturation. Layer-wise MaxVio rises well before downstream performance plateaus, indicating that internal metrics can flag potential training problems while evaluation scores are still

图 5：调试循环的两个实例。(a) 基础设施调试：基础设施编码者、模型训练者和指标检查者协同，发现问题趋势，通过智能体在超过 130 个代码版本、106 个超参数字段中定位根因。(b) 数据调试：流水线开发者、模型训练者和输出检查者协同，发现共同坏模式，通过智能体在 6 条流水线、1,053 个数据子集中定位节点。共享上下文基础设施支持跨团队诊断。

3.1 基础设施调试循环

以训练内部指标为起点，识别并解决预训练中的异常。上下文基础设施将这些信号与代码、配方配置连接，智能体据此追溯异常行为根因、提出定向修复，并在持续训练中验证效果。下面以两次试训的异常说明“发现、归因、修复、验证”四步。

发现。智能体扫描内部指标时，发现两种相反的 MoE 负载均衡模式（图 6）。初始试训中，在强均衡惩罚下，整体和逐层 MaxVio（Wang 等，2024a）均降到较低水平，提示可能均衡过度。修订试训则出现另一种模式：第 15、18 层持续存在较大的 MaxVio 违背，而整体仍在正常范围。单看整体指标会掩盖严重的局部层失衡。最后一层（第 48 层）也偏高，但其功能与输出空间密切相关，不能仅凭这一点认定故障。两种模式说明：监控 MoE 负载均衡须同时关注整体趋势与逐层偏差。

归因。智能体借助上下文基础设施，将指标轨迹与代码提交、优化器配置关联，在超过 130 个代码版本和 106 个超参数字段中搜索，最终将根因缩小到负载均衡惩罚实现的某次变化，以及它与特定参数组优化器状态更新的相互作用。归因链由训练停滞连接到内部指标异常，再连接到具体基础设施和配置变化。智能体没有把指标异常当作孤立症状，而是跨训练系统追踪其根因。

修复。根据诊断，我们先将负载均衡（Fedus 等，2022）系数从 10^-2 降至 10^-4，将学习率从 2.7×10^-3 降至 5×10^-4，并将路由器参数优化器从 Muon（Jordan 等，2024；Liu 等，2025）改为 AdamW（Loshchilov 和 Hutter，2019）。主训练沿用这些设置，另加两项改动：按全局批次而非每个微批次计算负载均衡损失（Qiu 等，2025）；归一化与偏置参数不做权重衰减。由于多个因素共同变化，比较无法隔离任何单项改动的贡献。这些变化一起构成主预训练采用的修订配置。

验证。后续训练显示，修订配置缓解了逐层失衡，也提示内部指标可能比下游评估更早揭示潜在问题。(1) 在相同 3T token 预算下，修订试训第 15 层 MaxVio 已超过 14，而主训练约为 5，明显恶化只在后续阶段出现（图 7）。修订大幅缓解了早期异常，但并未完全消除。(2) 内部指标比单独的下游评估更细致：主训练约 2.2T—6T token 期间，第 15 层 MaxVio 急剧上升（图 7 阴影），同期下游评估收益逐渐缩小；约 6T 后，该指标保持高位，而 Stage1-1 内下游收益基本趋平。

这种时间对应提示逐层负载失衡与训练饱和可能相关。逐层 MaxVio 在下游表现进入平台期之前就已上升，因此评估分数还在提高时，内部指标就可能提示潜在问题。

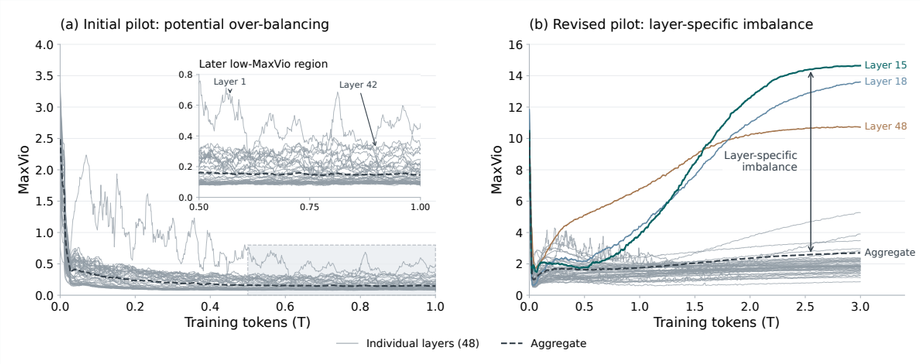
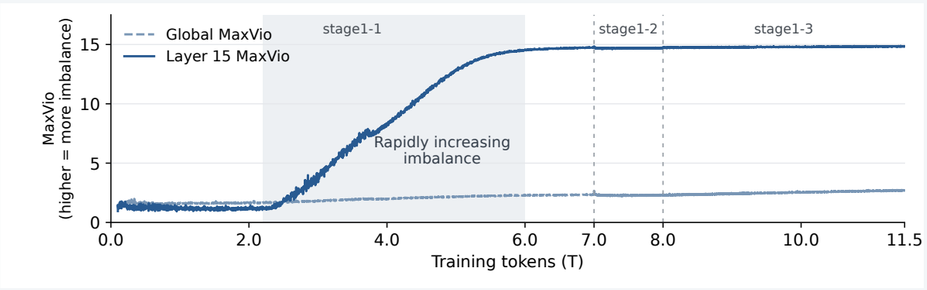
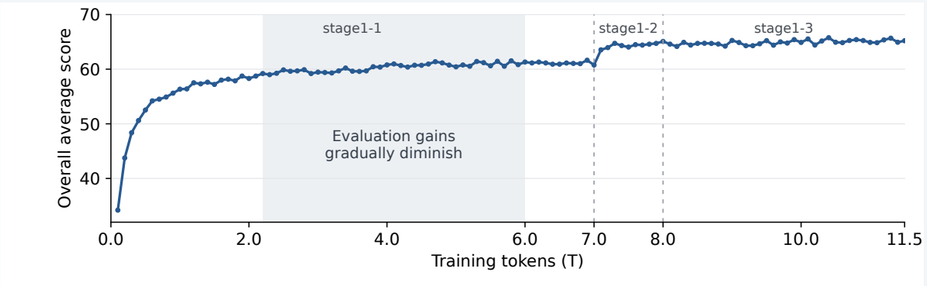


Figure 6: Two contrasting load-balancing anomalies in pilot runs. Gray curves show layer-wise MaxVio across 48 layers; the dashed curve shows aggregate MaxVio. (a) In the initial pilot, both aggregate and layer-wise MaxVio decline under strong balancing, suggesting potential over-balancing. The inset enlarges the low-MaxVio region. (b) In the revised pilot, layers 15, 18, and 48 exhibit elevated MaxVio despite a moderate aggregate, revealing layer-specific imbalance obscured by aggregate monitoring.



(a) Internal signal: layer-wise load imbalance.



(b) External signal: downstream evaluation.

Figure 7: Layer-wise load imbalance rises as downstream evaluation gains diminish during training. The gray shaded region marks the same interval (approximately 2.2–6T tokens) in both panels. (a) Layer-15 MaxVio increases rapidly during this interval, while global MaxVio remains relatively stable. (b) Over the same interval, downstream evaluation gains gradually diminish. Higher MaxVio indicates greater load imbalance. This contrast illustrates how layer-wise internal metrics provide a more granular signal for detecting potential training issues than downstream evaluation alone.

improving. Monitoring these metrics alongside downstream evaluations may therefore enable earlier investigation and intervention.

Finally, the context infrastructure allows these diagnostic experiences to persist across iterations. Because the

图 6：试训中两种相反的负载均衡异常。灰线为 48 层各自的 MaxVio，虚线为整体 MaxVio。(a) 初始试训在强均衡约束下，整体与逐层 MaxVio 都下降，提示可能均衡过度；插图放大后期低 MaxVio 区域。(b) 修订试训中，第 15、18、48 层 MaxVio 偏高，而整体值适中，说明整体监控掩盖了层级失衡。横轴为训练 token 数（T）。

图 7：随着训练推进，逐层负载失衡上升，下游评估收益递减。两幅图的灰色阴影对应同一时段，约 2.2T—6T token。(a) 内部信号：第 15 层 MaxVio 快速增加，而全局 MaxVio 相对稳定。(b) 外部信号：同期下游评估收益逐渐减少。MaxVio 越大表示负载越不均衡。这一对比说明，与单独使用下游评估相比，逐层内部指标能为发现潜在训练问题提供更细致的信号。图中还标出了 Stage1-1、Stage1-2 和 Stage1-3。

（接上页）在评估分数仍在提高时，将这些内部指标与下游评估一同监测，可能支持更早调查和干预。

最后，上下文基础设施使这些诊断经验跨迭代保留下来。由于此前的诊断及支持证据都已保存，相似异常再次出现时就可以预先判断，而无须重新从头发现。这样，我们便能依据早期 MaxVio 信号进一步调查，而不是等到下游评估收益趋平之后才行动。

previous diagnosis and its supporting evidence were preserved, the recurrence of similar anomalies could be anticipated rather than rediscovered from scratch. This allowed us to use the early MaxVio signal as a basis for further investigation, rather than waiting until downstream evaluation gains had flattened.

3.2 Data Debugging Loop

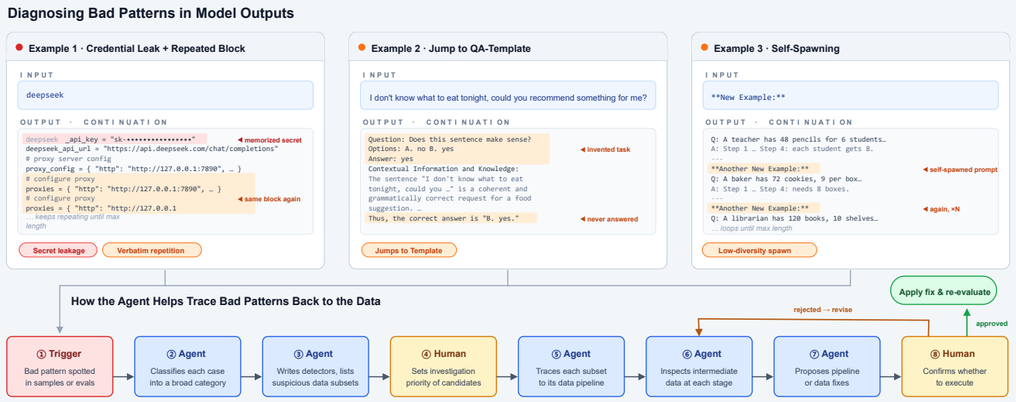


Figure 8: Diagnosing bad patterns in model outputs. *Top*: three representative output patterns, a credential leak with a verbatim-repeated config block, a jump into a fixed "Does this sentence make sense?" QA template, and a self-spawning chain of near-identical word problems. *Bottom*: the eight-step loop that traces each pattern back to a data-pipeline defect. Agents handle detection, tracing, and inspection at scale, humans set triage priority and approve the fix.

During the two pilots, we continuously monitored the output behavior of the trained model. Agents identified recurring patterns in the model's responses to greetings and common questions, which are indicative of repetitive, low-diversity, or insufficiently cleaned training data. They extracted these anomalous patterns and searched the data pipeline for matching signatures at scale.

Anomalous patterns. Figure 8 illustrates three representative patterns observed in model outputs. (1) First, *credential leakage and repetition*: given the prompt `deepseek`, the model emits a secret resembling an API key, followed by repeated copies of an identical proxy-configuration block, continuing until the generation length limit is reached. (2) Second, *jumping to a fixed QA template*: when given a natural request for a dinner recommendation, the model fails to answer the question and instead switches to a fixed "Does this sentence make sense?" multiple-choice template, proceeding to solve this invented task. (3) Third, *self-spawning repetitions*: given only the marker `**New Example:**`, the model generates an unbounded sequence of nearly identical division word problems, each following the same four-step solution scaffold.

Automated diagnosis process. The bottom of Figure 8 illustrates how agents trace these anomalous patterns back to specific data-pipeline defects. The eight-step process can be organized into three stages. (1) The first stage is *detection and pattern extraction*. Agents identify anomalies in model-generated samples or evaluation results and classify the corresponding cases. Based on the identified patterns, agents write detectors and perform large-scale matching across all 1,053 data subsets to identify those potentially containing the corresponding anomalies. (2) The second stage is *attribution and localization*. After candidate data subsets have been identified, humans determine the priority of investigation. Agents then trace these subsets to their data pipelines (six in total) and inspect the intermediate data produced at more than 30 processing stages across these pipelines, ultimately locating the specific data sources and processing stages responsible for the anomalies. (3) The final stage is *resolution and verification*. Once the source of an anomaly has been identified, agents propose fixes to the data or pipeline. Humans decide whether to approve their execution. Approved fixes are applied to the relevant processing stages, followed by model re-evaluation to verify their effectiveness; rejected proposals are returned for revision.

Throughout this process, agents handle large-scale anomaly detection, data tracing, and stage-by-stage inspection, while humans retain control over two key decision points: investigation prioritization and fix approval. This division of labor makes large-scale investigations systematic and practical, beyond what could reasonably be accomplished through manual inspection alone.

3.2 数据调试循环

图 8：诊断模型输出中的坏模式。上方三个典型案例：凭据泄露并逐字重复配置块；跳入固定的"这句话有意义吗？"问答模板；自发接续近乎相同的应用题。下方为将输出模式追溯到数据流水线缺陷的八步循环。智能体负责规模化检测、追踪和检查；人类决定调查优先级并批准修复。

例 1：输入 deepseek，输出看似 API 密钥的秘密值，随后不断重复相同代理配置，直到长度上限。例 2：输入"我不知道今晚吃什么，能推荐一下吗？"，模型却虚构"这句话有意义吗？A. 否 B. 是"的任务，分析该句并回答"B. 是"，始终没有回答原请求。例 3：仅输入 **New Example:**，模型就依次生成教师分铅笔、面包师装饼干、图书管理员分配图书等除法题，均沿用相同四步解答，反复自行生成 **Another New Example:**，直到长度上限。

两次试训期间，我们持续监测模型输出行为。智能体从问候语和常见问题的回答中发现重复模式，它们提示训练数据存在重复、多样性不足或清洗不充分。智能体提取异常模式，并在数据流水线中规模化搜索匹配特征。

异常模式。(1) 凭据泄露与重复：提示 deepseek 导致模型输出类似 API 密钥的秘密信息，并重复相同代理配置块直至长度上限。(2) 跳入固定问答模板：自然的晚餐推荐请求被替换成"这句话有意义吗？"的固定选择题，模型转而解决虚构任务。(3) 自发重复：只给出 **New Example:** 标记，模型就生成无界、近乎相同的除法应用题序列，每题均使用同一四步解题框架。

自动诊断流程。图 8 下方的八步过程可分为三阶段。

第一阶段：检测与模式提取。智能体从生成样本或评估结果中识别异常、分类案例，再编写检测器，在全部 1,053 个数据子集中进行大规模匹配，找出可能包含对应异常的子集。

第二阶段：归因与定位。人类确定候选子集的调查优先级。智能体将其追溯到六条数据流水线，并逐一检查三十多个处理阶段的中间数据，最终定位引发异常的具体来源和处理步骤。

第三阶段：解决与验证。智能体提出数据或流水线修复方案，由人类决定是否批准。批准后在相关阶段应用修复，再重新评估模型验证效果；未获批准的方案退回修订。

八步对应为：①发现样本或评估中的坏模式；②智能体分类；③编写检测器并列出可疑子集；④人类确定调查优先级；⑤追溯子集对应流水线；⑥检查各阶段中间数据；⑦提出修复；⑧人类确认是否执行。

整个过程由智能体承担大规模异常检测、数据追踪和逐阶段检查，人类保留调查排序与修复批准这两个关键决策点。这种分工使人工检查难以完成的大规模调查变得系统而可行。

Root cause attribution and fixes. Through this process, the three anomalous patterns were traced to three distinct data-pipeline defects. (1) First, the credential-leakage and repetition case was traced to insufficiently cleaned configuration files in the code corpus. These files contained sensitive information and repetitive configuration content that had entered the training data and subsequently surfaced in model generations. We addressed this issue by fixing the corresponding data-processing stage to remove such inadequately cleaned content. (2) Second, the fixed-QA-template case was traced to insufficient filtering of synthetic data in a subset of an open-source QA corpus. This subset contained a large volume of "Does this sentence make sense?" questions. We strengthened synthetic-data filtering at the corresponding stage to reduce the presence of such fixed-template examples in the training data. (3) Finally, the self-spawning case was traced to a synthetic-data pipeline whose prompt design lacked sufficient diversity, resulting in highly similar generated examples. We revised the corresponding prompt design to improve the diversity of the synthetic data. For all three cases, we modified the corresponding data-pipeline stages and re-evaluated the model after applying the fixes to verify their effectiveness.

3.3 Accumulated Diagnoses

The two debugging loops described above illustrate how agents identify anomalies in model behavior and trace them back to underlying issues in the infrastructure, training recipe, and data pipeline. However, these are only representative examples of a broader effort to diagnose and resolve problems throughout the training process.

Across the two pilots and main pre-training, we applied the same approach to systematically investigate issues across different parts of the training system. Many of these issues could not be attributed to a single component. Diagnosing them required connecting evidence from internal metrics, infrastructure code, training configurations, and data pipelines, and tracing the interactions among them. Table 3 summarizes five cross-stack issues identified and resolved through this process, covering load balancing, optimizer configurations, data quality, weight decay, and data loading. These findings demonstrate how agents, supported by the context infrastructure, can identify and resolve problems that span multiple parts of the training system.

Table 3: The five cross-stack diagnoses resolved with the context infrastructure across the two pilots and main pre-training.

Area	Finding	Fix
Load balancing	Expert load was either over-balanced across all layers or severely imbalanced in a few individual layers (Figure 6).	Adopt global-batch load balancing (Qiu et al., 2025), which relaxes the per-micro-batch constraint and allows experts to specialize.
Optimizer	Weight-norm and gradient-norm trends diverged across modules.	Reassign which modules use Muon and which stay on AdamW (Table 5).
Data quality	Pilot models generated advertisements, which were traced to advertisements and low-quality web pages remaining in the training corpus.	Filter them further with quality classifiers.
Weight decay	Router logits showed abnormal trends during training.	Exclude normalization and bias parameters from weight decay.
Data loader	When resuming a training task with the same global batch size but a different DP size, the sampler did not preserve the per-step data order of the previous task.	Restore the per-step sample order of the previous task when resuming with a different DP size.

Beyond the fixes themselves, the time required to complete these diagnoses demonstrates the value of the context infrastructure. In our experience, diagnosing issues of this kind can take months without such infrastructure. With the support of agents and the ability to connect evidence across the training stack, we completed all five diagnoses in roughly one month, substantially reducing the manual effort required to identify and trace the problems.

More importantly, the value of these diagnoses extends beyond the immediate fixes. The context infrastructure preserves the observations, attempted fixes, and negative results from each investigation, allowing subsequent iterations to build on previous findings rather than start from scratch. Each investigation thus contributes not only to improving the ongoing training run, but also to a growing body of knowledge that can be reused in future iterations. Together, the two debugging loops turn individual problem-solving efforts into a cumulative process of continuous improvement, with concrete data and training-recipe improvements carried forward into the successive iterations shown in Figure 4.

根因归因与修复。三个异常被追溯到三种不同的数据流水线缺陷。

(1) 凭据泄露与重复来自代码语料中清洗不充分的配置文件，敏感信息和重复配置进入训练数据，随后出现在模型生成内容中。我们修复相关处理步骤，移除这些未充分清洗的内容。

(2) 固定问答模板来自某开源 QA 语料子集，其合成数据过滤不足，包含大量"这句话有意义吗？"问题。我们加强该阶段的合成数据过滤，减少固定模板样本。

(3) 自发接续重复来自一条提示词缺乏多样性的合成数据流水线，生成样本高度相似。我们修改提示设计，提高合成数据多样性。三个案例均修改了对应流水线阶段，并在修复后重评模型以验证效果。

3.3 累积诊断

上述两类循环说明，智能体如何发现模型行为异常，并将其追溯至基础设施、训练配方和数据流水线问题。这些只是整个训练期间持续诊断与解决问题工作的代表案例。

在两次试训和主预训练中，我们用相同方法系统调查训练系统各部分的问题。很多问题无法归因于单一组件，必须连接内部指标、基础设施代码、训练配置和数据流水线的证据，并追踪彼此相互作用。表 3 总结了五项已识别并解决的跨系统问题，涉及负载均衡、优化器、数据质量、权重衰减和数据加载。这说明上下文基础设施支持的智能体能够解决横跨训练系统多个部分的问题。

表 3：两次试训和主训练中，通过上下文基础设施解决的五项跨系统诊断。

负载均衡：所有层过度均衡，或少数层严重失衡（图 6）。修复：采用全局批次负载均衡（Qiu 等，2025），放宽逐微批次约束，使专家能够形成专长。

优化器：不同模块的权重范数与梯度范数趋势分化。修复：重新分配使用 Muon 和保留 AdamW 的模块（表 5）。

数据质量：试训模型生成广告，追溯到语料中残留的广告和低质量网页。修复：用质量分类器进一步过滤。

权重衰减：路由器 logits 在训练中出现异常趋势。修复：归一化与偏置参数不做权重衰减。

数据加载器：在全局批次大小相同、数据并行（DP）规模不同的条件下恢复训练时，采样器未保留之前每一步的数据顺序。修复：DP 规模改变时，恢复此前训练的逐步样本顺序。

诊断耗时也展示了上下文基础设施的价值。根据我们的经验，没有此类基础设施，类似问题可能需要数月才能诊断。借助智能体及跨系统证据连接，我们约一个月就完成了全部五项诊断，大幅减少人工识别与追踪工作。

这些诊断的价值还超越眼前修复。上下文基础设施保存每次调查的观察、修复尝试和负面结果，使后续迭代能继承已有发现。每次调查既改进当前训练，也为未来积累可复用知识。两类调试循环由此将单次问题求解转化为持续累积的改进过程，并将具体的数据与配方改动带入图 4 所示的后续迭代。

4 Headroom Loop

The goal of pre-training is not to maximize today's score but to leave the greatest headroom for post-training. Measuring this headroom can supply the signal for the headroom loop. We first describe three complementary measurements (Section 4.1), then show what they revealed that base evaluation did not (Section 4.1), and finally how the recipe loop used them (Section 4.2).

4.1 Measuring Post-Training Headroom

We use three measures to assess post-training headroom: Tiny-SFT, learning aptitude, and max-over-protocols evaluation. Together, they capture three aspects of a model's potential: its performance after limited post-training, its efficiency in learning from additional data, and the capabilities it can demonstrate under evaluation protocols.

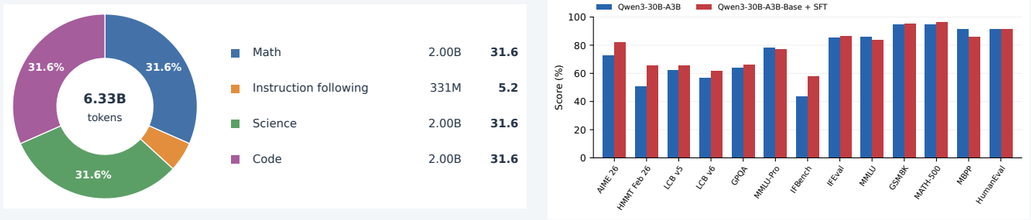


Figure 9: Tiny-SFT data composition and validation. *Left:* Token composition of the 6.33B-token mix. Math, Science (Xu et al., 2026a), and Code (Ahmad et al., 2025) each contribute approximately 2.00B tokens (31.6%), while instruction following (Pyatkin et al., 2025) contributes 0.331B tokens (5.2%). *Right:* Qwen3-30B-A3B versus its base model (Qwen3-30B-A3B-Base) after one epoch of Tiny-SFT. Scores are percentages across 13 benchmarks; the fine-tuned model scores higher on 10 of them. This validation supports the effectiveness of Tiny-SFT as a proxy for monitoring model performance after limited post-training.

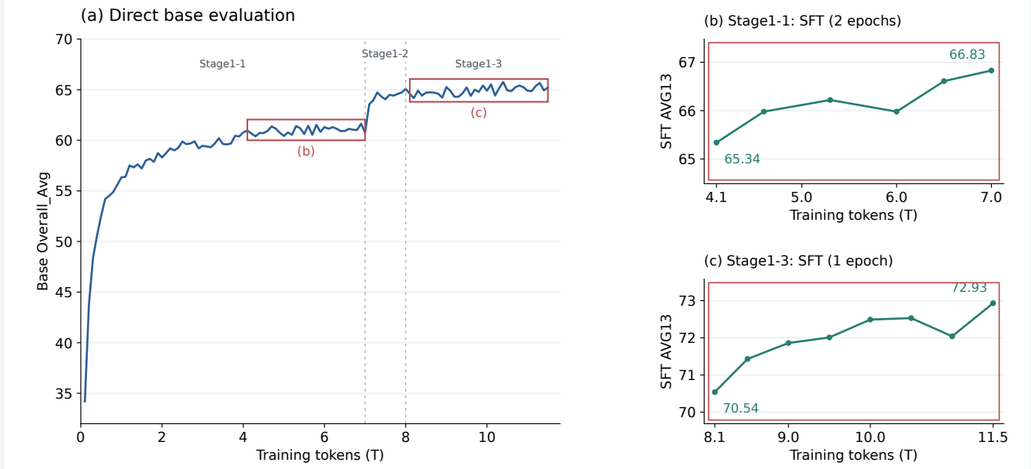


Figure 10: Apparent stagnation in base evaluation can coexist with improving post-SFT performance. (a) Direct base-model Overall_Avg over pre-training; red boxes mark the checkpoint intervals examined in (b) and (c). Each point in (b) and (c) reports SFT AVG13 after applying the same SFT procedure independently to a checkpoint from (a). (b) Stage1-1, steps 123,000–210,000 (4.1T-7T tokens): post-SFT scores improve by 1.49 points despite the near-flat base trend. (c) Stage1-3, steps 243,000–345,000 (8.1T-11.5T tokens): post-SFT scores improve by 2.39 points while base scores fluctuate around a plateau.

Tiny-SFT. Base-model evaluation alone does not fully capture a model's potential after post-training. We

4 潜力循环

预训练的目标不是最大化今天的分数，而是为后训练留下最大潜力。衡量这种潜力可为潜力循环提供信号。我们先介绍三种互补的测量（第 4.1 节），再展示它们揭示了哪些基础评估无法发现的现象，最后说明配方循环如何使用这些测量（第 4.2 节）。

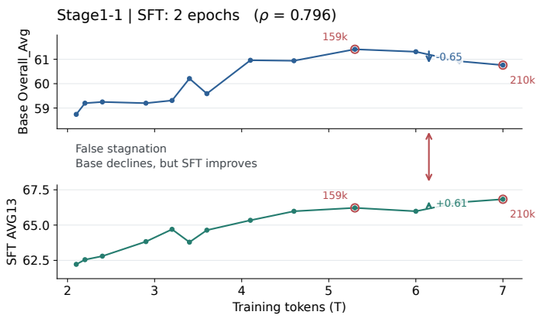
4.1 衡量后训练潜力

我们采用 Tiny-SFT、学习能力、跨评估协议取最大值三种测量。它们分别刻画：有限后训练后的表现、从新增数据中学习的效率，以及不同评估协议下能够展现的能力。

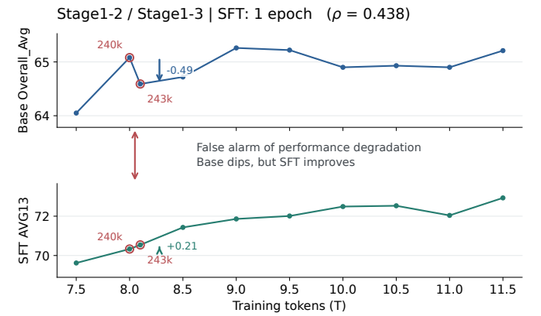
图 9：Tiny-SFT 数据构成及验证。左图：总计 6.33B token，数学、科学（Xu 等，2026a）、代码（Ahmad 等，2025）各约 2.00B，占 31.6%；指令遵循（Pyatkin 等，2025）为 0.331B，占 5.2%。右图：将 Qwen3-30B-A3B 与其基础模型经过一个 epoch 的 Tiny-SFT 后比较。13 项基准均以百分数计，微调模型在其中 10 项上得分更高。这支持将 Tiny-SFT 作为监控有限后训练表现的有效代理。基准包括 AIME 26、HMMT Feb 26、LCB v5/v6、GPQA、MMLU-Pro、IFBench、IFEval、MMLU、GSM8K、MATH-500、MBPP 和 HumanEval。

图 10：基础评估表面停滞，可与 SFT 后表现持续提升同时出现。(a) 预训练期间基础模型 Overall_Avg；红框标出 (b)、(c) 考察的检查点区间。(b)、(c) 每个点都对应独立对某检查点施加相同 SFT 程序后得到的 AVG13。(b) Stage1-1，第 123,000—210,000 步（4.1T—7T token）：基础趋势近乎平坦，但两轮 SFT 后分数从 65.34 到 66.83，提高 1.49 分。(c) Stage1-3，第 243,000—345,000 步（8.1T—11.5T token）：基础分数围绕平台波动，但一轮 SFT 后从 70.54 到 72.93，提高 2.39 分。

Tiny-SFT。仅评估基础模型无法完整反映后训练潜力。因此，我们为每个受评预训练检查点分配少量 SFT 预算，并直接在面向后训练模型的基准上测试。所得分数近似代表该检查点经历一轮有代表性的后训练后的表现，能更现实地衡量下游潜力。



Checkpoint	Base score	Base rank	SFT score	SFT rank
159,000	61.41	1	66.22	3
180,000	61.31	2	65.98	4.5
123,000	60.96	3	65.34	6
138,000	60.94	4	65.98	4.5
195,000	60.92	5	66.61	2
210,000	60.76	6	66.83	1
102,000	60.21	7	63.79	10
108,000	59.59	8	64.64	8
96,000	59.31	9	64.70	7
72,000	59.25	10	62.80	11
66,000	59.20	11.5	62.55	12
87,000	59.20	11.5	63.83	9
63,000	58.74	13	62.22	13



Checkpoint	Base score	Base rank	SFT score	SFT rank
270,000	65.26	1	71.86	6
285,000	65.22	2	72.01	5
345,000	65.21	3	72.93	1
240,000	65.08	4	70.33	9
315,000	64.93	5	72.53	2
300,000	64.90	6.5	72.49	3
330,000	64.90	6.5	72.04	4
255,000	64.72	8	71.43	7
243,000	64.59	9	70.54	8
225,000	64.05	10	69.62	10

Figure 11: Base and post-SFT checkpoint rankings are not interchangeable. *Left*: base Overall_Avg and SFT AVG13 for the same checkpoints. *Right*: all paired checkpoints, sorted by base rank within each group, with their corresponding SFT ranks. Rank 1 is best, ties receive average ranks, and ρ denotes Spearman correlation computed within each group from the reported scores. Base and SFT scores use different benchmark suites.

therefore apply a small SFT budget to each evaluated pre-training checkpoint and directly measure its performance on benchmarks designed for post-trained models. The resulting scores approximate how the checkpoint would perform after a representative post-training phase, providing a more realistic measure of its downstream potential. The SFT corpus spans multiple task categories to approximate the diversity of actual post-training workflows (Figure 9). We note that SFT was not repeated with multiple random seeds, so small differences between adjacent checkpoints should be interpreted with caution. In our experiments, Tiny-SFT helped us identify two interesting patterns that are not readily apparent from base-model evaluation alone:

(1) ***False stagnation.*** We observed that base-model scores can appear to plateau, even as the model continues to improve after SFT. Figure 10 shows two intervals in which average base-model scores fluctuate around a near-plateau, while average scores after SFT continue to improve. In Stage1-1, from step 123,000 to 210,000 (4.1T–7T tokens), the average base-model score remains near 61, while the average score after two epochs of SFT increases from 65.34 to 66.83. In Stage1-3, from step 243,000 to 345,000 (8.1T–11.5T tokens), the average base-model score stays around 64–66, while the average score after one epoch of SFT rises from 70.54 to 72.93. It is worth noting that the base-model and post-SFT scores are computed over different benchmark collections. We therefore compare their trends within each evaluation setting rather than their absolute values. These results show that relying only on base-model evaluation could lead to premature decisions to stop training or unnecessary changes to the training recipe.

(2) ***Ranking disagreement.*** We also found that the relative ranking of checkpoints can change after SFT, compared with their ranking under base-model evaluation. To examine this difference, we paired 23 pre-training checkpoints with their available SFT results and compared their average base-model scores across the 22 public and internal benchmarks listed in Appendix B with their average scores after SFT

图 11：基础模型与 SFT 后检查点排名不能互换。左侧显示同一组检查点的基础 Overall_Avg 和 SFT AVG13；右侧按组内基础分数排名列出全部检查点及其 SFT 排名。第 1 名最佳，并列取平均名次；ρ 为按报告分数计算的组内 Spearman 相关系数。基础与 SFT 分数使用不同基准集合，原始逐项数据见左侧原表。

上组：Stage1-1，SFT 两轮，ρ=0.796。第 159k 至 210k 步，基础分数下降 0.65，SFT 分数上升 0.61，表现为“假性停滞”。下组：Stage1-2/Stage1-3，SFT 一轮，ρ=0.438。第 240k 至 243k 步，基础分数下降 0.49，SFT 分数上升 0.21，表现为“性能退化的假警报”。表头依次为检查点、基础分数、基础排名、SFT 分数、SFT 排名。

（接上页）SFT 语料覆盖多种任务类别，以近似实际后训练流程的多样性（图 9）。须注意，SFT 没有用多个随机种子重复，因此相邻检查点的小幅差异应谨慎解读。实验中，Tiny-SFT 帮助识别了仅靠基础评估不易发现的两种现象。

(1) 假性停滞。即便基础模型分数似乎进入平台，模型在 SFT 后仍可能持续提升。图 10 展示了两个区间：基础平均分在平台附近波动，SFT 后平均分持续增长。Stage1-1 的第 123,000—210,000 步（4.1T—7T token），基础平均分约为 61，而两轮 SFT 后从 65.34 升至 66.83。Stage1-3 的第 243,000—345,000 步（8.1T—11.5T token），基础平均分约为 64—66，而一轮 SFT 后从 70.54 升至 72.93。两种分数由不同基准集合计算，因此比较的是各自设置下的趋势，而非绝对值。这说明，仅依赖基础评估可能导致过早停止训练，或对配方进行不必要的修改。

(2) 排名不一致。检查点在基础评估与 SFT 后的相对排名可能改变。我们将 23 个预训练检查点与可用的 SFT 结果配对，比较附录 B 中 22 项公开及内部基准的基础平均分，以及 SFT 后平均分（图 11）。早期检查点指令遵循能力更弱，因此使用同一 Tiny-SFT 数据，但为不同训练阶段分配不同 SFT 预算。

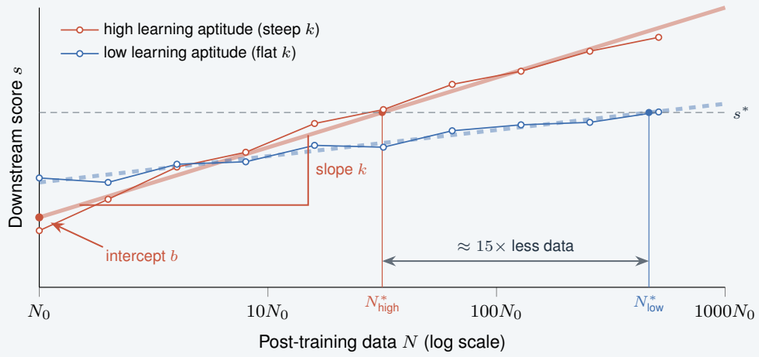


Figure 12: A toy illustration of learning aptitude. Thin lines with hollow markers are raw scores at SFT checkpoints, and each straight line is their least-squares fit of Eq. (1): the intercept b is the score at the first checkpoint N_0, and the slope k is the gain per tenfold increase in post-training data. The model with high learning aptitude starts lowest, yet reaches the target score s^* with about $15\times$ less data than the baseline (Eq. (2)). The high-start, flat model only reaches it at the end of the plotted budget.

(Figure 11). Since checkpoints early in training have weaker instruction-following ability, we used the same Tiny-SFT data but assigned different SFT budgets across training stages. Specifically, we computed rankings separately for 13 Stage1-1 checkpoints evaluated after two SFT epochs and 10 Stage1-2/Stage1-3 checkpoints evaluated after one epoch. The corresponding Spearman rank correlations are 0.796 and 0.438. The rankings are positively associated, particularly in Stage1-1, but they do not provide the same assessment of checkpoint headroom. For example, within the Stage1-1 group, step 159,000 ranks first under base-model evaluation but third after SFT, whereas step 210,000 moves from sixth to first. Between these checkpoints, the average base-model score decreases from 61.41 to 60.76, while the average score after SFT increases from 66.22 to 66.83. These results show that base-model scores alone cannot fully capture differences in post-training potential across checkpoints.

Learning aptitude. A model's initial performance does not necessarily reflect how effectively it can learn from additional post-training data. We therefore measure its *learning aptitude*, which captures how efficiently a model improves as the post-training data budget increases.

To measure learning aptitude, we select a specific domain or task family and fine-tune different base models using the same domain-specific SFT data. We evaluate a sequence of intermediate checkpoints throughout training to track how each model's performance changes as it sees more data. This allows us to fit a learning curve and measure how efficiently the model learns from additional data. For most models, scores grow approximately linearly with the logarithm of the post-training data budget. We therefore fit a log-linear learning curve for each base model m (Figure 12):

$$s_m(N) = b_m + k_m \log_{10} \frac{N}{N_0}, \tag{1}$$

where $s_m(N)$ is the downstream score after N units of post-training data (in practice, SFT steps at a fixed batch size), and N_0 is the data budget of the first evaluated SFT checkpoint. The intercept b_m is the score at this first checkpoint, representing the model's initial performance in the domain. The slope k_m measures the score gain for every tenfold increase in post-training data, capturing how efficiently the model learns from additional data. We estimate both parameters by ordinary least squares over all SFT checkpoints, excluding the untuned base model.

Neither parameter alone captures a model's learning aptitude. A model with a high initial score but a shallow learning curve starts strong but gains little from additional data. In contrast, a model with a lower initial score but a steeper learning curve may eventually surpass it as the data budget increases. We therefore consider both parameters together and estimate the amount of post-training data required for model m to reach a target score s^*:

$$N^*_m(s^*) = N_0 \cdot 10^{(s^* - b_m)/k_m}, \tag{2}$$

We then compare this data requirement with that of a baseline model, using the ratio N^*_{base}/N^*_m to measure data savings. A larger ratio means that the model requires less post-training data to reach the same target performance.

图 12：学习能力示意。带空心标记的细线是各 SFT 检查点原始分数，直线是公式 (1) 的最小二乘拟合。截距 b 是首个检查点 N_0 的分数，斜率 k 是后训练数据每增加十倍带来的分数增益。学习能力强的模型虽然起点最低，但达到目标分数 s* 所需数据约为基线的 1/15（公式 (2)）；起点高而曲线平缓的模型直到所绘预算末尾才达到目标。

（接上页）具体而言，我们分别为经过两轮 SFT 的 13 个 Stage1-1 检查点，以及经过一轮 SFT 的 10 个 Stage1-2/Stage1-3 检查点计算排名，Spearman 相关系数分别为 0.796 和 0.438。排名正相关，尤其是 Stage1-1，但对检查点潜力的判断并不相同。例如 Stage1-1 中，第 159,000 步基础评估第 1、SFT 后第 3；第 210,000 步则从第 6 升至第 1。两者之间，基础平均分由 61.41 降至 60.76，SFT 后却由 66.22 升至 66.83。这说明基础分数无法完全反映不同检查点的后训练潜力差异。

学习能力。模型初始表现不一定反映它从新增后训练数据中学习的效率。因此，我们衡量随着后训练数据预算增大，模型改善表现的效率，即学习能力。

测量时，选定领域或任务族，用相同领域 SFT 数据微调不同基础模型，并评估一系列中间检查点，追踪模型见到更多数据后的变化，据此拟合学习曲线。多数模型的分数与后训练数据预算的对数近似线性，因此对每个基础模型 m 拟合（图 12）：

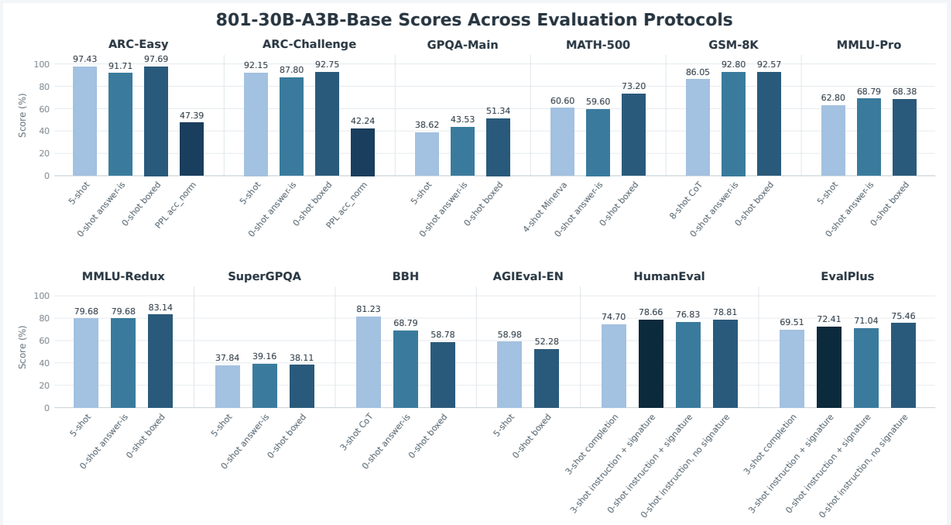
$s_m(N) = b_m + k_m \log_{10}(N/N_0)$ (1)

其中，$s_m(N)$ 是使用 N 单位后训练数据后的下游分数；实践中，N 为固定批次大小下的 SFT 步数。N_0 为首个受评 SFT 检查点的数据预算。截距 b_m 是该检查点的分数，代表模型在此领域的初始表现；斜率 k_m 表示数据每增至十倍的分数增益，刻画从新增数据中学习的效率。用所有 SFT 检查点进行普通最小二乘拟合，排除未经微调的基础模型。

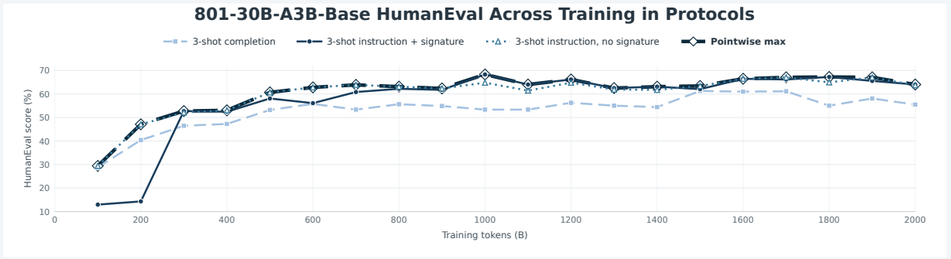
单独一个参数不足以刻画学习能力。高起点、低斜率的模型开始很强，却难从新数据中受益；低起点、高斜率的模型则可能随着数据预算增长最终超过它。因此，我们共同考虑两个参数，估计模型 m 达到目标分数 s* 所需的数据量：

$N^*_m(s^*) = N_0 \times 10^{((s^* - b_m)/k_m)}$ (2)

再以基线与该模型的数据需求比 N^*_{base}/N^*_m 衡量数据节省。比值越大，表示达到同一目标表现所需后训练数据越少。



(a) 801-30B-A3B Base scores across benchmarks under different evaluation protocols. The same benchmark can yield different scores depending on the protocol, showing that a single evaluation setting may not capture the capabilities a model has acquired.



(b) HumanEval scores throughout pre-training under different evaluation protocols. Both the scores and the relative ranking of protocols change across checkpoints. We therefore take the pointwise maximum across available protocols at each checkpoint to obtain a more complete measure of the model's demonstrated capability.

Figure 13: Assessing base-model capabilities with max-over-protocols evaluation.

In this work, we use learning aptitude to compare different models rather than different checkpoints within a single training run. As shown in Section 8, a model with a lower initial score but higher learning aptitude can eventually surpass a model with a higher initial score after sufficient fine-tuning. Applying this measure to checkpoint selection within a single training run is a natural direction for future work.

Max-over-protocols evaluation. A model's evaluation score does not always fully reflect its acquired capabilities, as the extent to which these capabilities are demonstrated can depend on the evaluation protocol. The same benchmark may yield very different scores under perplexity scoring, few-shot prompting, and zero-shot prompting (Figure 13a; protocol definitions in Appendix C). Moreover, the relative performance of these protocols can change across pre-training checkpoints, as illustrated by HumanEval in Figure 13b.

A single evaluation protocol may therefore underestimate model's capability. To obtain a more complete picture, we evaluate each checkpoint under all available protocols and take the highest score for each benchmark. Our goal is to measure the capabilities a model can demonstrate, rather than how well it adapts to a particular evaluation format. Taking the maximum across protocols provides a more complete estimate of the model's capability ceiling. For cross-model comparisons in Section 8, we follow the evaluation protocols specified in that section.

图 13：通过跨评估协议取最大值来评估基础模型能力。

(a) 801-30B-A3B-Base 在不同协议下的基准分数。同一基准会因协议不同而得分不同，单一设置可能无法完整体现模型已学到的能力。图中包括 ARC-Easy、ARC-Challenge、GPQA-Main、MATH-500、GSM8K、MMLU-Pro、MMLU-Redux、SuperGPQA、BBH、AGIEval-EN、HumanEval、EvalPlus。协议包括少样本、零样本、“The answer is”答案格式、boxed 格式、PPL 归一化准确率、Minerva、思维链，以及代码补全/带函数签名指令/不带签名指令；各协议定义见附录 C，原始数值见左图。

(b) 预训练期间不同协议下的 HumanEval 分数。分数和协议之间的相对排名都会随检查点改变。因此，在每个检查点取可用协议中的逐点最大值，以更完整地衡量模型展现出的能力。横轴为训练 token 数（B），纵轴为 HumanEval 分数（%）。

（接上页）本研究用学习能力比较不同模型，而不是同一次训练中的不同检查点。如第 8 节所示，初始分数较低但学习能力较强的模型，经过充分微调后可能超过初始分数更高的模型。将这一测量用于单次训练内部的检查点选择，是自然的未来方向。

跨协议最大值评估。模型评估分数未必完整反映已获得的能力，因为能力展现的程度依赖评估协议。同一基准在困惑度评分、少样本提示和零样本提示下可能得到显著不同的分数（图 13a；协议定义见附录 C）。而且各协议的相对效果还会随预训练检查点改变，如图 13b 的 HumanEval 所示。

因此，单一协议可能低估能力。为获得更完整的图景，我们在全部可用协议下评估每个检查点，并为每个基准选取最高分。目标是衡量模型能够展现的能力，而不是它对某一种评估格式的适应程度。跨协议取最大值提供了更完整的能力上限估计。第 8 节的跨模型比较则使用该节明确规定的评估协议。

4.2 The Recipe Loop

The *recipe loop* brings post-training headroom measurements into the pre-training process, using them as feedback to assess training progress, guide recipe adjustments, and inform subsequent iterations. The two pilots, stopped at 1T and 3T tokens, and the subsequent main pre-training run form the sequence of iterations in this loop (Figure 4).

Throughout this process, we use headroom measurements to track how the model's post-training potential evolves across training stages and feed the results back into the pre-training recipe. Rather than focusing solely on current base-model scores, we consider post-training performance, learning efficiency, and the capabilities the model can demonstrate under different evaluation protocols. This helps us judge whether pre-training is still accumulating useful capabilities and whether the current recipe needs further adjustment.

Based on the issues identified through earlier debugging, we continue to refine the infrastructure and training configurations while developing the training data. In particular, main pre-training introduced new QA data and revised the proportions and sampling weights of web, code, mathematics, books, and papers.

The central idea is that pre-training progress should not be judged solely by current base-model scores, but also by whether the model continues to accumulate capabilities that can be further developed through post-training. Even when base-model evaluation appears to plateau, pre-training may still be making meaningful progress if post-training headroom continues to grow.

5 Experience-to-Metric Loops

Many data-processing decisions in pre-training have traditionally relied on expert experience, such as how to clean and filter data, and how to judge whether a processing strategy works well. We turn these experience-driven tasks into standardized workflows with explicit evaluation metrics. This allows agents to analyze errors, revise their approaches, compare alternatives, and improve their performance through repeated iterations.

We call this the *Experience-to-Metric Loop*, and apply it to three tasks: classifier development, site discovery, and crawl-coverage optimization. All three follow the same principle: turn human experience into measurable tasks, let agents carry out and improve the work, and use evaluation results to guide the next iteration.

5.1 Classifier Development Loop

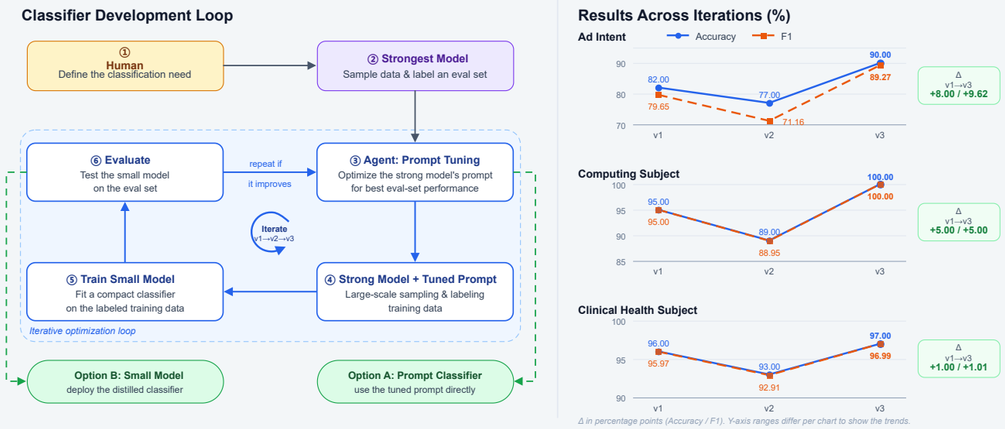


Figure 14: Iterative development of a data classifier. *Left*: a human defines the classification task (1); a strong model samples data and annotates an evaluation set (2); an agent analyzes cases and tunes the labeling prompt (3); the strong model uses that prompt to label training data (4); a compact classifier is distilled from these labels (5) and evaluated (6). The component can be deployed as a tuned prompt (Option A) or a distilled model (Option B). *Right*: accuracy and F1 over three iterations (v1–v3) for classifiers of ad intent, computing subject, and clinical health subject.

Earlier work on *Data Darwinism* provides a framework for the first question. It introduces a ten-level taxonomy (L0–L9) that organizes data processing operations, from data acquisition and format normalization to filtering,

4.2 配方循环

配方循环将后训练潜力测量纳入预训练，以此反馈评估进展、指导配方调整并支持后续迭代。分别止于 1T、3T token 的两次试训和后续主预训练，构成此循环的迭代序列（图 4）。

整个过程中，我们用潜力测量追踪不同训练阶段的后训练潜力演化，再将结果反馈至配方。关注的不只是当前基础分数，还包括后训练表现、学习效率，以及不同协议下展现的能力，以判断预训练是否仍在积累有用能力、当前配方是否需要调整。

依据早期调试发现的问题，我们在开发数据的同时，继续完善基础设施和训练配置。主预训练尤其引入了新 QA 数据，并调整网页、代码、数学、图书和论文的比例及采样权重。

核心思想是：判断预训练进展，既要看当前基础分数，也要看模型是否仍在积累能经后训练进一步开发的能力。即使基础评估进入平台，只要后训练潜力继续增加，预训练就仍可能取得有意义的进展。

5 经验量化循环

预训练中的许多数据处理决策传统上依赖专家经验，例如如何清洗、过滤数据，如何判断处理策略效果。我们将这些任务转成具备明确评估指标的标准工作流，让智能体能够分析错误、修订方法、比较替代方案，并在反复迭代中改进。

我们将其称为“经验量化循环”，应用于分类器开发、站点发现和抓取覆盖优化。三者遵循同一原则：将人类经验转化为可测量任务，让智能体执行并改进工作，再以评估结果指导下一轮迭代。

5.1 分类器开发循环

图 14：数据分类器的迭代开发。左图六步：①人类定义分类需求；②强模型采样并标注评估集；③智能体分析案例、调优标注提示词；④强模型用调好的提示词大规模标注训练数据；⑤训练紧凑分类器；⑥在评估集上评测，如仍有改进则继续迭代。部署可选调优后的提示词分类器（A），或蒸馏的小模型（B）。

右图为三个分类器 v1—v3 的准确率/F1（%）：广告意图为 82.00/79.65、77.00/71.16、90.00/89.27，总提升 8.00/9.62 个百分点；计算机主题为 95.00/95.00、89.00/88.95、100.00/100.00，提升 5.00/5.00；临床健康主题为 96.00/95.97、93.00/92.91、97.00/96.99，图中报告提升 1.00/1.01。各图纵轴范围不同，以显示变化趋势。

此前的 Data Darwinism 为第一个问题提供了框架：将数据处理分为 L0—L9 十级，从数据获取、格式规范化到过滤、模型驱动优化和合成（Qin 等，2026a）。沿用其在 daVinci-LLM 中的做法（Qin 等，2026b），我们用这一框架指导流水线设计。

model-driven refinement, and synthesis (Qin et al., 2026a). Following its use in daVinci-LLM (Qin et al., 2026b), we use this framework to guide pipeline design. For example, OCR belongs to format normalization (L1), rule-based and model-based filtering correspond to L2–L3, and refinement for cleaning or repairing content falls under L4. This taxonomy helps us define what each component should do before choosing its model, prompt, or implementation.

DataEvolve explores a complementary question: how data-curation strategies can improve through repeated evaluation and diagnostic feedback (Mi et al., 2026). Inspired by this approach, we develop classifiers, filters, OCR models, and rewriters through a shared evaluation workflow.Data Darwinism guides the choice of processing operations, while the evaluation loop helps determine whether a particular implementation performs its intended task well. Figure 14 illustrates this process for classifiers, with three stages (six steps in the figure).

Step 1: Define the task and construct an evaluation set. A researcher specifies what the classifier should do. A strong model then annotates representative samples to create a reference evaluation set. This establishes quality criteria and provides a common basis for comparing prompts, models, and processing rules. Since the reference labels come from a strong model, the evaluation measures agreement with that model rather than with human judgment.

Step 2: Analyze errors and improve the component. Agents analyze hundreds to thousands of examples in parallel, identify recurring error patterns, and use these findings to revise prompts, compare models, or adjust processing rules. Each revision is evaluated against the same task definition and evaluation set, allowing us to determine whether the changes actually improve performance.

Step 3: Validate and deploy. The revised component is evaluated before deployment. For a classifier, the result can be used as a tuned labeling prompt or distilled into a smaller model using training data labeled by the strong model. The distilled classifier is evaluated and the loop continues revisions improve its performance. The evaluation catches regressions: in Figure 14, all three classifiers lost accuracy at v2 before improving at v3.

In under three months, we built six data pipelines, with agents contributing 36 model-selection decisions as well as prompt optimization and classifier construction.

5.2 Web Acquisition: Site Discovery and Crawl-Coverage Loops

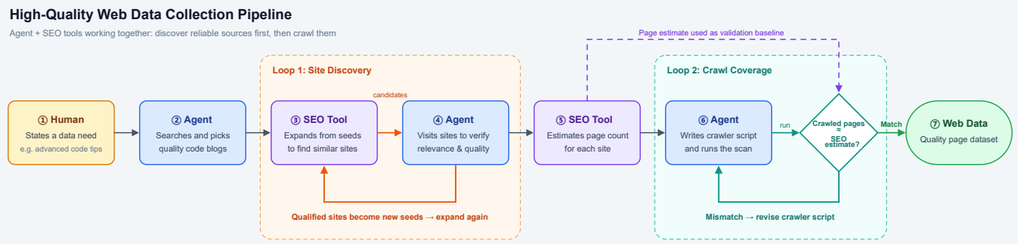


Figure 15: Structure-aware web data acquisition. A human requirement seeds an agent-driven pipeline with two loops. Loop 1 (site discovery) expands and verifies related sites through the web's link structure. Loop 2 (crawl coverage) crawls each confirmed site to a coverage target informed by its SEO page estimate. Links, search indexes, and robots directives are treated as structured signals rather than discarded.

The public web is a major source of pre-training data, but systematically finding high-quality sources and collecting their content as completely as possible remain challenging. A common approach starts from seed documents, uses a classifier to retrieve relevant pages, and identifies the domains where these pages are concentrated (Shao et al., 2024). However, two questions remain: how can we continuously discover more valuable sites, and how can we tell whether we have collected enough data from each site?

We address these questions with two connected agent-driven loops: the *site discovery loop*, which finds high-quality data sources, and the *crawl-coverage loop*, which aims to collect their content as completely as possible. Both use information from search-engine-optimization (SEO) tools, which provide related-site information and estimates of indexed page counts, as signals to guide discovery and measure crawl coverage. Figure 15 shows the pipeline.

Loop1: Site Discovery Loop. The goal of site discovery is to expand a small set of high-quality seed sites into a broader collection of relevant data sources. We use SEO tools to identify candidate sites related to the seeds through the web's link structure. An agent then visits each candidate to check its relevance and quality, adding verified sites to the seed set for the next round of discovery. We measure the effectiveness of each round by its *verified yield*, the number of discovered sites that pass verification. Each round's verified sites become seeds for the next round, allowing discovery to expand iteratively. The process stops when a round produces no new qualified sites. This

例如，OCR 属于格式规范化（L1），规则和模型过滤对应 L2—L3，清洗或修复内容的生成式优化属于 L4。该分类体系使我们在选择模型、提示词或实现方式之前，先明确各组件职责。

DataEvolve 研究互补问题：数据整理策略如何通过反复评估和诊断反馈改进（Mi 等，2026）。受其启发，我们用共享评估流程开发分类器、过滤器、OCR 模型和改写器。Data Darwinism 指导选择处理操作，评估循环则判断具体实现是否胜任预定任务。图 14 以分类器展示三阶段（图中六步）流程。

步骤 1：定义任务、构建评估集。研究者明确分类器职责，强模型标注代表样本，形成参考评估集，为比较提示词、模型和规则提供共同质量标准。参考标签由强模型生成，因此评估衡量的是与该模型的一致程度，而非与人类判断的一致程度。

步骤 2：分析错误、改进组件。智能体并行分析数百至数千样本，识别反复出现的错误模式，修订提示词、比较模型或调整规则。每次修订均依据相同任务定义和评估集检验，确定是否真正改善。

步骤 3：验证与部署。修订组件须先评估再部署。分类器可直接采用调好的标注提示词，也可利用强模型标注数据蒸馏成小模型；小分类器继续评估和迭代。评估能够捕捉退步：图 14 中三个分类器在 v2 均准确率下降，随后 v3 提升。

不到三个月，我们构建了六条数据流水线，智能体贡献了 36 项模型选择决策，并完成提示词优化和分类器构建。

5.2 网页获取：站点发现与抓取覆盖循环

图 15：利用网页结构的数据获取。人类需求启动智能体驱动流水线。循环 1 沿网页链接结构扩展并验证相关站点；循环 2 依据 SEO 页面数估计设定覆盖目标并抓取。链接、搜索索引、robots 指令均被当作结构化信号保留。

流程为：①人类提出数据需求，如高级编程技巧；②智能体搜索并挑选优质代码博客；③SEO 工具从种子拓展相似站点；④智能体访问并验证相关性与质量；⑤SEO 工具估计站点页面数；⑥智能体编写并执行爬虫；⑦形成优质网页数据集。合格站点成为新种子继续扩展；抓取数与估计不符时，修订爬虫。

公开网页是主要预训练数据源，但系统发现优质来源、尽可能完整地收集内容仍有困难。常见做法从种子文档出发，用分类器检索相关页面，识别这些页面集中的域名（Shao 等，2024）。仍有两个问题：怎样持续发现更多有价值站点？怎样判断每个站点是否收集充分？

我们用相连的站点发现与抓取覆盖循环解决这两个问题。二者都借助搜索引擎优化（SEO）工具提供的相关站点信息和索引页面数估计，指导发现并衡量覆盖（图 15）。

循环 1：站点发现。将少量优质种子扩成更广的相关来源集合。SEO 工具沿链接结构找候选站点，智能体逐一验证相关性和质量，将通过验证者加入下一轮种子。每轮效果以“验证产出量”衡量，即发现且通过验证的站点数。当某轮不再产生新合格站点时停止。这将“接下来去哪里找”的经验决策转化为可测量、可重复的过程。

turns the experience-based decision of where to look next into a measurable and repeatable discovery process.

Loop2: Crawl-Coverage Loop. Once valuable sites have been identified, the next task is to collect their content as completely as possible. For each confirmed site, an agent writes and runs a crawler, then compares the number of retrieved pages with the page-count estimate provided by SEO tools. We use the *coverage ratio* as the evaluation metric, defined as the number of retrieved pages divided by the estimated total number of pages:

$$\text{Coverage Ratio} = \frac{\text{Retrieved Pages}}{\text{Estimated Pages}}$$

Although SEO page counts are estimates, they provide a quantitative reference for assessing crawl coverage. The agent compares the retrieved count with this estimate, investigates potential gaps, revises the crawler, and runs it again to measure the resulting coverage. In this way, crawling becomes an iterative process driven by measurable coverage rather than an informal judgment of when a site is sufficiently crawled.

Closing the loops. The site discovery and crawl-coverage loops address two complementary challenges in web data acquisition: where to find more valuable data sources and how to collect their content as completely as possible. Together with the classifier development loop, they follow the same principle: turn experience-driven tasks into workflows with explicit evaluation metrics, so that agents can execute the work, assess the results, and improve their approaches through repeated iterations. Specifically, classifier development uses task-specific evaluation scores, site discovery uses the verified yield of each round, and web crawling uses the coverage ratio. The key is to give tasks that once relied on expert judgment explicit objectives and evaluation procedures. Agents can then use feedback to determine how well a task has been completed and what to improve next. This turns experience-based decisions into systematic and repeatable processes, making pre-training more scientific one component at a time.

6 Scaling Agentic Data into Pre-Training

Agentic data, consisting of trajectories generated by agents solving real-world tasks with verifiable outcomes, has proven effective in post-training (Pan et al., 2025; Yang et al., 2025b). Real-world software development provides a rich source of such tasks and offers a new opportunity for pre-training. Our goal is to scale this data source to hundreds of billions of tokens while preserving task diversity and data quality.

However, achieving this scale requires more than conventional data-processing pipelines. Real-world software tasks are diverse and involve complex task construction, execution environments, testing, and integration across agent harnesses. We argue that *extensive use of agents is a necessary condition for scaling agentic data into pre-training*. The main bottleneck is not generating a handful of high-quality examples, but systematically transforming diverse real-world software development activities into executable, verifiable, and scalable training data.

6.1 Why Agentic Data Requires Agents

Complex task construction and validation. Real software development offers a rich source of tasks for agentic data. Recent work has explored several directions: making development tasks executable through automated environment and evaluation-script construction (Fu et al., 2026b), preserving development context and execution feedback in mid-training trajectories (Zeng et al., 2026), and connecting related PRs into longer task chains (Jiang et al., 2026).

Scaling this data source to pre-training requires handling the full diversity of real-world development activity. This includes both PRs with clear task definitions and well-maintained tests (Jimenez et al., 2024; Pan et al., 2025; Badertdinov et al., 2025), and those with incomplete descriptions, implicit requirements, or multiple work items bundled together. For each PR, the pipeline must construct three mutually consistent components: a reproducible execution environment, an instruction specifying what the agent is expected to do, and tests and evaluation scripts that verify whether the task requirements are met.

These components must be developed together. For example, a repository's CI may inject environment variables that were omitted from the constructed environment, causing the evaluation to fail for reasons unrelated to the code change. In another case, a test may require a function with a specific name, while the instruction describes only the desired behavior, causing a valid implementation to be rejected. A single PR may also bundle multiple work items, leaving the instruction, environment, and tests with inconsistent task boundaries.

Task construction therefore involves more than simply extracting a task description from a PR. The environment, instruction, and evaluation must agree on what the task requires. Agents need to inspect repository context, identify inconsistencies across these components, and revise them accordingly. At scale, this process must handle the long tail of repository-specific conventions and development practices.

循环 2：抓取覆盖。发现有价值站点后，智能体为每个站点编写并运行爬虫，将抓取页数与 SEO 估计页数比较，采用：

覆盖率 = 已抓取页面数 / 估计总页面数。

SEO 页数虽只是估计，仍提供定量参照。智能体比较数量、调查缺口、修订爬虫并重跑，测量新覆盖率。抓取由此由可测量覆盖驱动，而非凭感觉判断"抓得够不够"。

闭合循环。站点发现解决去哪里找更多优质来源，抓取覆盖解决如何尽可能完整收集。与分类器开发一起，三者将经验任务转为有明确评估指标的工作流，使智能体执行、评估并迭代改进。分类器开发使用任务评估分数；站点发现使用每轮验证产出量；网页抓取使用覆盖率。关键是为专家判断型任务明确目标和评估程序，让智能体依据反馈判断完成程度和下一步改进方向。这样，经验决策成为系统、可重复的过程，从一个个组件入手使预训练更科学。

6 将智能体数据扩展到预训练规模

智能体数据由解决真实任务、结果可验证的轨迹构成，已证明对后训练有效（Pan 等，2025；Yang 等，2025b）。真实软件开发提供了丰富任务和预训练新机会。目标是在保持任务多样性、数据质量的同时，将此数据源扩至数千亿 token。

达到这一规模需要超越传统处理流水线。真实软件任务多样，涉及复杂的任务构建、执行环境、测试与跨运行框架集成。我们认为，广泛使用智能体是将此类数据扩至预训练规模的必要条件。主要瓶颈不是生成少量优质样本，而是系统地将多样的软件开发活动转成可执行、可验证、可扩展的数据。

6.1 为什么智能体数据需要智能体

复杂的任务构建与验证。近期研究探索了自动构建环境和评估脚本以让开发任务可执行（Fu 等，2026b），在中期训练轨迹中保留开发上下文及执行反馈（Zeng 等，2026），以及将相关 PR 连接成长任务链（Jiang 等，2026）。

扩至预训练规模须处理真实开发活动的全部多样性，包括描述清楚、测试维护良好的 PR（Jimenez 等，2024；Pan 等，2025；Badertdinov 等，2025），也包括描述不完整、需求隐含或捆绑多个工作项的 PR。每个 PR 都须构建三个相互一致的组件：可复现执行环境、明确预期工作的指令、验证需求满足情况的测试及评估脚本。

这些组件必须共同开发。例如，仓库 CI 注入的环境变量若在构建环境中遗漏，评估会因与代码无关的原因失败；测试若要求特定函数名，而指令只描述行为，正确实现也可能被拒绝；一个 PR 含多个工作项时，指令、环境和测试还可能划定不同任务边界。

任务构建因而不仅是从 PR 提取描述。环境、指令和评估必须对任务要求一致。智能体需要检查仓库上下文，识别并修正不一致；规模化时还要处理各仓库特有惯例和开发实践的长尾。

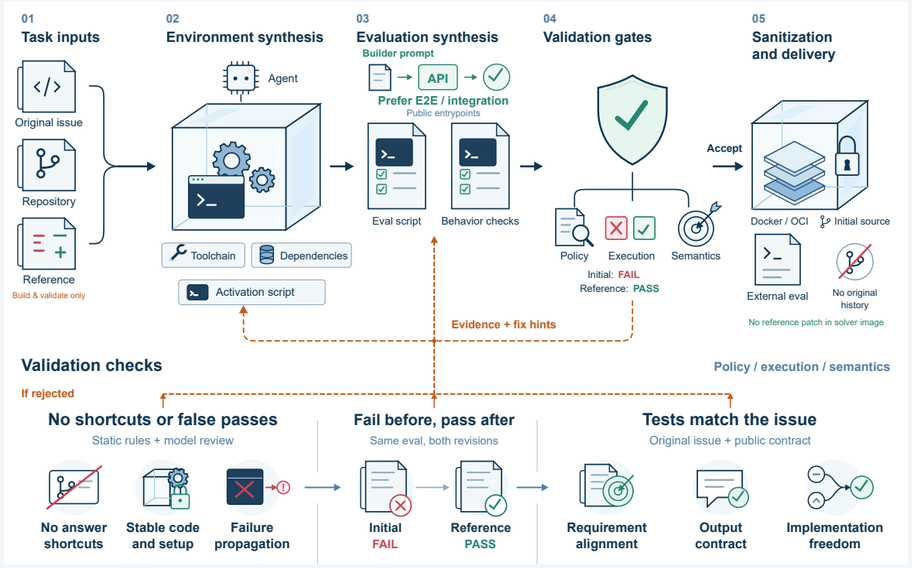


Figure 16: Overview of agentic-data pipeline. The key component is the Validation gates, which admits a task only when it clears policy, execution, and semantic checks and the same evaluation fails on the initial code but passes on the reference, and sanitization into a Docker/OCI image that keeps the external eval but strips the reference patch and original history. Rejected tasks return with evidence and fix hints (dashed path).

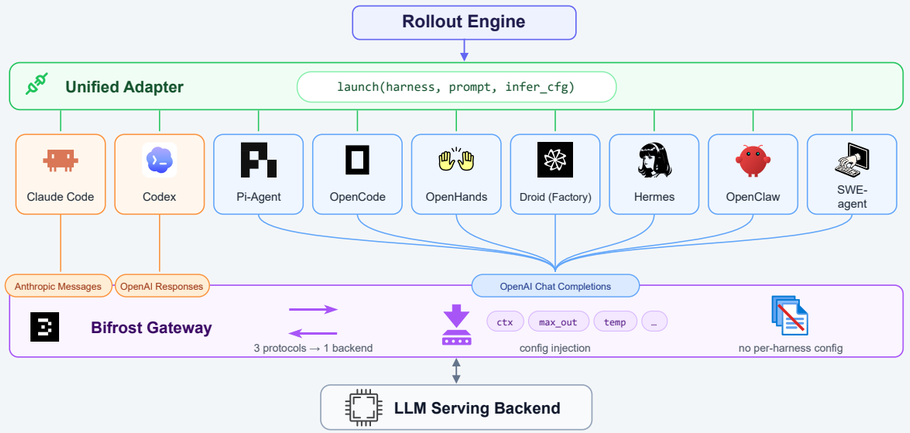


Figure 17: The multi-harness rollout layer. A single `launch(harness, prompt, infer_cfg)` call in the unified adapter drives all nine integrated harnesses, which among them speak three API protocols (Anthropic Messages (Anthropic, 2026), OpenAI Responses (OpenAI, 2026b), and OpenAI Chat Completions (OpenAI, 2026a)). The Bifrost gateway (Maxim AI, 2025) normalizes the three protocols onto one LLM serving backend and injects inference settings (context length, maximum output, temperature) centrally.

Multi-harness integration and maintenance. Even after constructing a task, generating diverse trajectories at scale requires supporting multiple harnesses during execution. We integrated nine harnesses into our agentic data pipeline: Claude Code (Anthropic, 2025), Codex (OpenAI, 2025), Pi-Agent (Earendil Works, 2025), Open-

图 16：智能体数据流水线概览。

①任务输入：构建器提示词、智能体 API、原始 issue、仓库，以及仅供构建和验证使用的参考修改。

②环境合成：工具链、依赖与激活脚本。

③评估合成：优先端到端/集成测试，经公开入口检查行为，生成评估脚本。

④验证关卡：任务必须同时通过策略、执行、语义检查，且相同评估必须在原始代码上失败、参考修改后通过。策略检查用静态规则和模型审查，排除答案捷径、虚假通过，保证代码与配置稳定及失败正确传播；执行检查要求同一评估在修改前失败、修改后通过；语义检查依据原始 issue 和公开契约，确保需求一致、输出契约正确，并保留实现自由。

⑤清理与交付：打包为 Docker/OCI 镜像，保留初始源代码及外部评估，剥离参考补丁和原始历史。未通过任务携带证据和修复提示返回修订（虚线路径）。求解器镜像中没有参考补丁。

图 17：多运行框架轨迹生成层。统一适配器通过一次 launch(harness, prompt, infer_cfg) 调用驱动九种集成框架。框架分别使用 Anthropic Messages（Anthropic，2026）、OpenAI Responses（OpenAI，2026b）、OpenAI Chat Completions（OpenAI，2026a）三种 API 协议。Bifrost 网关（Maxim AI，2025）将三协议归一到同一个 LLM 服务后端，集中注入上下文长度、最大输出、温度等设置，无须逐框架配置。

九种框架为 Claude Code、Codex、Pi-Agent、OpenCode、OpenHands、Droid（Factory）、Hermes、OpenClaw、SWE-agent。

多框架集成与维护。任务构建完成后，要大规模生成多样轨迹，还需要支持多种运行框架。我们集成了 Claude Code（Anthropic，2025）、Codex（OpenAI，2025）、Pi-Agent（Earendil Works，2025）、OpenCode（Anomaly，2025）、OpenHands（Wang 等，2025）、Droid（Factory AI，2025）、Hermes（Nous Research，2025）、OpenClaw（OpenClaw contributors，2026）和 SWE-agent（Yang 等，2024）。

Code (Anomaly, 2025), OpenHands (Wang et al., 2025), Droid (Factory AI, 2025), Hermes (Nous Research, 2025), OpenClaw (OpenClaw contributors, 2026), and SWE-agent (Yang et al., 2024). Each harness has its own interfaces, dependencies, and failure modes. Even the same model can behave differently across harnesses. In addition, harness updates and new plugins introduce further configuration changes.

As a result, the combinations of models, harnesses, and versions require ongoing integration and maintenance. We use a unified adapter to launch rollouts across different harnesses, with agents helping diagnose execution failures, revise integrations, and validate updated configurations. As these combinations grow and change, manual maintenance becomes unsustainable. Agents are therefore necessary for sustaining large-scale trajectory generation.

Table 4: Agentic anti-hack patterns, grouped by category. Each is a concrete way a model can pass a synthesized task without solving it. Agents scan trajectories for these behaviors and flag candidates for review or re-synthesis.

Category	Specific behavior
Pull the reference fix from version control	Fetch its own merged PR diff or the fixing commit patch Recover the fix from unreachable Git objects (`git fsck`, `cat-file`)
Fetch upstream code over the network	Download files from `raw.githubusercontent`, the GitHub API, or PyPI JSON Clone, fetch, or add a remote pointing at the upstream repository
Install an already-fixed version	`pip download` or `install --target` with a pinned version range Pull packages through conda, apt, or an external mirror (aliyun, tuna, ustc)
Find a solved copy already on disk	Read installed, cached, or `site-packages` source copies Scan from root with `find`, `grep`, or `rg`, or walk the conda tree
Tamper with the evaluation	Write or rewrite test files Deselect or skip the failing tests (`pytest --deselect`)
Reason then exfiltrate	Plan in chain-of-thought to leave the sandbox, then issue a network call

Trajectory quality control and anti-hacking. Anti-hacking is critical to producing high-quality agentic data. Models can exploit evaluation shortcuts in many ways (Zhong et al., 2025; jacobkahn, 2025), and agents are essential for detecting such behavior at scale. In a data synthesis pipeline, agents must distinguish between two types of problems: *hacks*, where a model bypasses the task rather than genuinely solving it, and *synthesis defects*, where the execution environment, tests, or task instructions themselves are flawed. This distinction is particularly difficult because a model's cheating behavior can mask defects in the task, making an invalid task appear solvable. To address this, agents scan execution trajectories for anomalous behaviors, such as accessing Git repositories and logs, downloading related packages, searching for code commits, or using shell commands that bypass the actual task logic. Suspicious cases are flagged for human review or re-synthesis (Table 4).

Thus, producing agentic data at scale requires not only making tasks executable, but also verifying that models genuinely solve them and that the resulting trajectories are trustworthy.

6.2 Scaling Agentic Data for Pre-Training

Through the task-construction, multi-harness execution, and quality-control processes described above, we established a pipeline for producing agentic data at scale.

More than 84% of the raw pull requests we collected were successfully converted into executable tasks. For these PRs, we constructed execution environments, extracted task requirements, and synthesized corresponding tests (Figure 18). We then rolled out the resulting 220K executable PRs across nine harnesses. The trajectories were generated using small and medium-sized open-weight models, including DeepSeek-Flash and GPT-OSS. After layered filtering, we obtained 1.1M agent trajectories that met our pre-training quality standards, totaling more than 100B tokens, with approximately 91K tokens per trajectory on average. Together with trajectories from other diverse sources, the total volume of agentic data available for pre-training reached **134B** tokens. Importantly, these trajectories are not merely an auxiliary ingredient for post-training, but a *first-class data source for pre-training*.

Agent-driven data pipelines represent a shift in how training data is produced. As models become more capable, they can solve harder tasks and generate richer training signals. At the same time, as harnesses and tools evolve, agents can adapt the underlying integration layer. Beyond adding more harnesses, the plugin ecosystems within each harness are also growing rapidly, further expanding the space of configurations that must be supported.

各框架具有不同接口、依赖和失败模式。同一模型在不同框架下也可能表现不同；框架更新及新插件进一步带来配置变化。

因此，模型、框架与版本的组合需要持续集成和维护。统一适配器负责跨框架启动轨迹生成，智能体协助诊断执行故障、修订集成和验证新配置。组合不断增加变化，人工维护难以持续，智能体因而是大规模轨迹生成得以维持的必要条件。

表 4：智能体数据中的防作弊检查模式。以下均是模型可能绕过真正求解、直接通过合成任务的具体行为；智能体扫描轨迹并标记供复核或重新合成。

从版本控制获取参考修复：获取该任务已合并 PR 的差异或修复提交补丁；从不可达 Git 对象恢复修复（git fsck、cat-file）。

经网络取得上游代码：从 raw.githubusercontent、GitHub API 或 PyPI JSON 下载文件；克隆、fetch 或添加指向上游仓库的远端。

安装已修复版本：按固定版本范围 pip download 或 install --target；从 conda、apt 或外部镜像（aliyun、tuna、ustc）获取软件包。

在磁盘寻找已解决副本：读取已安装、缓存或 site-packages 中的源代码；用 find、grep、rg 从根目录扫描或遍历 conda 树。

篡改评估：新建或重写测试文件；取消选择或跳过失败测试（pytest --deselect）。

先推理后外传：在思维链中计划离开沙箱，随后发出网络调用。

轨迹质量控制与反作弊。模型可利用多种评估捷径（Zhong 等，2025；jacobkahn，2025），智能体对规模化检测至关重要。合成流水线中必须区分两类问题：作弊，即绕过任务而非真正求解；合成缺陷，即环境、测试或指令本身有误。这尤其困难，因为作弊可能掩盖任务缺陷，使无效任务看起来可解。

智能体因此扫描访问 Git 仓库和日志、下载相关软件包、搜索代码提交、以 shell 命令绕过实际任务逻辑等异常行为，并将可疑案例交由人类复核或重新合成（表 4）。规模化数据生产不仅要使任务可执行，还须确认模型真正解决了任务、轨迹值得信任。

6.2 面向预训练的智能体数据扩展

通过上述任务构建、多框架执行与质量控制，我们建立了规模化数据流水线。

收集的原始 PR 超过 84% 被成功转成可执行任务：构建环境、提取需求并合成测试（图 18）。22 万个可执行 PR 在九种框架中展开，使用 DeepSeek-Flash、GPT-OSS 等中小型开放权重模型生成轨迹。经分层过滤，得到 110 万条满足预训练质量标准的轨迹，总计超过 100B token，平均每条约 91K token。加上其他多样来源，预训练可用智能体数据总计 134B token。这些轨迹不是仅供后训练的辅助材料，而是预训练的一等数据源。

智能体驱动流水线改变了训练数据生产方式。模型能力变强，便可解决更难任务、产生更丰富信号；运行框架与工具演化时，智能体还能适配底层集成。除框架数量增加外，各框架内的插件生态也迅速扩张，进一步扩大须支持的配置空间。

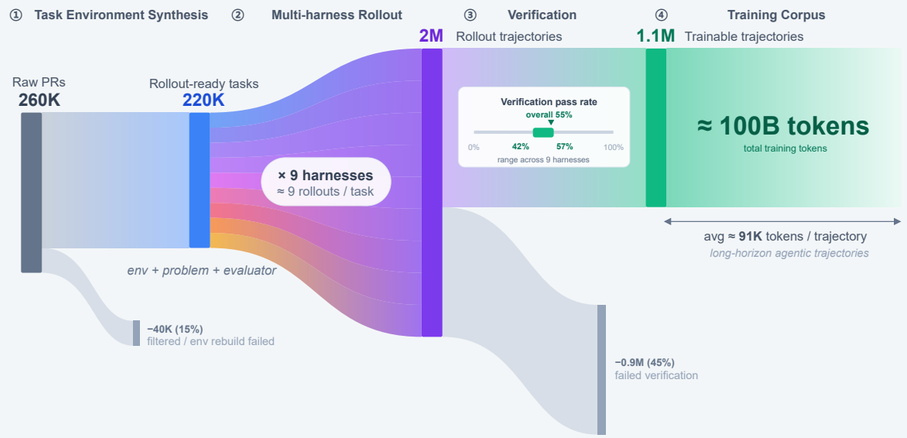


Figure 18: Funnel chart of agentic data pipeline. 260K raw PRs are filtered into 220K executable tasks. Rolling each task out under 9 harnesses gives about 2M trajectories. After verification, 1.1M trajectories remain, totaling more than 100B training tokens (about 91K tokens per trajectory on average). Tasks and trajectories are drawn on separate scales.

We expect agent-driven data pipelines to become an increasingly important source of high-quality, diverse, and verifiable training data for future frontier pre-training runs.

7 Training Recipe of 801-30B-A3B

We adopt the Qwen3-30B-A3B (Yang et al., 2025a) Mixture-of-Experts architecture, with approximately 30 billion total parameters and 3 billion active parameters per token. Figure 21a summarizes the selected training recipe: three pre-training sub-stages followed by two context-extension stages. All stages use a global batch of 33,554,432 tokens and Muon (Jordan et al., 2024; Liu et al., 2025), with AdamW (Loshchilov and Hutter, 2019) for the excluded parameter groups, including embeddings, the output head, normalization parameters, biases, and routers.

Stage1 maintains a sequence length of 4,096 tokens. Stage1-1 trains from scratch through step 210,000 (approximately 7T tokens); Stage1-2 continues to step 240,000 (approximately 8T), and Stage1-3 supplies the checkpoint at step 345,000 (approximately 11.5T). At these transitions, the data mixture is updated and the RoPE (Su et al., 2024) base increases from 10^4 to 5×10^5 and 5×10^6, following the base-frequency adjustment for longer contexts (Xiong et al., 2024). The learning rate remains 5×10^{-4} after an 2,000-step warmup; Stage1-2 and Stage1-3 resume the optimizer and training state without a warmup. Stage1 uses global-batch load balancing (Fedus et al., 2022; Qiu et al., 2025) with coefficient 10^{-4} and router z-loss (Zoph et al., 2022) with coefficient 10^{-3}.

Stage2 extends the sequence length to 32,768 tokens and is configured for 3,000 steps (approximately 100B tokens). Stage3 further extends it to 262,144 tokens and is configured for 6,000 steps (approximately 200B tokens). Both retain the RoPE base of 5×10^6, initialize from the preceding stage's weights with fresh optimizer state, use a 1,000-step warmup, and disable both load-balancing and router z-loss penalties. Stage2 keeps the learning rate at 5×10^{-4} after warmup, whereas Stage3 applies cosine decay to 5×10^{-5}. Stage3 also enables length-aware packing. The three stages together correspond to approximately 11.8T tokens.

The training corpus draws from multiple data types (shown in Figure 20): web text, PDF documents, question-answer pairs, code repositories, and agentic trajectories. Beyond the agentic data pipeline detailed in Section 6, each major category branches into sub-pipelines (shown in Figure 19). *Web data* is cross-classified by advertisement presence, content type tags, domain category, and language, creating a Cartesian product of fine-grained cells that serve as the atomic units for review and mixture tuning. *Math data* undergoes model-driven rewriting to increase diversity and refine equation and symbols. *PDF data* passes through OCR extraction, vision-language model (VLM) refinement to correct OCR errors, and multi-stage quality filters. *QA data* pipelines classify whether the problem's context is complete or not and rollout the CoT with monitoring pass rate to probe their difficulty. *Code data* includes static code files and pull-request records.

图 18：智能体数据流水线漏斗。26 万原始 PR 经筛选成为 22 万可执行任务，约 4 万（15%）因过滤或环境重建失败被移除。每个任务在九种框架下约展开九次，共约 200 万条轨迹。约 90 万（45%）未通过验证，留下 110 万条可训练轨迹，总计超过 100B token，平均约 91K token/条。总验证通过率 55%，九种框架间约为 42%—57%。任务数与轨迹数使用不同绘图尺度。

我们预计，智能体驱动流水线会成为未来前沿预训练中越来越重要的优质、多样、可验证数据来源。

7 801-30B-A3B 的训练配方

采用 Qwen3-30B-A3B（Yang 等，2025a）MoE 架构，总参数约 300 亿，每个 token 激活约 30 亿。图 21a 总结配方：三个预训练子阶段，之后两个上下文扩展阶段。所有阶段全局批次均为 33,554,432 token，使用 Muon（Jordan 等，2024；Liu 等，2025）；嵌入、输出头、归一化、偏置和路由器等排除参数组使用 AdamW（Loshchilov 和 Hutter，2019）。

Stage1 序列长度为 4,096。Stage1-1 从零训练至第 210,000 步（约 7T token）；Stage1-2 至第 240,000 步（约 8T）；Stage1-3 提供第 345,000 步检查点（约 11.5T）。阶段切换时更新数据混合，RoPE（Su 等，2024）基数依次从 10^4 增至 5×10^5、5×10^6，遵循面向长上下文的基频调整（Xiong 等，2024）。初始预热 2,000 步后，学习率保持 5×10^-4；Stage1-2、1-3 恢复优化器及训练状态，不再预热。Stage1 使用全局批次负载均衡（Fedus 等，2022；Qiu 等，2025），系数 10^-4；路由器 z-loss（Zoph 等，2022）系数 10^-3。

Stage2 将长度扩至 32,768，配置 3,000 步（约 100B token）。Stage3 扩至 262,144，配置 6,000 步（约 200B）。二者保持 RoPE 基数 5×10^6，从前一阶段权重初始化，但重置优化器状态，预热 1,000 步，关闭负载均衡与路由器 z-loss 惩罚。Stage2 预热后学习率为 5×10^-4；Stage3 以余弦衰减至 5×10^-5，并启用长度感知打包。三大阶段合计约 11.8T token。

训练语料包含网页文本、PDF、问答对、代码仓库和智能体轨迹（图 20）。除了第 6 节的智能体流水线，各大类还分出子流水线（图 19）。网页按广告存在性、内容标签、领域与语言交叉分类，生成笛卡尔积细粒度单元，作为审查和混合调节的最小单位。数学通过模型改写增加多样性、完善公式符号。PDF 经过 OCR、视觉语言模型（VLM）修复 OCR 错误及多级质量过滤。QA 判断问题上下文是否完整，生成思维链并监控通过率以探测难度。代码包括静态文件与 PR 记录。

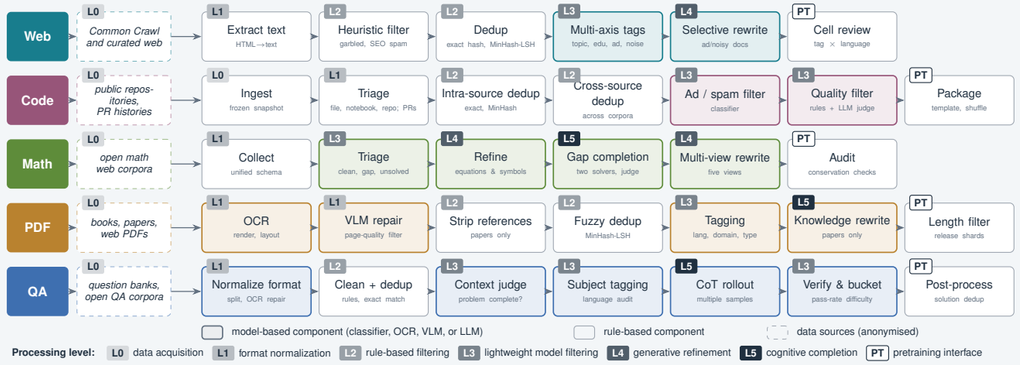


Figure 19: Pretrain data pipelines except the agentic data pipeline. Filled boxes mark model-based components such as classifiers, OCR, VLMs, and LLM rewriters, which we build and tune with the workflow in Figure 14 (Qin et al., 2026a).

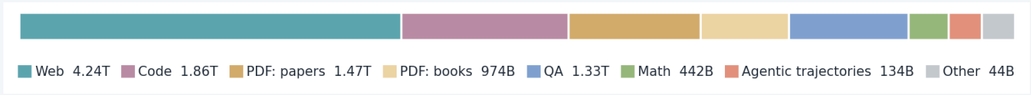
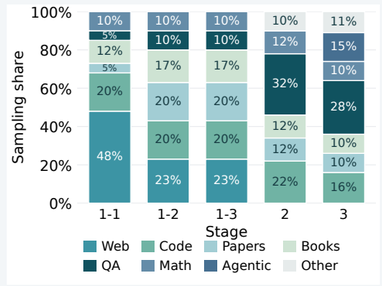


Figure 20: Composition of the pre-training corpus, measured in unique tokens that entered training (10.5T in total).

	Stage1 (11.5T)			Ctx. extension (300B)	
Setup	**1-1**	**1-2**	**1-3**	**Stage2**	**Stage3**
Tokens added	7T	1T	3.5T	100B	200B
Seq. length	4K	4K	4K	32K	256K
RoPE θ	10^4	5×10^5	5×10^6	5×10^6	5×10^6
Learning rate	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4} $\rightarrow 5 \times 10^{-5}$
Warmup steps	2,000	No	No	1,000	1,000
LB loss coef.	10^{-4}	10^{-4}	10^{-4}	Off	Off
Router z-loss	10^{-3}	10^{-3}	10^{-3}	Off	Off

(a) Training stages and key recipe changes.



(b) Configured data sampling proportions.

Figure 21: Training recipe of 801-30B-A3B across the five training stages. *(a)* Key recipe changes, including the progression from 4K to 32K and 256K sequence lengths and the RoPE-base updates during Stage1. *(b)* Configured data sampling proportions. Stage1-2 and Stage1-3 retain the same category-level mixture despite updates to their data sources. Other denotes capability-targeted data.

Figure 21b shows how the configured sampling mixture changes across training stages. These proportions describe sampling weights, whereas Figure 20 reports unique corpus tokens. Stage1-1 assigns 48% to web text and 20% to code. In Stage1-2 and Stage1-3, the web share falls to 23%, while papers, books, and QA increase to 20%, 17%, and 10%, respectively. These two sub-stages share the same category-level proportions, although their underlying data sources differ. Stage2 removes the dedicated web component, raises QA to 32%, and assigns 10% to capability-targeted data. Stage3 introduces a 15% agentic component and assigns 11% to capability-targeted data, including instruction-following and long-context examples. The capability-targeted category also includes retrieved web, book, and paper material, so the absence of a dedicated web component does not imply the absence of all web-derived content.

7.1 Attention-Balanced Data Sampling

Mixing long sequences into training data can cause throughput degradation far beyond what their proportion would suggest. In theory, the expected training time should be the proportion-weighted sum of the time spent on

图 19：除智能体数据之外的预训练流水线。实心框为分类器、OCR、VLM、LLM 改写器等模型组件，使用图 14 的流程构建与调优（Qin 等，2026a）。

网页：CommonCrawl 与精选网页→HTML 提取文本→启发式过滤（乱码、SEO 垃圾）→去重（精确哈希、MinHash-LSH）→多维标签（主题、教育性、广告、噪声）→选择性改写（广告/噪声文档）→标签×语言单元审查。

代码：公开仓库和 PR 历史→获取冻结快照→按文件、notebook、仓库及 PR 分流→源内去重（精确、MinHash）→跨来源去重→广告/垃圾分类过滤→质量过滤（规则+LLM 评判）→模板化、打乱与打包。

数学：开放数学语料及网页语料→统一模式收集→分类（干净、缺失、未解决）→公式符号修整→缺口补全（两个求解器及评判器）→五种视角改写→内容保真审计。

PDF：图书、论文、网页 PDF→OCR（渲染、版面）→VLM 修复及页面质量过滤→去参考文献（仅论文）→MinHash-LSH 模糊去重→语言/领域/类型标签→知识改写（仅论文）→长度过滤并输出分片。

QA：题库与开放 QA→格式规范化（拆分、OCR 修复）→规则清洗和精确去重→上下文完整性判断→学科标注及语言审查→多样本思维链生成→验证、按通过率难度分桶→解答去重等后处理。

层级：L0 数据获取；L1 格式规范化；L2 规则过滤；L3 轻量模型过滤；L4 生成式优化；L5 认知补全；PT 预训练接口。

图 20：进入训练的唯一 token 构成，总计 10.5T。网页 4.24T、代码 1.86T、PDF 论文 1.47T、PDF 图书 974B、QA 1.33T、数学 442B、智能体轨迹 134B、其他 44B。

图 21：五个阶段的训练配方。(a) 关键参数变化，序列长度 4K→32K→256K，Stage1 更新 RoPE 基数。(b) 配置的数据采样比例。Stage1-2 和 1-3 虽更新来源，但大类比例相同。Other 指面向特定能力的数据。

五阶段新增 token 依次为 7T、1T、3.5T、100B、200B；长度为 4K、4K、4K、32K、256K；预热步数为 2,000、无、无、1,000、1,000。Stage1 三段均采用 10^-4 负载均衡系数与 10^-3 路由器 z-loss，后两段关闭。

图 21b 表示采样权重，而图 20 表示唯一语料 token。Stage1-1 网页占 48%，代码占 20%。Stage1-2、1-3 网页降至 23%，论文、图书、QA 分别增至 20%、17%、10%；大类比例相同，但底层来源不同。Stage2 移除独立网页类别，QA 增至 32%，能力定向数据为 10%。Stage3 加入 15% 智能体数据，能力定向数据为 11%，包括指令遵循和长上下文样本。能力定向数据也含检索来的网页、图书与论文，因此没有独立网页类别并不表示完全没有网页来源内容。

7.1 注意力负载均衡的数据采样

混入长序列可能导致远超其占比的吞吐下降。理论上，预期训练时间应是长短样本耗时按比例加权的和；实际中，仅加入 10% 长样本就可能使训练速度减半。

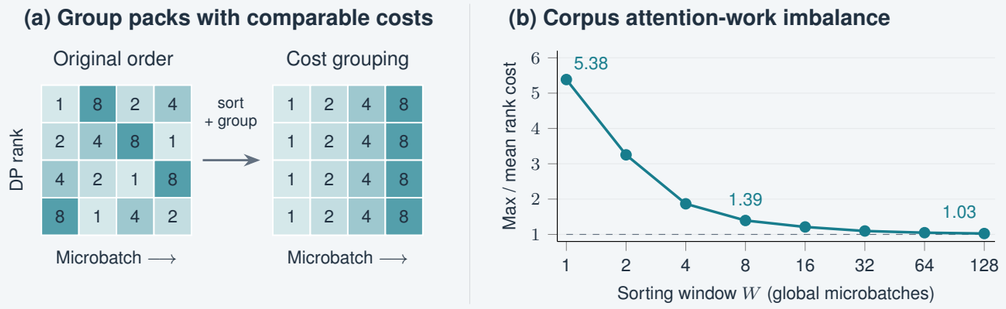


Figure 22: (a) Grouping packs by attention cost reduces imbalance; values are illustrative. (b) Imbalance in our 256K corpus across sorting windows, using 128 DP ranks and one pack per rank. The ratio is maximum to mean rank cost, averaged across microbatches; 1 denotes perfect balance.

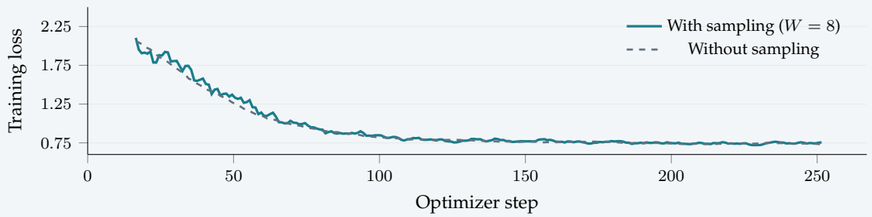


Figure 23: Training loss with and without attention-balanced data sampling (32-step moving average).

long and short samples. In practice, adding only 10% long samples can halve the training speed.

Using the context infrastructure, the agent traces this unexpected slowdown to computation bubbles caused by imbalanced attention workloads across data-parallel (DP) ranks. In synchronous data-parallel training, ranks with heavier attention workloads become stragglers, leaving other ranks idle at synchronization points. This imbalance is particularly severe for packed sequences, where attention cost depends on individual document lengths rather than total token count. We estimate each pack's cost as where m_p counts document segments and $\ell_{p,j}$ is the length of segment j.

$$\widehat{C}(p) = \sum_{j=1}^{m_p} \ell_{p,j}^2, \tag{3}$$

To mitigate this imbalance, the dataloader sort packs by estimated attention cost within a bounded window and group similarly costly packs into global microbatches. When each DP rank receives multiple packs, we further assign packs in descending cost order to the least-loaded rank with remaining capacity (Figure 22a).

The sorting window limits sample reordering. Larger windows improve load balance (Figure 22b), but document length may be associated with data source and difficulty. Aggressive sorting can therefore bias the per-step mixture and substantially increase loss fluctuations early in long-context training. We used a limited sorting window with a buffer of roughly twice the distributed world size to balance computational efficiency and sample diversity.

To evaluate this strategy, we compared two 256K-context pretraining runs on 256 GPUs with matching configurations. We evaluate the proposed dataloader on 256 GPUs with context parallelism of 4 (CP=4), yielding 64 data-parallel (DP) replicas. With one pack per DP replica and a sorting window of eight global microbatches, our method achieves a $2.26\times$ throughput improvement (from 1,126 to 2,543 TGS). Smoothed training losses follow similar trajectories (Figure 23). Because this scheduling alters the global sample order, we evaluated final model quality separately, and benchmark differences with and without scheduling remained within run-to-run fluctuations.

7.2 Optimizer Allocation

We use Muon for expert and attention weight matrices, and AdamW for router, embedding, output, normalization, and bias parameters. To examine this allocation alongside the optimizer diagnosis in Table 3, we compare four

图 22：(a) 按注意力计算成本将打包序列分组，可减少不均衡；数值仅示意。(b) 256K 语料在不同排序窗口下的不均衡，使用 128 个 DP rank，每个 rank 一个包。指标为每个微批次最大 rank 成本/平均成本，再跨微批次求平均；1 为完全均衡。横轴是按全局微批次数计的排序窗口 W，示例比值从 5.38 降至 1.39，最终约 1.03。

图 23：有、无注意力均衡采样的训练损失，采用 32 步移动平均；采样窗口 W=8，横轴为优化器步数。

借助上下文基础设施，智能体将意外减速追溯到 DP rank 间注意力负载不均衡引起的计算空泡。同步数据并行中，注意力工作较重的 rank 成为慢节点，其他 rank 在同步点空等。打包序列尤其严重，因为注意力成本取决于单个文档长度，而非总 token 数。每个包的估计成本为：

$\hat{C}(p) = \Sigma(j=1\ldots m_p)\ \ell^2_(p,j)$ (3)

m_p 为文档片段数，ℓ_(p,j) 为第 j 个片段长度。

为减轻失衡，数据加载器在有限窗口中按估计成本排序，将成本接近的包组成全局微批次。每个 DP rank 接收多个包时，再按成本降序，将包分配给尚有容量且负载最轻的 rank（图 22a）。

排序窗口限制样本重排范围。窗口越大，负载越均衡（图 22b），但文档长度可能与来源、难度相关，激进排序会改变每一步的数据混合，并明显增大长上下文训练初期的损失波动。我们采用有限窗口，缓冲区约为分布式 world size 的两倍，兼顾计算效率和样本多样性。

我们在 256 张 GPU 上比较两次其他配置匹配的 256K 上下文预训练。上下文并行为 4（CP=4），因此有 64 个 DP 副本，每副本一个包，排序窗口为八个全局微批次。吞吐从 1,126 提升到 2,543 TGS，提高 2.26 倍。平滑损失轨迹相近（图 23）。由于调度改变全局样本顺序，我们单独评估最终模型质量，基准差异仍在不同训练运行间的波动范围内。

7.2 优化器分配

专家和注意力权重矩阵使用 Muon；路由器、嵌入、输出、归一化和偏置参数使用 AdamW。结合表 3 中的优化器诊断，我们比较了第二次试训中的四种配置（表 5）。

Table 5: Optimizer allocation in the second pre-training pilot. E denotes expert matrices and E-A denotes expert and attention matrices. LBL denotes the load-balancing scope. All Muon variants use RMS matching to AdamW (Liu et al., 2025), momentum 0.95, and five Newton–Schulz iterations; AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$. Weight decay is 0.1. Router parameters use AdamW in every setting.

Setting	Experts	Attention	Router	Embedding/output/norm/bias	LBL
AdamW–Global	AdamW	AdamW	AdamW	AdamW	Global
Muon-E–Global	Muon	AdamW	AdamW	AdamW	Global
Muon-E-A–Global	Muon	Muon	AdamW	AdamW	Global
Muon-E-A–Local	Muon	Muon	AdamW	AdamW	Local

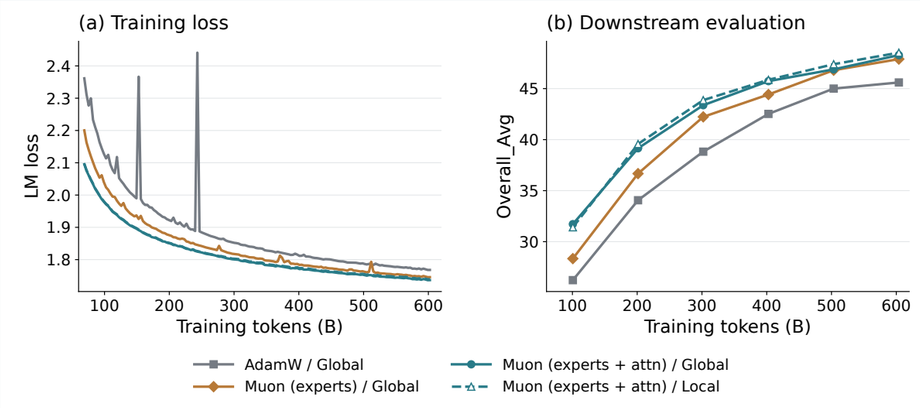


Figure 24: Optimizer allocation and load-balancing comparisons for the four settings in Table 5. *Left*: training LM loss, excluding auxiliary losses, averaged in consecutive 100-step bins after the 2,000-step warmup. *Right*: direct base-model Overall Avg scores at 100B-token intervals. The two expert-and-attention settings share a color, with a dashed line denoting local load balancing. Both global Muon allocations improve on all-AdamW in this interval, while the local/global score differences are small.

configurations from our second pre-training pilot (Table 5). Three use global load balancing and vary the parameters assigned to Muon; a fourth uses the same expert-and-attention allocation with local load balancing. All four use the same configured data mixture, a sequence length of 4,096, a global batch of 8,192 sequences (33.55M tokens), and a learning rate of 5×10^{-4} after 2,000 warmup steps. The load-balancing and router z-loss coefficients are 10^{-4} and 10^{-3}, respectively.

Figure 24 compares these runs over their evaluation interval, ending at step 18,000 (604B tokens). Under global load balancing, both hybrid configurations have higher downstream scores than all-AdamW at every evaluated checkpoint. At the endpoint, `Overall_Avg` is 45.60 for all-AdamW, 47.89 for Muon on experts, and 48.28 for Muon on experts and attention. Their mean LM losses over the last 100 steps are 1.768, 1.746, and 1.737, respectively. Extending Muon to attention provides an improvement in these runs, although the score gap narrows with training. The local and global expert-and-attention variants remain close: their endpoint scores are 48.52 and 48.28. These observations support the hybrid parameter allocation. Downstream evaluation, however, does not distinguish local from global load balancing over this interval. We nonetheless adopt global load balancing for main pre-training because its internal metrics are more stable. Each setting is represented by a single training run.

8 Evaluation

We evaluate 801-30B-A3B through complementary comparisons of base-model performance, general capabilities after supervised fine-tuning (SFT), and agentic capabilities against post-trained models. The base-model comparison assesses the pre-trained checkpoint directly, while the Base+SFT comparison examines general capabilities after adaptation. The agentic comparison further evaluates performance on SWE-bench Verified, SWE-bench Pro, CyberGym, and KernelBench. These settings are presented separately to distinguish direct base-model evaluation

表 5：第二次预训练试训的优化器分配。E 为专家矩阵，E-A 为专家与注意力矩阵，LBL 为负载均衡范围。全部 Muon 变体采用与 AdamW 的 RMS 匹配（Liu 等，2025），动量 0.95，Newton–Schulz 迭代 5 次；AdamW 的 (β₁,β₂)=(0.9,0.95)，权重衰减 0.1。所有配置的路由器均用 AdamW。

AdamW–Global：全部参数用 AdamW，全局均衡。

Muon-E–Global：专家用 Muon，其余用 AdamW，全局均衡。

Muon-E-A–Global：专家和注意力用 Muon，其余用 AdamW，全局均衡。

Muon-E-A–Local：专家和注意力用 Muon，其余用 AdamW，局部均衡。

图 24：四种配置的优化器与负载均衡比较。左图为排除辅助损失的 LM 损失，2,000 步预热后按连续 100 步分箱平均；右图为每 100B token 的基础模型 Overall_Avg。专家加注意力的两种配置同色，虚线表示局部均衡。两种全局 Muon 分配在此区间均优于全 AdamW，而局部/全局分数差异较小。

三种配置采用全局均衡并改变 Muon 的参数分配，第四种采用专家加注意力分配与局部均衡。四者数据混合一致，长度 4,096，全局批次 8,192 条序列（33.55M token），预热 2,000 步后学习率 5×10^-4；负载均衡与路由器 z-loss 系数分别为 10^-4、10^-3。

比较区间止于第 18,000 步（604B token）。全局均衡下，两种混合优化器在全部受评检查点的下游分数均高于全 AdamW。终点 Overall_Avg 分别为：全 AdamW 45.60、仅专家 Muon 47.89、专家加注意力 Muon 48.28；最后 100 步平均 LM 损失依次为 1.768、1.746、1.737。将 Muon 扩到注意力矩阵带来改善，但分数差距随训练缩小。

专家加注意力的局部/全局变体接近，终点分数为 48.52/48.28。这些观察支持混合参数分配；但在此区间内，下游评估无法明确区分局部与全局均衡。我们仍在主训练中选用全局均衡，因为其内部指标更稳定。每种设置仅有一次训练运行。

8 评估

我们从基础模型表现、SFT 后通用能力，以及与后训练模型比较的智能体能力三个互补角度评估 801-30B-A3B。基础比较直接测预训练检查点；Base+SFT 比较适应后的通用能力；智能体比较使用 SWE-bench Verified、SWE-bench Pro、CyberGym、KernelBench。分别呈现这些设置，以区分直接基础评估与适应后的下游表现。

from downstream performance after adaptation.

Given current resource and time constraints, we have focused our initial evaluation and optimization on the settings reported in this section, with particular emphasis on agentic tasks. We plan to broaden task coverage and provide more comprehensive evaluation support in future work.

8.1 Evaluation Settings

8.1.1 Base Evaluation Settings

We evaluate base checkpoints on 13 benchmarks across five categories: commonsense, with ARC-Easy and ARC-Challenge (Clark et al., 2018); reasoning, with MMLU-Pro (Wang et al., 2024b), MMLU-Redux (Gema et al., 2025), BBH (Suzgun et al., 2023), and AGIEval-EN (Zhong et al., 2024); code generation, with HumanEval (Chen et al., 2021), EvalPlus (Liu et al., 2023), and MultiPL-E (Cassano et al., 2023); mathematics, with GSM-8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2024); and science, with GPQA-Main (Rein et al., 2024) and SuperGPQA (M-A-P Team et al., 2025). Table 6 summarizes the default prompting, scoring, and generation settings for each benchmark. The model-specific protocol-selection procedure is described at the end of this subsection.

Prompting and generation. All benchmarks use zero-shot prompting except BBH, which uses three demonstrations, and AGIEval-EN, which uses five. For the non-code benchmarks, we use greedy decoding with temperature 0 and generate one candidate per problem. For HumanEval, EvalPlus, and MultiPL-E, we sample four candidates per problem with temperature 0.2 and top-p 0.95. Each code-generation candidate is limited to 2,048 generated tokens. The other generation limits range from 1,024 to 16,384 tokens, as listed in the table.

Table 6: Default base-evaluation settings (except Gemma-4-26B-A4B). Rows follow the five benchmark categories introduced in the text. T denotes temperature, candidates are counted per problem, and generation limits are in tokens. Greedy decoding uses $T = 0$. MultiPL-E scores are weighted by the number of problems in each language.

Category	Benchmark	Metric	Shots	Decoding	T	Top-p	Candidates	Max. tokens
Commonsense	ARC-Easy	Exact match	0	Greedy			1	16,384
	ARC-Challenge	Exact match	0	Greedy			1	16,384
Reasoning	MMLU-Pro	Exact match	0	Greedy			1	16,384
	MMLU-Redux	Exact match	0	Greedy			1	4,096
	BBH	Exact match	3	Greedy			1	1,024
	AGIEval-EN	Exact match	5	Greedy			1	2,048
Code	HumanEval	Avg@4	0	Sampling	0.2	0.95	4	2,048
	EvalPlus	Avg@4	0	Sampling	0.2	0.95	4	2,048
	MultiPL-E	Avg@4	0	Sampling	0.2	0.95	4	2,048
Math	GSM-8K	Exact match	0	Greedy			1	2,048
	MATH-500	Math Verify	0	Greedy			1	16,384
Science	GPQA-Main	Exact match	0	Greedy			1	16,384
	SuperGPQA	Exact match	0	Greedy			1	4,096

Scoring and aggregation. We use exact match for the non-code benchmarks except MATH-500, whose answers are scored with Math Verify. For HumanEval, EvalPlus, and MultiPL-E, we report Avg@4: test-passing outcomes are averaged over the four generated candidates for each problem and then across problems. Our MultiPL-E evaluation covers ten languages: Python, C++, C#, Go, Java, JavaScript, PHP, Rust, Bash, and TypeScript. We aggregate their scores with weights proportional to the number of evaluated problems in each language.

Code execution. For HumanEval, EvalPlus, and MultiPL-E, we set the compilation and execution timeouts to 15 seconds each. Runtime and compiler versions, along with preparation, compilation, and execution commands, are provided in Appendix D (Table 12).

Model-specific protocol selection for Gemma. For Gemma-4-26B-A4B, limited instruction-following ability makes the results particularly sensitive to prompting and evaluation conventions. We therefore evaluate multiple prompt and protocol configurations and report the maximum score obtained for each benchmark.

受当前资源和时间限制，初步评估与优化集中于本节设置，特别是智能体任务。未来将扩大任务覆盖，提供更全面的评估支持。

8.1 评估设置

8.1.1 基础评估设置

在五类 13 项基准上评估基础检查点：常识类 ARC-Easy、ARC-Challenge（Clark 等，2018）；推理类 MMLU-Pro（Wang 等，2024b）、MMLU-Redux（Gema 等，2025）、BBH（Suzgun 等，2023）、AGIEval-EN（Zhong 等，2024）；代码类 HumanEval（Chen 等，2021）、EvalPlus（Liu 等，2023）、MultiPL-E（Cassano 等，2023）；数学类 GSM8K（Cobbe 等，2021）、MATH-500（Lightman 等，2024）；科学类 GPQA-Main（Rein 等，2024）、SuperGPQA（M-A-P Team 等，2025）。表 6 列默认提示、评分和生成设置，模型专属协议选择见本小节末尾。

提示与生成。除 BBH 用 3 个示例、AGIEval-EN 用 5 个示例外，均为零样本。非代码基准采用温度 0 的贪婪解码，每题一个候选。三项代码基准温度 0.2、top-p 0.95，每题采样四个候选，各最多 2,048 个输出 token。其他生成上限为 1,024—16,384，详见表 6。

表 6：默认基础评估设置（Gemma-4-26B-A4B 除外）。表头：类别、基准、指标、示例数、解码方式、温度 T、Top-p、每题候选数、最大 token。贪婪解码 T=0。

ARC-Easy、ARC-Challenge：精确匹配，0-shot，贪婪，1 个候选，16,384。

MMLU-Pro：精确匹配，0-shot，贪婪，1 个，16,384。

MMLU-Redux：精确匹配，0-shot，贪婪，1 个，4,096。

BBH：精确匹配，3-shot，贪婪，1 个，1,024。

AGIEval-EN：精确匹配，5-shot，贪婪，1 个，2,048。

HumanEval、EvalPlus、MultiPL-E：Avg@4，0-shot，采样，T=0.2，top-p=0.95，4 个，2,048。

GSM8K：精确匹配，0-shot，贪婪，1 个，2,048。

MATH-500：MathVerify，0-shot，贪婪，1 个，16,384。

GPQA-Main：精确匹配，0-shot，贪婪，1 个，16,384。

SuperGPQA：精确匹配，0-shot，贪婪，1 个，4,096。

评分与汇总。除 MATH-500 用 MathVerify 外，非代码均精确匹配。代码报告 Avg@4：每题四个候选的测试通过结果先平均，再跨题平均。MultiPL-E 覆盖 Python、C++、C#、Go、Java、JavaScript、PHP、Rust、Bash、TypeScript 十种语言，按各语言受评题数加权。

代码执行。三项代码评估的编译与执行超时分别为 15 秒；运行时、编译器版本及准备、编译、执行命令见附录 D 表 12。

Gemma 的专属协议选择。Gemma-4-26B-A4B 的指令遵循能力有限，结果对提示和评估惯例尤其敏感，因此评估多个提示/协议配置，并对每个基准报告最高分。

8.1.2 Post-training Evaluation

General Capabilities. We evaluate general capabilities on MMLU (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024b), GPQA Diamond (Rein et al., 2024), AIME 2026 and HMMT February 2026 (Balunović et al., 2025), GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), LiveCodeBench v5 and v6 (Jain et al., 2025), MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), IFBench (Pyatkin et al., 2025), IFEval (Zhou et al., 2023), AA-LCR v1.1 (Artificial Analysis Team, 2025), SimpleQA Verified (Haas et al., 2025), the public set of AA-Omniscience (Jackson et al., 2025), CL-bench (Dou et al., 2026b), and CL-bench Life (Dou et al., 2026a).

We use a sampling temperature of 0.6 throughout. The maximum output length is 262,144 tokens for AIME 2026, HMMT February 2026, LiveCodeBench v5 and v6, CL-bench, and CL-bench Life; 131,072 tokens for GPQA Diamond; 32,768 tokens for MMLU, MMLU-Pro, GSM8K, MATH-500, MBPP, HumanEval, IFBench, and IFEval; and 4,096 tokens for SimpleQA Verified and AA-Omniscience. For CL-bench and CL-bench Life, top-p is set to 0.95. We sample 32 responses per problem for AIME 2026 and HMMT February 2026; eight for LiveCodeBench v5 and v6, GPQA Diamond, MBPP, and HumanEval; four for IFBench, IFEval, AA-LCR v1.1, CL-bench, and CL-bench Life; and one for the remaining benchmarks.

We report accuracy for all benchmarks, averaged over the sampled responses. Mathematical answers are graded using Math-Verify, multiple-choice answers are compared against the reference choices, and generated code is evaluated against the benchmark test cases. AA-LCR v1.1 employs GPT-5.6 Luna (medium) as its judge, while SimpleQA Verified and AA-Omniscience rely on Qwen2.5-32B as the judge. For CL-bench and CL-bench Life, we use DeepSeek-V4.1 with thinking enabled and rubric-level majority voting: each response is judged independently three times, and each rubric's verdict is determined by a majority vote. Only final answer content is graded; responses containing only reasoning receive zero credit on every rubric.

Agentic Capabilities. We evaluate agentic capabilities on SWE-bench Verified (Chowdhury et al., 2024), SWE-bench Pro (Deng et al., 2026), CyberGym (Wang et al., 2026), and KernelBench (Ouyang et al., 2025).

For SWE-bench Verified and SWE-bench Pro, we use the SWE-agent harness (Yang et al., 2024), with a sampling temperature of 0.7, a repetition penalty of 1.1, and a maximum of 300 interaction steps. For SWE-bench Verified, we additionally apply anti-hacking safeguards against the shortcut behaviors summarized in Table 4.

For KernelBench, we use a sampling temperature of 0.6, top-p of 0.95, and a repetition penalty of 1.1. Each model turn allows up to 65,536 output tokens, with a maximum of 20 interaction rounds. Context compression is triggered when the context reaches 150K tokens.

For CyberGym, we use the Claude Code harness (Anthropic, 2025), with a sampling temperature of 1.0 and top-p of 0.95. The context window is 262,144 tokens, with automatic compression triggered at 90% utilization. Each task attempt has a two-hour time limit.

8.2 Base Model Evaluation

We first compare 801-30B-A3B with models for which base checkpoints are available: Qwen3.5-35B-A3B (Qwen Team, 2026a), Qwen3-30B-A3B (Yang et al., 2025a), Kimi-Linear-48B-A3B (Kimi Team et al., 2025), Nemotron-3-Nano-30B-A3B (NVIDIA et al., 2025), Nemotron-3.5-Lightning-30B-A3B (NVIDIA et al., 2026), and Gemma-4-26B-A4B (Gemma Team et al., 2026). Table 7 reports results on the 13 benchmarks spanning commonsense, reasoning, code generation, mathematics, and science, together with their average score.

Competitive base-model performance. 801-30B-A3B achieves an average score of 73.13, the second-highest among the seven base models, behind Qwen3.5 at 76.67 and close to Nemotron-3.5-Lightning at 73.07. It exceeds Qwen3, Nemotron-3-Nano, and Kimi-Linear by 6.97, 4.64, and 8.76 percentage points, respectively. This result establishes a competitive starting point before the SFT evaluation considered below. 801-30B-A3B obtains the highest scores on GSM-8K and HumanEval, reaching 92.80% and 78.81%, respectively. On MMLU-Pro, it scores 68.38%, compared with 67.01% for Qwen3.5 and 69.13% for Nemotron-3.5-Lightning. The base-model results demonstrate a competitive overall profile and distinct task-level strengths of our model.

8.3 SFT Model Results

8.3.1 General Capabilities

We then examine whether the strength of the base model carries over after adaptation. Every model here is fine-tuned on the same SFT data, which is deliberately lightweight: 6.33 B tokens and no reinforcement learning (Section 4.1). Differences among the resulting models therefore mainly reflect what each base model brings to adaptation.

8.1.2 后训练评估

通用能力。评估 MMLU（Hendrycks 等，2021）、MMLU-Pro（Wang 等，2024b）、GPQA Diamond（Rein 等，2024）、AIME 2026 与 HMMT February 2026（Balunović 等，2025）、GSM8K（Cobbe 等，2021）、MATH-500（Lightman 等，2024）、LiveCodeBench v5/v6（Jain 等，2025）、MBPP（Austin 等，2021）、HumanEval（Chen 等，2021）、IFBench（Pyatkin 等，2025）、IFEval（Zhou 等，2023）、AA-LCR v1.1（Artificial Analysis Team，2025）、SimpleQA Verified（Haas 等，2025）、AA-Omniscience 公开集（Jackson 等，2025）、CL-bench（Dou 等，2026b）、CL-bench Life（Dou 等，2026a）。

采样温度均为 0.6。最大输出：AIME、HMMT、LCB v5/v6、CL-bench 和 CL-bench Life 为 262,144 token；GPQA Diamond 为 131,072；MMLU、MMLU-Pro、GSM8K、MATH-500、MBPP、HumanEval、IFBench、IFEval 为 32,768；SimpleQA Verified 和 AA-Omniscience 为 4,096。两项 CL-bench 的 top-p=0.95。

每题响应数：AIME 和 HMMT 为 32；LCB v5/v6、GPQA Diamond、MBPP、HumanEval 为 8；IFBench、IFEval、AA-LCR v1.1、CL-bench、CL-bench Life 为 4；其他为 1。

各基准报告对采样响应取平均的准确率。数学用 Math-Verify，选择题与参考选项比较，代码运行基准测试。AA-LCR v1.1 用 GPT-5.6 Luna（medium）评判；SimpleQA Verified、AA-Omniscience 用 Qwen2.5-32B；两项 CL-bench 用开启思考的 DeepSeek-V4.1，每个响应独立评判三次、逐评分条目多数投票。只评分最终答案；只有推理内容的响应在所有条目上得零分。

智能体能力。在 SWE-bench Verified（Chowdhury 等，2024）、SWE-bench Pro（Deng 等，2026）、CyberGym（Wang 等，2026）、KernelBench（Ouyang 等，2025）上评估。

两项 SWE-bench 用 SWE-agent（Yang 等，2024），温度 0.7，重复惩罚 1.1，最多交互 300 步；Verified 另针对表 4 捷径采取反作弊措施。

KernelBench：温度 0.6，top-p 0.95，重复惩罚 1.1，每轮最多输出 65,536 token，最多 20 轮；上下文达 150K token 时压缩。

CyberGym：Claude Code（Anthropic，2025），温度 1.0，top-p 0.95，上下文 262,144 token，利用率达 90% 自动压缩；每次任务尝试限时两小时。

8.2 基础模型评估

比较有基础检查点的模型：Qwen3.5-35B-A3B、Qwen3-30B-A3B、Kimi-Linear-48B-A3B、Nemotron-3-Nano-30B-A3B、Nemotron-3.5-Lightning-30B-A3B、Gemma-4-26B-A4B。表 7 报告五类 13 项基准及平均分。

有竞争力的基础表现。801 平均 73.13，在七个模型中排第 2，仅次于 Qwen3.5 的 76.67，接近 Nemotron-3.5-Lightning 的 73.07；比 Qwen3、Nemotron-3-Nano、Kimi-Linear 分别高 6.97、4.64、8.76 个百分点。这构成 SFT 前有竞争力的起点。GSM8K 与 HumanEval 获最高分 92.80%、78.81%；MMLU-Pro 为 68.38%，对比 Qwen3.5 的 67.01% 和 Nemotron-3.5-Lightning 的 69.13%。结果显示整体竞争力与不同任务上的优势。

8.3 SFT 模型结果 | 8.3.1 通用能力

继续检验基础优势是否延续到适应后。所有模型使用相同、刻意保持轻量的 SFT 数据：6.33B token，无 RL（第 4.1 节）。结果差异主要反映各基础模型为适应带来的能力。

Table 7: Base-model evaluation. All entries are base checkpoints. Scores are percentages; AVG is the unweighted mean of the 13 benchmark scores. Bold indicates the highest score among the models shown. *Gemma-4-26B-A4B reports the maximum score across multiple prompts and evaluation protocols for each benchmark, as described in Section 8.1.1.

Category	Benchmark	801-30B-A3B Base	Qwen3.5 35B-A3B Base	Qwen3 30B-A3B Base	Kimi-Linear 48B-A3B Base	Nemotron 3-Nano 30B-A3B Base	Nemotron 3.5-Lightning 30B-A3B Base	Gemma-4 26B-A4B*
Commonsense	ARC-Easy	97.69	**98.27**	90.40	93.43	93.86	98.06	96.46
	ARC-Challenge	92.75	**97.01**	83.70	85.75	84.98	94.54	89.93
Reasoning	MMLU-Pro	68.38	67.01	45.15	49.56	59.26	**69.13**	43.55
	MMLU-Redux	83.14	**86.14**	62.26	73.44	76.95	83.74	73.30
	BBH	81.23	**86.35**	80.33	76.07	78.50	77.59	72.62
	AGIEval-EN	58.98	**66.56**	62.59	60.49	58.59	58.93	49.30
Code	HumanEval	**78.81**	71.80	71.95	68.14	64.79	69.97	63.11
	EvalPlus	75.46	**83.08**	79.42	75.15	73.02	80.79	53.20
	MultiPL-E	57.74	**66.48**	61.04	59.63	56.04	62.16	18.18
Math	GSM-8K	**92.80**	87.72	82.79	88.86	90.14	91.36	68.46
	MATH-500	73.20	**83.00**	76.60	62.40	76.00	74.00	36.20
Science	GPQA-Main	51.34	**55.58**	33.26	25.22	42.19	51.12	35.27
	SuperGPQA	39.16	**47.68**	30.60	18.63	36.06	38.57	24.90
AVG		73.13	76.67	66.16	64.37	68.49	73.07	55.73

Table 8 reports two groups of benchmarks. The lower block contains the *classic general benchmarks*, a suite of 13 widely used benchmarks covering the general capabilities commonly evaluated in models of this class, summarized by AVG13. The upper block contains five *specialized-capability evaluations*, covering three specific areas: long-context reasoning (AA-LCR v1.1), factual knowledge (AA-OmniScience on its public set and SimpleQA-verified), and learning from context (CL-bench and CL-bench Life). Their mean is reported as AVG5.

Performance on specialized-capability evaluations. 801-30B-A3B achieves an AVG5 of 28.70, leading Qwen3.5 (20.54) by 8.16 points and ranking first on four of the five evaluations. Its strongest results are on long-context reasoning and factual knowledge, particularly AA-LCR v1.1 (61.00%), AA-OmniScience (31.30%), and SimpleQA-verified (34.50%). The pre-training corpus was decontaminated against SimpleQA using 8-gram overlap matching, making test-set leakage an unlikely explanation for this result. The exception is CL-bench Life, where 801-30B-A3B trails Qwen3.5 and Nemotron-3-Nano.

Performance on classic general benchmarks. 801-30B-A3B achieves an AVG13 of 81.34, ranking second behind Qwen3.5-35B-A3B (83.76), consistent with its ranking in base-model evaluation (Table 7). It performs particularly well on mathematics and code, matching the highest Math500 score (98.40%) and reaching 95.75% on GSM8K and 95.12% on HumanEval. Its main weaknesses are in competitive programming and instruction following, as reflected by its results on LiveCodeBench v5 and IFEval Strict.

8.3.2 Agentic Capabilities

We further compare 801-30B-A3B, after the same lightweight SFT, with post-trained models released by their providers, which have generally gone through a full post-training process, on SWE-bench Verified, SWE-bench Pro, CyberGym, and KernelBench. In addition to general-purpose models of a similar size, namely Qwen3-30B-A3B-Thinking-2507 (Qwen Team, 2025), Qwen3.5-35B-A3B (Qwen Team, 2026a), Qwen3.6-35B-A3B (Qwen Team, 2026b), K2-Horizon-MoVA-36B-A4B (IFM Team, 2026), Gemma-4-26B-A4B-it (Gemma Team et al., 2026), and Nemotron-3.5-Lightning-30B-A3B (NVIDIA et al., 2026), the comparison includes Flint-40B-A2.5B (Recursive, 2026), a coding-specialized MoE model. Table 9 summarizes these results. Unlike the Base+SFT comparison above, this setting compares our model after 6.33B tokens of tiny SFT with released models that have gone through complete post-training pipelines, so we report it as a reference point for post-training headroom rather than as a like-for-like comparison.

801-30B-A3B achieves the highest score on KernelBench (77.60%) and the highest average across the four benchmarks (58.43). It also achieves competitive results on SWE-bench Pro and CyberGym. Notably, 801-30B-A3B

表 7：基础模型评估。所有列均为基础检查点，分数单位为百分比；AVG 为 13 项基准的非加权平均，粗体为各行最佳。Gemma* 按第 8.1.1 节逐基准报告多个提示/协议的最大值。原始逐项数据完整保留在左页。

类别依次为常识、推理、代码、数学、科学。模型平均分：801-30B-A3B 73.13（第 2）；Qwen3.5-35B-A3B 76.67（第 1）；Qwen3-30B-A3B 66.16；Kimi-Linear-48B-A3B 64.37；Nemotron-3-Nano-30B-A3B 68.49；Nemotron-3.5-Lightning-30B-A3B 73.07（第 3）；Gemma-4-26B-A4B 55.73。

表 8 分为两组。下半部为 13 项常用经典通用基准，以 AVG13 汇总；上半部为五项专项能力评估，覆盖长上下文推理（AA-LCR v1.1）、事实知识（AA-OmniScience 公开集、SimpleQA-verified）、从上下文学习（CL-bench、CL-bench Life），均值为 AVG5。

专项能力表现。801 的 AVG5 为 28.70，比 Qwen3.5 的 20.54 高 8.16 分，五项中四项第 1。优势主要在长上下文推理与事实知识，尤其 AA-LCR v1.1 为 61.00%、AA-OmniScience 为 31.30%、SimpleQA-verified 为 34.50%。预训练语料已依据 SimpleQA 进行 8-gram 重叠去污染，因此测试集泄露不太可能解释该结果。例外是 CL-bench Life，801 落后于 Qwen3.5 与 Nemotron-3-Nano。

经典通用基准表现。801 的 AVG13 为 81.34，排名第 2，次于 Qwen3.5-35B-A3B 的 83.76，与基础评估排名一致（表 7）。数学、代码尤其突出：Math500 达 98.40%，并列最高；GSM8K 95.75%，HumanEval 95.12%。主要弱项是竞赛编程和指令遵循，体现在 LiveCodeBench v5 与 IFEval Strict。

8.3.2 智能体能力

在相同轻量 SFT 后，将 801 与厂商发布、通常经过完整后训练的模型比较，基准为 SWE-bench Verified、SWE-bench Pro、CyberGym、KernelBench。相近规模通用模型包括 Qwen3-30B-A3B-Thinking-2507、Qwen3.5-35B-A3B、Qwen3.6-35B-A3B、K2-Horizon-MoVA-36B-A4B、Gemma-4-26B-A4B-it、Nemotron-3.5-Lightning-30B-A3B；另含代码专用 MoE Flint-40B-A2.5B。结果见表 9。

不同于此前 Base+SFT 同条件比较，这里用仅经过 6.33B token Tiny-SFT 的本模型，对比经历完整后训练流程的发布模型。因此它是后训练潜力的参照，不是同条件比较。

801 获得最高 KernelBench 分数 77.60%，以及四项最高平均分 58.43。SWE-bench Pro 与 CyberGym 结果也具有竞争力。

Table 8: General capabilities of Base+SFT models(%). All base models are fine-tuned on the same 6B-token SFT data without reinforcement learning. Scores are percentages; higher is better. AVG5 and AVG13 are the unweighted means of the upper and lower blocks, respectively. Bold indicates the highest score among the models shown, including ties.

Benchmark	801-30B-A3B Base+SFT	Qwen3.5 35B-A3B Base+SFT	Nemotron 3-Nano 30B-A3B Base+SFT	Qwen3 30B-A3B Base+SFT	Nemotron 3.5-Lightning 30B-A3B Base+SFT	Gemma-4 26B-A4B Base+SFT
Specialized-capability Evaluations						
AA-LCR v1.1	**61.00**	52.00	32.00	19.75	18.50	18.25
AA-OmniScience (public set)	**31.30**	21.70	16.70	18.80	17.80	19.00
SimpleQA-verified	**34.50**	12.00	7.00	9.10	9.90	10.20
CL-bench	**13.22**	12.80	7.74	6.95	5.00	4.42
CLbench-life	3.46	**4.20**	3.95	2.22	1.48	1.73
AVG5	**28.70**	20.54	13.48	11.36	10.54	10.72
Classic General Benchmarks						
AIME 2026	86.77	**87.60**	78.65	82.29	81.25	45.73
HMMT Feb 2026	68.37	**71.02**	60.04	65.62	64.02	31.06
LiveCodeBench v5	69.53	**77.37**	72.00	65.73	70.83	48.07
LiveCodeBench v6	66.41	**71.95**	68.19	62.02	64.22	41.41
GPQA Diamond	74.75	**75.76**	66.54	65.97	70.45	53.28
MMLU-Pro	80.00	**82.13**	76.58	77.40	79.22	72.62
IFBench	63.33	**64.92**	59.42	58.08	58.17	45.17
IFEval Strict	85.54	**89.33**	87.34	86.74	86.69	68.95
MMLU	84.62	**88.75**	83.34	84.01	86.60	82.87
GSM8K	95.75	**95.98**	94.62	95.60	**95.98**	93.78
Math500	**98.40**	**98.40**	96.80	96.40	97.20	84.80
MBPP	88.83	**90.30**	87.44	86.25	89.75	72.55
HumanEval	95.12	**95.35**	93.67	91.39	94.05	84.98
AVG13	81.34	83.76	78.82	78.27	79.88	63.48

Table 9: Agentic benchmark results (%). Bold indicates the highest score among the models shown; AVG is the unweighted mean of the four benchmark scores. All baselines are provider-released post-trained models; 801-30B-A3B is evaluated after the lightweight SFT stage only.

Model	SWE-bench Verified	SWE-bench Pro	CyberGym	KernelBench	AVG
Qwen3-30B-A3B-Thinking-2507	20.30	7.01	1.19	5.60	8.53
Qwen3.5-35B-A3B	69.10	39.23	13.47	17.20	34.75
Qwen3.6-35B-A3B	71.65	45.28	24.22	37.20	44.59
K2-Horizon-MoVA-36B-A4B	64.15	44.19	**39.35**	33.60	45.32
Gemma-4-26B-A4B-it	57.15	34.57	8.49	44.00	36.05
Nemotron-3.5-Lightning-30B-A3B	55.35	32.49	8.29	20.40	29.13
Flint-40B-A2.5B	**72.80**	**53.01**	33.91	41.20	50.23
801-30B-A3B Base+SFT	68.95	48.29	38.90	**77.60**	**58.43**

is evaluated after only a lightweight SFT stage using 6.33B tokens, without reinforcement learning, whereas the compared models have generally undergone much more extensive post-training. These results demonstrate the strong agentic capabilities that 801-30B-A3B already exhibits after lightweight post-training.

8.4 Learning aptitude

Lower start, faster learning on BFCL At the first checkpoint (step 55), 801-30B-A3B scores 31.32 on BFCL Full, below the 40.04 of Qwen3-30B-A3B. Its slope, however, is 8.62 points per tenfold increase in SFT steps against 3.33, about 2.6× steeper. From step 697 onward it leads at every shared checkpoint and ends at 48.65 against 46.26 at step

表 8：Base+SFT 模型通用能力（%）。所有基础模型在相同约 6B token SFT 数据上微调，不使用 RL。越高越好，AVG5/AVG13 分别为上、下半部的非加权均值，粗体表示最高分（含并列）。完整逐项数值保留在左页。

列顺序：801、Qwen3.5、Nemotron-3-Nano、Qwen3、Nemotron-3.5-Lightning、Gemma-4，均为 Base+SFT。

专项 AVG5 依次为 28.70、20.54、13.48、11.36、10.54、10.72。

经典通用 AVG13 依次为 81.34、83.76、78.82、78.27、79.88、63.48。

表 9：智能体基准结果（%）。粗体表示最高，AVG 为四项非加权均值。全部基线均为厂商发布的后训练模型；801 只经过轻量 SFT。以下按“SWE-bench Verified / SWE-bench Pro / CyberGym / KernelBench / AVG”排列：

Qwen3-30B-A3B-Thinking-2507：20.30 / 7.01 / 1.19 / 5.60 / 8.53。

Qwen3.5-35B-A3B：69.10 / 39.23 / 13.47 / 17.20 / 34.75。

Qwen3.6-35B-A3B：71.65 / 45.28 / 24.22 / 37.20 / 44.59。

K2-Horizon-MoVA-36B-A4B：64.15 / 44.19 / 39.35 / 33.60 / 45.32。

Gemma-4-26B-A4B-it：57.15 / 34.57 / 8.49 / 44.00 / 36.05。

Nemotron-3.5-Lightning-30B-A3B：55.35 / 32.49 / 8.29 / 20.40 / 29.13。

Flint-40B-A2.5B：72.80 / 53.01 / 33.91 / 41.20 / 50.23。

801-30B-A3B Base+SFT：68.95 / 48.29 / 38.90 / 77.60 / 58.43。

值得强调，801 仅用 6.33B token 轻量 SFT，不含强化学习；对比模型一般经历了远更广泛的后训练。结果说明，801 在轻量后训练后已展现出较强智能体能力。

8.4 学习能力

BFCL 上起点较低、学习更快。首个检查点（第 55 步），801 在 BFCL Full 上为 31.32，低于 Qwen3-30B-A3B 的 40.04；但其斜率为每十倍 SFT 步数提升 8.62 分，对方为 3.33，约陡 2.6 倍。从第 697 步开始，801 在每个共同检查点均领先，最终第 3,485 步达到 48.65，对方为 46.26。

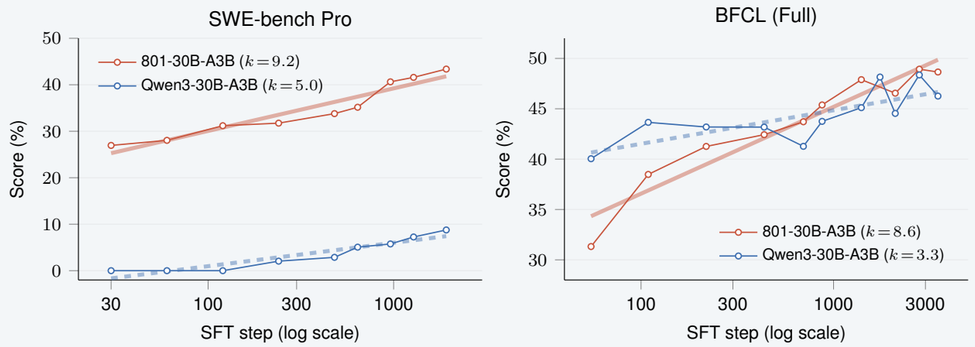


Figure 25: Learning aptitude of 801-30B-A3B and Qwen3-30B-A3B on SWE-bench Pro (left) and BFCL Full (right). Hollow markers are scores at successive SFT checkpoints; the pale line is the least-squares fit of Eq. (1), and k in the legend is its slope, the gain in points per tenfold increase in SFT steps.

3,485. This is the case that motivates learning aptitude: a checkpoint that starts lower but learns faster overtakes one that starts higher once fine-tuning runs long enough.

Higher start and faster learning on SWE-bench Pro On SWE-bench Pro, 801-30B-A3B leads from the first checkpoint, 26.95 against 0 at step 30, and learns faster, with a slope of 9.15 against 5.03. Qwen3-30B-A3B stays at zero for the first three checkpoints (up to step 120) and reaches 8.76 at step 1,922, where 801-30B-A3B reaches 43.37. Here the intercept and the slope point the same way, so the gap widens rather than closes as fine-tuning continues.

9 Conclusion

We attempted to answer a question that current RSI successes leave open: what can agents do for frontier pre-training, where the *propose* → *run* → *measure* → *update* loop cannot close because the run is not repeatable and the objective is not a score? Our answer, drawn from building 801-30B-A3B in three months, is to organize improvement around three types of loops. Debugging loops comprise the infra debugging loop and the data debugging loop, which connect unusual internal metrics and output patterns to their causes. Five cross-stack diagnoses, resolved in about one month, came out of these investigations. The headroom loop is instantiated as the recipe loop, using tiny-SFT, learning aptitude, and max-over-protocols evaluation to guide recipe revisions and monitor training progress. Experience-to-metric loops comprise the classifier development loop, the site discovery loop, and the crawl-coverage loop, turning experience-based judgment into feedback that agents can use to improve data workflows. All of them share a context infrastructure that integrates evidence across teams at scale and preserves negative results and their reasons.

The resulting model is evidence that this regime works at scale. 801-30B-A3B is state-of-the-art among open base models at the 30B MoE scale in both direct base evaluation and matched-SFT comparison, and after a 6B-token SFT stage it approaches a fully post-trained frontier of its size class on agentic tasks. But we hold the model to be the byproduct and the account to be the contribution. Our experience suggests that RSI need not stop at frontier pre-training. It takes a different form: loops eliminate anomalies, headroom measurements guide training, and quantitative feedback improves data workflows. Shared context allows evidence and lessons to accumulate across teams and iterations. There may be no second run, but the one we have can still learn from its own progress.

图 25：801-30B-A3B 与 Qwen3-30B-A3B 在 SWE-bench Pro（左）和 BFCL Full（右）上的学习能力。空心点为连续 SFT 检查点分数，浅色直线为公式 (1) 的最小二乘拟合；图例 k 是斜率，表示 SFT 步数每增加十倍的分数增益。SWE-bench Pro 的 k 约为 9.2/5.0；BFCL Full 约为 8.6/3.3。横轴为对数尺度 SFT 步数，纵轴为分数（%）。

（接上页）这正是引入学习能力概念的原因：起点低但学习快的检查点，在微调足够久后，会超过起点高的模型。

SWE-bench Pro 上起点更高、学习也更快。第 30 步首个检查点，801 为 26.95，对方为 0；斜率 9.15 对 5.03。Qwen3 在前三个检查点（直到第 120 步）一直为零，第 1,922 步达到 8.76，而 801 达到 43.37。此时截距和斜率方向一致，随着微调继续，差距扩大而非缩小。

9 结论

现有 RSI 成功实践留下一个问题：前沿预训练不可重复、目标又不是一个分数，“提出→执行→测量→更新”无法闭合，智能体还能发挥什么作用？从三个月构建 801 的实践中，我们得出的回答是围绕三类循环组织改进。

调试循环包括基础设施调试与数据调试，将内部指标和输出模式的异常连接到根因，约一个月完成五项跨系统诊断。潜力循环具体化为配方循环，用 Tiny-SFT、学习能力、跨协议最大值指导配方修订并监控进展。经验量化循环包括分类器开发、站点发现和抓取覆盖，将经验判断变成智能体可用来改进数据工作流的反馈。所有循环共享上下文基础设施，大规模整合跨团队证据，并保留负面结果及原因。

最终模型表明这种范式能够在大规模场景运行。在 30B MoE 规模开放基础模型中，801 在直接基础评估和同条件 SFT 比较中达到领先水平；经过 6B token SFT 后，其智能体任务表现接近同规模完整后训练模型的前沿水平。

但我们认为，模型是副产品，实践记录才是贡献。经验说明 RSI 不必止步于前沿预训练，而是以不同形式出现：循环消除异常，潜力测量指导训练，定量反馈改进数据工作流。共享上下文让证据与经验跨团队、跨迭代累积。也许没有第二次完整训练，但我们拥有的这一次，依然能够从自身进展中学习。

Author Contributions

The authors are listed in alphabetical order by their last names.

Contributors: Xiaojie Cai, Jiayi Chen, Junwei Chen, Yu Chen, Sikai Cheng, Jinquan Dai, Siyuan Feng, Bin Fu, Dayuan Fu, Guosheng Gao, Qipeng Guo, Chenwei He, Zhulin Hu, Yaxing Huang, Shuomeng Jia, Mohan Jiang, Keyang Lei, Hongliang Li, Xuefeng Li, Zhaocheng Li, Jifan Lin, Pengfei Liu, Xinghan Liu, Yixiu Liu, Pengrui Lu, Siyuan Lu, Kai Lv, Tiantian Mi, Yang Nan, Yu Qiao, Yiwei Qin, Xipeng Qiu, Junhao Shi, Runzhou Shi, Jiadi Su, Bohao Tang, Hanning Wang, Keqi Wang, Pan Wang, Ran Wang, Shaojing Wang, Yishuo Wang, Zhixin Wang, Shenyu Wu, Muhang Xie, Tianze Xu, Hang Yan, Lvmanshan Ye, Hongche Yu, Junjie Yu, Lantian Zhang, Ruihang Zhang, Bin Zhao, Chenyang Zhou, Zijin Zhou, Wenhong Zhu, Zemu Zhu, Yumin Zhuang, Haoyang Zou

Affiliations: Shanghai Innovation Institute, Xinxu AI

作者贡献

作者按姓氏字母顺序排列。

贡献者：Xiaojie Cai、Jiayi Chen、Junwei Chen、Yu Chen、Sikai Cheng、Jinquan Dai、Siyuan Feng、Bin Fu、Dayuan Fu、Guosheng Gao、Qipeng Guo、Chenwei He、Zhulin Hu、Yaxing Huang、Shuomeng Jia、Mohan Jiang、Keyang Lei、Hongliang Li、Xuefeng Li、Zhaocheng Li、Jifan Lin、Pengfei Liu、Xinghan Liu、Yixiu Liu、Pengrui Lu、Siyuan Lu、Kai Lv、Tiantian Mi、Yang Nan、Yu Qiao、Yiwei Qin、Xipeng Qiu、Junhao Shi、Runzhou Shi、Jiadi Su、Bohao Tang、Hanning Wang、Keqi Wang、Pan Wang、Ran Wang、Shaojing Wang、Yishuo Wang、Zhixin Wang、Shenyu Wu、Muhang Xie、Tianze Xu、Hang Yan、Lvmanshan Ye、Hongche Yu、Junjie Yu、Lantian Zhang、Ruihang Zhang、Bin Zhao、Chenyang Zhou、Zijin Zhou、Wenhong Zhu、Zemu Zhu、Yumin Zhuang、Haoyang Zou。

机构：Shanghai Innovation Institute（上海创智学院）、XinxuAI。

人名与机构英文名保留原文，以便准确检索。

References

[1] Wasi Uddin Ahmad, Somshubra Majumdar, et al. 2025. Opencodereasoning-ii: A simple test time scaling approach via self-critique.

[2] Anomaly. 2025. OpenCode: The open source AI coding agent. https://github.com/anomalyco/opencode. Accessed 2026-10-14.

[3] Anthropic. 2025. Claude Code. https://github.com/anthropics/claude-code. Accessed 2026-10-14.

[4] Anthropic. 2026. Messages API reference. https://docs.claude.com/en/api/messages. Accessed 2026-10-14.

[5] Artificial Analysis Team. 2025. Artificial analysis long context reasoning benchmark (AA-LCR). Hugging Face dataset https://huggingface.co/datasets/ArtificialAnalysis/AA-LCR. Version 1.1 released September 2026; methodology at https://artificialanalysis.ai/methodology/intelligence-benchmarking.

[6] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models.

[7] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. 2025. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*. ArXiv:2505.20411.

[8] Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. MathArena: Evaluating LLMs on uncontaminated math competitions. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

[9] Tamay Besiroglu, Ege Erdil, Matthew Barnett, and Josh You. 2024. Chinchilla scaling: A replication attempt. *arXiv preprint arXiv:2404.10102*.

[10] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about physical commonsense in natural language. *arXiv preprint arXiv:1911.11641*.

[11] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, et al. 2023. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691.

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[13] Mingguang Chen, Licheng Wang, and Bo Qu. 2026. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops.

[14] Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, et al. 2024. Introducing SWE-bench Verified. OpenAI Blog. Updated February 24, 2025.

[15] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

[16] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

[17] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. 2026. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*. ArXiv:2509.16941.

[18] Shihan Dou, Yujiong Shen, Chenhao Huang, Junjie Ye, Jiayi Chen, Junzhe Wang, Qianyu He, Shichun Liu, Changze Lv, Jiahang Lin, et al. 2026a. CL-bench Life: Can language models learn from real-life context?

[19] Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. 2026b. CL-bench: A benchmark for context learning.

[20] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378.

参考文献

本页及后续五页保留原始参考文献条目，包括作者、原始题名、年份、出版信息、链接和访问日期，便于检索与核对。

正文中的作者—年份引用与原始参考文献对应；此处不对书目信息进行改写。

[21] Earendil Works. 2025. Pi: A minimal, extensible agent harness. https://github.com/earendil-works/pi. Accessed 2026-10-14.

[22] Factory AI. 2025. Droid: Factory's agent-native development platform. https://github.com/factory-ai/factory. Accessed 2026-10-14.

[23] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

[24] Dayuan Fu, Mohan Jiang, Tongyu Wang, Dian Yang, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. 2026a. davinci-kernel: Co-evolving skill selection, summarization, and utilization via RL for GPU kernel optimization. *arXiv preprint arXiv:2606.16497*.

[25] Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. 2026b. daVinci-Env: Open SWE Environment Synthesis at Scale. *CoRR*, abs/2603.13023.

[26] Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. 2025. Are we done with MMLU? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5069–5096. Association for Computational Linguistics.

[27] Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. 2026. Gemma 4 technical report.

[28] Taicheng Guo, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2026. Autollmresearch: Training research agents for automating LLM experiment configuration – learning from cheap, optimizing expensive. *arXiv preprint arXiv:2605.11518*.

[29] Lukas Haas, Gal Yona, Giovanni D'Antonio, Sasha Goldshtein, and Dipanjan Das. 2025. SimpleQA Verified: A reliable factuality benchmark to measure parametric knowledge.

[30] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*.

[31] Yue Huang, Yanbo Wang, Lichi Li, Yuanyuan Chen, Andreas Haupt, Xiaonan Luo, Zheyuan Liu, Jihye Choi, et al. 2026. RSI Arena: Towards recursive self-improvement with human feedback. *arXiv preprint arXiv:2610.33624*.

[32] IFM Team. 2026. Introducing K2 Horizon: Frontier performance, radically open. Official blog, https://ifm.ai/blog/k2/. Model card: https://huggingface.co/IFM/K2-Horizon-MoVA-36B-A4B.

[33] Declan Jackson, William Keating, George Cameron, and Micah Hill-Smith. 2025. AA-Omniscience: Evaluating cross-domain knowledge reliability in large language models. Public set: https://huggingface.co/datasets/ArtificialAnalysis/AA-Omniscience-Public.

[34] jacobkahn. 2025. Repo state loopholes during agentic evaluation. GitHub issue #465, SWE-bench/SWE-bench. Opened 2025-09-03 by GitHub user jacobkahn; marked fixed for SWE-bench Verified on 2026-02-03.

[35] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

[36] Mohan Jiang, Dayuan Fu, Junhao Shi, Ji Zeng, Weiye Si, Keyu Li, Xuefeng Li, Yang Xiao, Wenjie Li, Dequan Wang, and Pengfei Liu. 2026. daVinci-Agency: Unlocking Long-Horizon Agency Data-Efficiently. *CoRR*, abs/2602.02619.

[37] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations (ICLR)*. ArXiv:2310.06770.

[38] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. 2024. Muon: An optimizer for hidden layers in neural networks. Blog post, https://kellerjordan.github.io/posts/muon/. December 8, 2024; reference implementation at https://github.com/KellerJordan/Muon.

[39] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

[40] Andrej Karpathy. 2026. autoresearch: AI agents running research on single-GPU nanochat training automatically. https://github.com/karpathy/autoresearch.

参考文献（续）

原始作者、题名、年份、出版信息和链接见左页。

[41] Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. 2025. Kimi Linear: An expressive, efficient attention architecture.

[42] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*.

[43] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[44] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. 2025. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*.

[45] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

[46] M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, et al. 2025. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*. ArXiv:2502.14739.

[47] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[48] Maxim AI. 2025. Bifrost: AI gateway. https://github.com/maximhq/bifrost. Accessed 2026-10-14.

[49] Yan Meng, Dhruv Srikanth, Bingchen Zhao, Zhengyao Jiang, and Yuxiang Wu. 2026. Autodata: Agentic search for pre-training data selection. *arXiv preprint arXiv:2609.19754*.

[50] Tiantian Mi, Dongming Shan, Zhen Huang, Yiwei Qin, Muhang Xie, Yuxuan Qiao, Yixiu Liu, Chenyang Zhou, and Pengfei Liu. 2026. Data Darwinism Part II: DataEvolve – AI can Autonomously Evolve Pretraining Data Curation. *CoRR*, abs/2603.14420.

[51] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

[52] NeoHorse Team, Guoliang Cao, Guohao Dai, Tianyu Guo, Kai Han, Hailin Hu, Zihan Jiang, Xiang Kuang, Boxun Li, Yulong Li, et al. 2026. NeoHorse-1: Towards recursive self-improvement via agentic post-training with routing harness.

[53] Nous Research. 2025. Hermes Agent: The self-improving AI agent. https://github.com/NousResearch/hermes-agent. Accessed 2026-10-14.

[54] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, et al. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery.

[55] NVIDIA, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, et al. 2025. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for agentic reasoning.

[56] NVIDIA, Aastha Jhunjhunwala, Aayushi Ashokkumar Srivastava, Abhibha Gupta, Abhinav Garg, Abhinav Khattar, Achsah Wildish, Adam Rajfer, Adi Renduchintala, Adil Asif, Adriana Flores Miranda, et al. 2026. Nemotron 3.5 Super and Lightning: Open, efficient mixture-of-experts hybrid Mamba-Transformer models for agentic AI. Model card: https://huggingface.co/nvidia/NVIDIA-Nemotron-3.5-Lightning-30B-A3B-Base-BF16.

[57] OpenAI. 2025. Codex CLI. https://github.com/openai/codex. Accessed 2026-10-14.

[58] OpenAI. 2026a. Chat completions API reference. https://platform.openai.com/docs/api-reference/chat. Accessed 2026-10-14.

[59] OpenAI. 2026b. Responses API reference. https://platform.openai.com/docs/api-reference/responses. Accessed 2026-10-14.

[60] OpenClaw contributors. 2026. OpenClaw: Your assistant, on your devices, in your chats. https://github.com/openclaw/openclaw. Accessed 2026-10-14.

[61] Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. 2025. KernelBench: Can LLMs write efficient GPU kernels? In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 47356–47415. PMLR. ArXiv:2502.10517.

参考文献（续）

原始作者、题名、年份、出版信息和链接见左页。

[62] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 47717–47737. PMLR. ArXiv:2412.21139.

[63] Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. 2025. Generalizing verifiable instruction following. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

[64] Yiwei Qin, Zhen Huang, Tiantian Mi, Weiye Si, Chenyang Zhou, Qipeng Guo, Siyuan Feng, and Pengfei Liu. 2026a. Data Darwinism Part I: Unlocking the Value of Scientific Data for Pre-training. *CoRR*, abs/2602.07824.

[65] Yiwei Qin, Yixiu Liu, Tiantian Mi, Muhang Xie, Zhen Huang, Weiye Si, Pengrui Lu, Siyuan Feng, Xia Wu, Liming Liu, Ye Luo, Jinlong Hou, Qipeng Guo, Yu Qiao, and Pengfei Liu. 2026b. daVinci-LLM: Towards the Science of Pretraining. *CoRR*, abs/2603.27164.

[66] Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5005–5018, Vienna, Austria. Association for Computational Linguistics.

[67] Qwen Team. 2025. Qwen3-30B-A3B-Thinking-2507. Hugging Face model card, https://huggingface.co/Qwen/Qwen3-30B-A3B-Thinking-2507.

[68] Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents. Official blog, https://qwen.ai/blog?id=qwen3.5. Model cards: https://huggingface.co/Qwen/Qwen3.5-35B-A3B, https://huggingface.co/Qwen/Qwen3.5-35B-A3B-Base.

[69] Qwen Team. 2026b. Qwen3.6-35B-A3B: Agentic coding power, now open to all. Official blog, https://qwen.ai/blog?id=qwen3.6-35b-a3b. Model card: https://huggingface.co/Qwen/Qwen3.6-35B-A3B.

[70] Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. 2026. Posttrainbench: Can LLM agents automate LLM post-training? *arXiv preprint arXiv:2603.08640*.

[71] Recursive. 2026. Introducing Flint. Official blog, https://recursive.com/updates/introducing-recursive-flint. Model card: https://huggingface.co/RecursiveSI/flint.

[72] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling (COLM)*.

[73] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An adversarial Winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

[74] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

[75] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*.

[76] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. AI models collapse when trained on recursively generated data. *Nature*, 631:755–759.

[77] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced Transformer with rotary position embedding. *Neurocomputing*, 568:127063.

[78] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051. Association for Computational Linguistics.

[79] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. 2024a. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*.

[80] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2025. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*. ArXiv:2407.16741. Code: https://github.com/OpenHands/OpenHands.

参考文献（续）

原始作者、题名、年份、出版信息和链接见左页。

[81] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024b. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems 37 (Datasets and Benchmarks Track)*.

[82] Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. 2026. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *The Fourteenth International Conference on Learning Representations (ICLR)*. ArXiv:2506.02548.

[83] Lilian Weng. 2026. Scaling laws, carefully. Lil'Log blog post, https://lilianweng.github.io/posts/2026-06-24-scaling-laws/. Accessed 2026-10-14.

[84] Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, et al. 2026. RRSI: Regularized recursive self-improvement of agent harnesses.

[85] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. 2024. Effective long-context scaling of foundation models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4643–4663, Mexico City, Mexico. Association for Computational Linguistics.

[86] Mingyu Xu, Cheng Fang, Keyue Jiang, Yuqian Zheng, Yanghua Xiao, Baojian Zhou, Qifang Zhao, Suhang Zheng, Xiuwen Zhu, Jiyang Tang, Yongchi Zhao, Yijia Luo, Zhiqi Bai, Yuchi Xu, Wenbo Su, Wei Wang, Bing Zhao, Lin Qu, and Xiaoxiao Xu. 2026a. Logics-stem: Empowering LLM reasoning via failure-driven post-training and document knowledge enhancement.

[87] Weixian Xu, Yixiu Liu, Yang Nan, Lyumanshan Ye, Xiangkun Hu, Zhen Qin, and Pengfei Liu. 2026b. Neural Architecture Discovery via Autonomous Evolution.

[88] Weixian Xu, Tiantian Mi, Yixiu Liu, Yang Nan, Zhimeng Zhou, Lyumanshan Ye, Lin Zhang, Yu Qiao, and Pengfei Liu. 2026c. ASI-Evolve: AI Accelerates AI. *CoRR*, abs/2603.29640.

[89] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

[90] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652. ArXiv:2405.15793. Code: https://github.com/SWE-agent/SWE-agent.

[91] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025b. SWE-smith: Scaling data for software engineering agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*. ArXiv:2504.21798.

[92] Yang Yu, Peiyu Zang, Chi Hsu Tsai, Haiming Wu, Yixin Shen, Jialing Zhang, Haoyu Wang, Zhiyou Xiao, Jingze Shi, Yuyu Luo, et al. 2026. Towards automated kernel generation in the era of LLMs.

[93] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? In *Advances in Neural Information Processing Systems (NeurIPS)*.

[94] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

[95] Ji Zeng, Dayuan Fu, Tiantian Mi, Yumin Zhuang, Yaxing Huang, Xuefeng Li, Lyumanshan Ye, Muhang Xie, Qishuo Hua, Zhen Huang, et al. 2026. davinci-dev: Agent-native mid-training for software engineering. *arXiv preprint arXiv:2601.18418*.

[96] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. 2025. Absolute zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[97] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2024. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2299–2314. Association for Computational Linguistics.

[98] Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. 2025. ImpossibleBench: Measuring LLMs' propensity of exploiting test cases. *arXiv preprint arXiv:2510.20270*.

[99] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models.

参考文献（续）

原始作者、题名、年份、出版信息和链接见左页。

[100] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*.

参考文献（续）

本页为第 100 条参考文献：Zoph 等（2022），ST-MoE: Designing stable and transferable sparse expert models，arXiv:2202.08906。中文题意为“ST-MoE：设计稳定且可迁移的稀疏专家模型”。完整书目信息见左页。

A Layer-wise MaxVio Monitoring

Figure 26 reports the MaxVio trajectories for all 48 Mixture-of-Experts layers over training. Most layers remain within a relatively stable range, while a small number develop substantially larger violations; in particular, layer 15 exhibits the pronounced rise discussed in Figure 7. The vertical dashed lines indicate the stage boundaries, and the horizontal axis is training tokens in trillions.

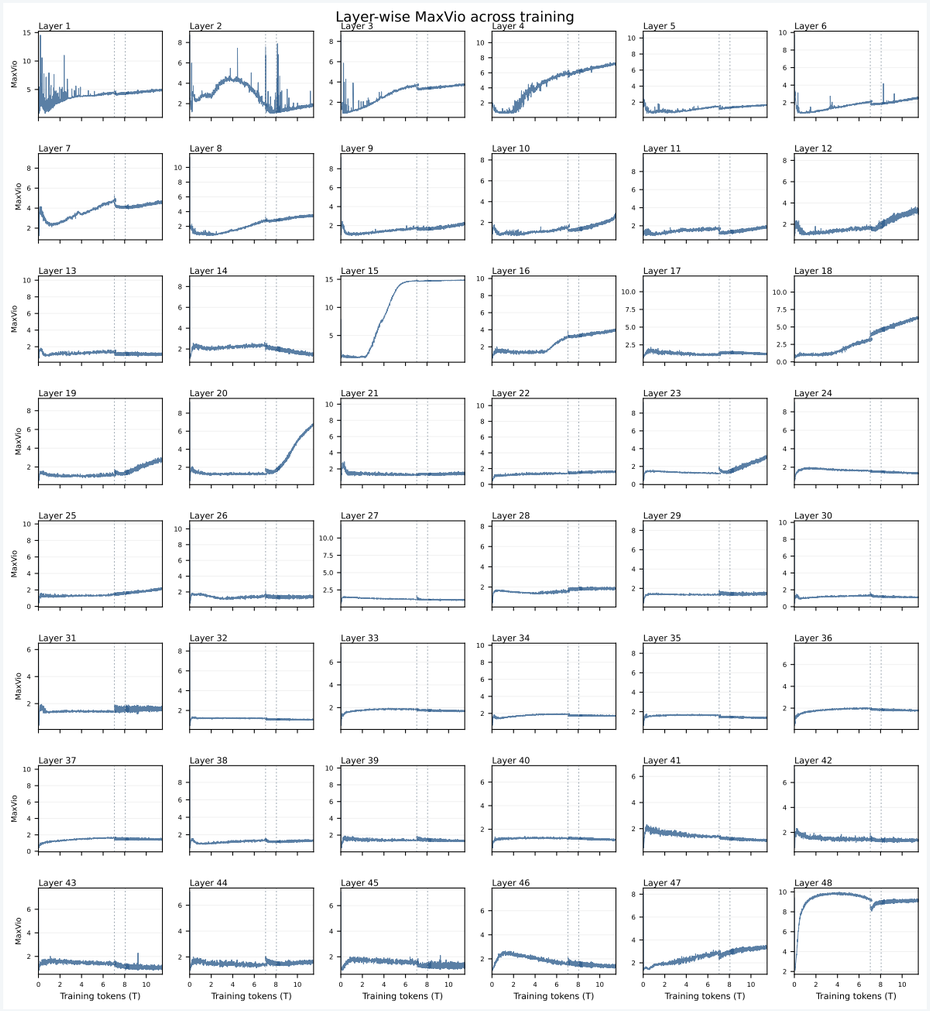


Figure 26: Layer-wise MaxVio trajectories for all 48 MoE layers.

附录 A：逐层 MaxVio 监测

图 26 报告训练过程中全部 48 个 MoE 层的 MaxVio 轨迹。多数层保持在相对稳定范围内，少数层则出现显著较大的违背值，尤其是图 7 已讨论的第 15 层明显上升。

竖直虚线表示训练阶段边界，横轴是以万亿为单位的训练 token 数。

图 26：全部 48 个 MoE 层的逐层 MaxVio 轨迹。

图内 Layer 1—Layer 48 对应第 1—48 层；Training tokens (T) 为训练 token 数（万亿）。MaxVio 越高表示负载越不均衡。

B Benchmarks for Pre-training Monitoring

Table 10 summarizes the 22 evaluation items used to monitor pre-training, grouped as in the training-progress plots. Overall_Avg is the unweighted mean of their 22 reported percentage scores.

Table 10: The 22-benchmark pre-training monitoring suite, with brief introductions to the evaluated tasks.

Benchmark	Description
General commonsense (7)	
ARC-Easy (Clark et al., 2018)	School-level science questions, easy split.
ARC-Challenge (Clark et al., 2018)	More challenging science questions requiring reasoning.
HellaSwag (Zellers et al., 2019)	Selecting plausible continuations of everyday events.
OpenBookQA (Mihaylov et al., 2018)	Combining basic science facts with commonsense.
PIQA (Bisk et al., 2019)	Physical commonsense and practical problem solving.
WinoGrande (Sakaguchi et al., 2019)	Resolving ambiguous references using commonsense.
TriviaQA (Joshi et al., 2017)	Factual short-answer questions without evidence context.
General reasoning (6)	
MMLU (Hendrycks et al., 2021)	Multiple-choice knowledge questions across 57 subjects.
MMLU-Pro (Wang et al., 2024b)	Harder, reasoning-oriented questions across 14 subjects.
MMLU-Redux (Gema et al., 2025)	Human-reviewed MMLU questions.
BBH (Suzgun et al., 2023)	Diverse challenging reasoning tasks; 27 configured subtasks.
AGIEval-EN (Zhong et al., 2024)	Ten English exam subsets covering logic, reading, and math.
DROP (Dua et al., 2019)	Reading comprehension with discrete reasoning.
Code (3)	
HumanEval (Chen et al., 2021)	Implementing Python functions from signatures and natural-language specifications to assess programming and problem-solving ability.
EvalPlus (Liu et al., 2023)	Code-generation evaluation with augmented tests, emphasizing functional correctness and edge-case handling.
MBPP (Austin et al., 2021)	Writing short Python programs from natural-language requirements.
Math (2)	
GSM-8K (Cobbe et al., 2021)	Grade-school word problems requiring multi-step arithmetic.
MATH-500 (Lightman et al., 2024)	Competition-style mathematics.
Science (4)	
GPQA-Main (Rein et al., 2024)	Graduate-level biology, physics, and chemistry questions.
SuperGPQA (M-A-P Team et al., 2025)	Graduate-level knowledge and reasoning across diverse disciplines, including science, engineering, medicine, law, economics, and the humanities.
MMLU-STEM	The 19 MMLU subjects: abstract algebra, anatomy, astronomy, college biology, college chemistry, college computer science, college mathematics, college physics, computer security, conceptual physics, electrical engineering, elementary mathematics, high school biology, high school chemistry, high school computer science, high school mathematics, high school physics, high school statistics, and machine learning.
MMLU-Pro-STEM	Six MMLU-Pro domains: biology, chemistry, computer science, engineering, mathematics, and physics.

附录 B：预训练监控基准

表 10 汇总用于监控预训练的 22 个评估项目，分组与训练进展图一致。Overall_Avg 是这 22 项百分比分数的非加权均值。

表 10：22 项预训练监控基准及任务简介。

通用常识（7 项）

ARC-Easy：学校层次科学问题，简单子集。

ARC-Challenge：需要推理的更难科学问题。

HellaSwag：为日常事件选择合理后续。

OpenBookQA：结合基础科学事实与常识。

PIQA：物理常识及实际问题求解。

WinoGrande：利用常识消解有歧义的指代。

TriviaQA：不提供证据上下文的事实简答题。

通用推理（6 项）

MMLU：覆盖 57 个学科的知识选择题。

MMLU-Pro：覆盖 14 个学科、更难且更强调推理的问题。

MMLU-Redux：经人工审查的 MMLU 题目。

BBH：多样高难推理任务，本配置包含 27 个子任务。

AGIEval-EN：10 个英语考试子集，涉及逻辑、阅读、数学。

DROP：需要离散推理的阅读理解。

代码（3 项）

HumanEval：依据函数签名和自然语言说明实现 Python 函数，评估编程与问题求解能力。

EvalPlus：采用扩充测试的代码生成评估，强调功能正确性和边界情况处理。

MBPP：根据自然语言需求编写简短 Python 程序。

数学（2 项）

GSM8K：需要多步算术的小学应用题。

MATH-500：竞赛式数学。

科学（4 项）

GPQA-Main：研究生层次的生物、物理和化学题。

SuperGPQA：跨科学、工程、医学、法学、经济、人文等学科的研究生层次知识与推理。

MMLU-STEM：MMLU 的 19 个学科，分别为抽象代数、解剖学、天文学、大学生物、大学化学、大学计算机科学、大学数学、大学物理、计算机安全、概念物理、电气工程、初等数学、高中生物、高中化学、高中计算机科学、高中数学、高中物理、高中统计、机器学习。

MMLU-Pro-STEM：MMLU-Pro 的六个领域，分别为生物、化学、计算机科学、工程、数学和物理。

各基准的作者—年份引用见左侧原表。

C Evaluation Protocols for Base Models

Figure 13 evaluates the same checkpoints under several protocols. A protocol is determined by four choices: perplexity-based (PPL) or generation-based scoring, the number of in-context examples (*k*-shot), the required final-answer format, and the answer-matching method. Table 11 defines each protocol label used in the figure.

Table 11: Evaluation protocols in Figure 13.

Protocol	In-context examples	Final-answer format	Scoring
Non-code tasks			
5-shot	5 solved examples: final answers only for ARC, GPQA-Main, MMLU-Redux, and AGIEval-EN; worked solutions for MMLU-Pro and SuperGPQA	"The answer is (X).", shown by the examples	Extracted letter, exact match
4-shot Minerva	4 worked solutions in Minerva format (MATH-500)	Follows the examples	Math equivalence
8-shot CoT	8 worked solutions (GSM-8K)	"The answer is N."	Strict match, exact match
3-shot CoT	3 official worked solutions per subtask (BBH)	"the answer is . . . "	Extracted answer, exact match
0-shot answer-is	None	Instructed to end with "the answer is X"	Extracted answer, exact match†
0-shot boxed	None	Instructed to put the answer in `\boxed{}`	Extracted answer, exact match†
PPL acc_norm	None	No generation	Highest length-normalized option log-likelihood
Code tasks			
3-shot completion	3	Continue the bare function signature and docstring	Unit tests, pass@1
k-shot instruction + signature	3 or 0	Instruction prompt; signature pre-filled in the answer	Unit tests, pass@1
k-shot instruction, no signature	3 or 0	Instruction prompt; the model writes the full function	Unit tests, pass@1

†Math equivalence for MATH-500.

附录 C：基础模型的评估协议

图 13 使用多个协议评估同一批检查点。协议由四项选择确定：困惑度（PPL）或生成式评分、上下文示例数（k-shot）、最终答案格式、答案匹配方法。表 11 定义图中各标签。

表 11：图 13 中的评估协议。

非代码任务

5-shot：5 个已解示例。ARC、GPQA-Main、MMLU-Redux、AGIEval-EN 示例只有最终答案；MMLU-Pro、SuperGPQA 提供解题过程。示例展示最终格式"The answer is (X)."；提取选项字母，精确匹配。

4-shot Minerva：4 个 Minerva 格式的完整解答（MATH-500）；答案跟随示例格式；按数学等价评分。

8-shot CoT：8 个完整解答（GSM8K）；最终格式"The answer is N."；严格匹配、精确匹配。

3-shot CoT：每个 BBH 子任务使用 3 个官方完整解答；格式"the answer is ..."；提取答案后精确匹配。

0-shot answer-is：无示例；要求以"the answer is X"结尾；提取答案、精确匹配†。

0-shot boxed：无示例；要求将答案放入 \boxed{}；提取答案、精确匹配†。

PPL acc_norm：无示例，无生成；选择长度归一化对数似然最高的选项。

代码任务

3-shot completion：3 个示例；接续裸函数签名和文档字符串；以单元测试评定 pass@1。

k-shot instruction + signature：3 或 0 个示例；指令式提示，答案预填函数签名；单元测试、pass@1。

k-shot instruction, no signature：3 或 0 个示例；指令式提示，模型自行编写完整函数；单元测试、pass@1。

† MATH-500 使用数学等价判断。

D Code Evaluation Environments

Table 12 provides the code execution environments for the base-model evaluations in Section 8.1.1. HumanEval and EvalPlus use the Python environment, while MultiPL-E uses all ten language environments.

Table 12: Code execution environments. Preparation or compilation commands precede execution. Java uses JVM build `25.0.1-internal-adhoc.conda.src`. `JAVATUPLES_JAR` and `TSC_JS` identify the JavaTuples dependency and the TypeScript compiler entry point, respectively.

Language / runtime	Preparation and execution commands
Python CPython 3.10.19	`exec(check_program, exec_globals)`
C++ GCC 13.3.0 libstdc++ 15.2.0	`g++ -std=c++17 program.cpp -o program` `./program`

C#
Mono JIT / mcs 6.12.0.199
	`mcs -define:DEBUG -r:System.Numerics.dll \`
	`  -out:Program.exe Program.cs`
	`MONO_TRACE_LISTENER=Console.Error mono Program.exe`

Go Go 1.23.4 (linux/amd64)	`go mod init solution` `go test -p=8`
Java OpenJDK / javac 25.0.1-internal	`javac -cp ".:$JAVATUPLES_JAR" Problem.java` `java -ea -cp ".:$JAVATUPLES_JAR" Problem`
JavaScript Node.js 20.12.2	`node program.js`
PHP PHP CLI 8.5.4 Zend Engine 4.5.4	`php program.php`
Rust rustc 1.96.0; LLVM 22.1.2	`rustc --edition 2021 program.rs -o program` `./program`
Bash GNU Bash 5.1.16	`bash program.sh`

TypeScript
TypeScript 5.4.5
Node.js 20.12.2
	`node "$TSC_JS" program.ts --target es2020 \`
	`  --module commonjs --lib es2020,dom --skipLibCheck`
	`node program.js`

附录 D：代码评估环境

表 12 提供第 8.1.1 节基础模型评估的代码执行环境。HumanEval 和 EvalPlus 使用 Python；MultiPL-E 使用全部十种语言。

表 12：代码执行环境。准备或编译命令先于执行。Java 使用 JVM 构建 25.0.1-internal-adhoc.conda.src。JAVATUPLES_JAR 和 TSC_JS 分别表示 JavaTuples 依赖与 TypeScript 编译器入口。代码与命令保留原文：

Python | CPython 3.10.19

exec(check_program, exec_globals)

C++ | GCC 13.3.0；libstdc++ 15.2.0

g++ -std=c++17 program.cpp -o program

./program

C# | Mono JIT/mcs 6.12.0.199

mcs -define:DEBUG -r:System.Numerics.dll \

-out:Program.exe Program.cs

MONO_TRACE_LISTENER=Console.Error mono Program.exe

Go | Go 1.23.4（linux/amd64）

go mod init solution

go test -p=8

Java | OpenJDK/javac 25.0.1-internal

javac -cp ".:$JAVATUPLES_JAR" Problem.java

java -ea -cp ".:$JAVATUPLES_JAR" Problem

JavaScript | Node.js 20.12.2

node program.js

PHP | PHP CLI 8.5.4；Zend Engine 4.5.4

php program.php

Rust | rustc 1.96.0；LLVM 22.1.2

rustc --edition 2021 program.rs -o program

./program

Bash | GNU Bash 5.1.16

bash program.sh

TypeScript | TypeScript 5.4.5；Node.js 20.12.2

node "$TSC_JS" program.ts --target es2020 \

--module commonjs --lib es2020,dom --skipLibCheck

node program.js