# IT TAKES WORKFLOWS TO EVOLVE BETTER WORKFLOWS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Tackling complex real-world tasks can exceed the capabilities of a single large language model (LLM), motivating the use of multi-agent workflows that coordinate specialized agents to work together on these tasks. Recent methods train LLMs to construct better workflows from execution outcomes, but they optimize only the workflow generator, while the other agents that build or execute each workflow remain fixed even though every outcome depends on all of them. However, extending training beyond the generator is challenging: the agents are coupled, and a workflow's outcome is a single sparse score that cannot tell which agent causes a failure. We propose **FLOWRIGHT**, which leverages the workflow as a harness to optimize workflows. By introducing a hierarchical, structure-aware reward paradigm, **FLOWRIGHT** enables one role to self-evolve and two or more roles to co-evolve, with no additional models, labels, or executions. Considering the limitation that workflows are commonly trained and evaluated on data that a single agent can already handle, we further propose **DATAWRIGHT**, an adaptive data hardening approach that converts existing datasets into workflow-level tasks with increased difficulty. Across document, slide, chart, code, math, and finance tasks, small open models trained with **FLOWRIGHT** achieve improved performance by up to $+7.41\%$, with co-evolving ($+5.03\%$) more roles gaining more than optimizing one of them alone ($+2.83\%$).

## 1 INTRODUCTION

Large language models (LLMs) are increasingly capable of leveraging tools and specialized skills to solve tasks (Zhang et al., 2025e; Qian et al., 2025; Wang et al., 2025c; Zhang et al., 2026a). However, tackling complex real-world problems can go beyond the capability of a single LLM, as it demands gathering evidence across heterogeneous inputs, reasoning over long contexts and diverse sources, and accumulating intermediate results over many steps (Yu et al., 2025b; Zhu et al., 2025). To address such tasks, LLMs construct multi-agent workflows that break complex tasks into subtasks and coordinate specialized agents (Zhang et al., 2025d; Hu et al., 2025b; Niu et al., 2025).

Yet constructing an effective workflow for a complex task remains challenging, and many multi-agent frameworks still rely on substantial manual design (Jin et al., 2026; Li et al., 2025; Agashe et al., 2025). To reduce this manual effort, recent works optimize LLMs to construct better workflows by training them against the outcomes of executing those workflows (Nie et al., 2025; Nielsen et al., 2026). However, these methods typically optimize a single workflow generator, while the agents that execute the workflow, and any other agents that help build it, remain fixed (§A.1). Since the outcome of a workflow depends jointly on every agent that builds or executes it, *training only the generator leaves other agents shaping every outcome without ever learning from it, significantly limiting how much the workflow as a whole can improve*.

Extending training beyond the generator introduces two additional challenges. First, agents in a workflow are interdependent: one agent's output can become another's input, and overall performance depends on their joint behavior Existing multi-agent training either trains each agent separately (Guo et al., 2026; Motwani et al., 2025) or co-trains agents with distinct roles only within a fixed pairing or a hand-designed system (Park et al., 2025; Zhao et al., 2026; Huang et al., 2026; Wan et al., 2025) (§A.2). *Training agents separately thus ignores how each agent's behavior shapes what the others learn, while co-training remains confined to a single role or a predefined agent structure rather than*



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

*the workflows constructed for each task*. Second, the outcome of a workflow is usually a single score for the whole workflow. It is sparse whenever the workflow falls short, and cannot tell which agent causes a failure. Methods that do attribute failures to specific agents or steps, however, rely heavily on learned critics or reward models, LLM judges, human annotation, or repeated executions (Liu et al., 2025; Zhang et al., 2025b; Ma et al., 2026) (§A.3). *Such attribution adds costly models, labels, or executions beyond the workflow run itself, and it can still misattribute where a failure originates*.

Beyond how workflows are built and trained, the data used to optimize and evaluate them pose a further challenge. Multi-agent workflows are commonly trained and evaluated on datasets such as single-question math, function-level coding, and short question answering, which a single agent can already handle (Zhang et al., 2026b) (§A.4). On such data, re-evaluations find that the gains of multi-agent systems over a strong single agent are often small or absent (Cemri et al., 2025; Choi et al., 2025; Zhu et al., 2026) (§C). *These datasets can therefore neither reveal nor optimize what a workflow adds beyond a single agent*.

Motivated by these limitations, we propose **FLOWRIGHT** (§3), which leverages the workflow as a harness (§2) to optimize any role (§3.5). Besides letting each role self-evolve, **FLOWRIGHT** co-evolves multiple roles inside the harness (§3.4), and stages the grade into a hierarchical, structure-aware reward (§3.3) with role-level credit localization (§3.2). Beyond workflow construction and optimization, **FLOWRIGHT** introduces a data adaptation algorithm (Alg. 1) that hardens existing datasets for workflow-level training and evaluation (§D). Our contributions are as follows:

1. **Shared harness for every role, at train and test time.** We define a shared workflow harness whose signal serves three purposes (§2.1): the evaluation metric (§D.4), the optimization signal for training any role (Fig. 2), and the meta distillation objective at test time (§3.5).
2. **Structure-aware credit at no extra cost.** We design a reward hierarchy that turns the sparse signal of a workflow into dense, role-level feedback (§3.3), with credit read from the execution trace that the harness already records, requiring no additional models, labels, or executions (§3.2).
3. **Workflow-level data hardening.** We propose **DATAWRIGHT**, a data adaptation approach whose three hardening strategies convert datasets a single agent can already handle into workflow-level tasks with increased difficulty levels, supporting both training and evaluation (§D).
4. **Four evolution modes that generalize.** **FLOWRIGHT** enables four evolution modes (§3.4): (a) single-role self-evolution, (b) agent-skill co-evolution, (c) upstream-downstream co-evolution, and (d) multi-agent co-evolution. Co-evolving more roles gains more than optimizing one alone, achieving $+5.03\%$ overall against $+2.83\%$ for a single role (§5.1). In addition, gains are generalizable across roles, backbones, RL algorithms, workflow topologies, and pools (§5.2).

## 2 PRELIMINARIES

### 2.1 WORKFLOW AS HARNESS

A complex task $x \sim \mathcal{X}$ is solved by a *workflow*, a directed graph $G = (V, E)$ whose nodes $V$ are agents, tools, or skills, and whose edges $E$ carry the data and control flow among them. A *harness* $\mathcal{H}$ grounds a workflow $G$ into verifiable measurement through execution and grading: it runs $G$ on $x$, returns the answer $y$, and scores it as $h(G, x) \in [0, 1]$. A *Generator* $\pi_g$ produces $G$ for $x$:

$$x \xrightarrow{\pi_g} G \xrightarrow{\mathcal{H}} \big(y,\ h(G,x)\big), \quad h(G,x) \in [0,1] \tag{1}$$

where $h$ serves three purposes: the evaluation metric, the training signal, and the test-time objective. A *role* $\rho \in \mathcal{R}$ (Tab. 7) is any component of the upstream-downstream framework (Fig. 1) whose policy $\pi_\rho$, with parameters $\theta_\rho$, is optimized against that signal.

### 2.2 UPSTREAM BUILDS, DOWNSTREAM RUNS

**FLOWRIGHT** factors into an *upstream* and a *downstream*. Upstream, the *Generator* $\pi_g$ produces $G$. As shown in Fig. 1, upstream workflow generation can take on different structures to construct a better $G$. Downstream, the harness $\mathcal{H}$ executes and grades $G$ to return $\big(y,\ h(G,x)\big)$. The workflow is the interface between them: *what the upstream constructs is what the downstream executes* (Eq. 1).

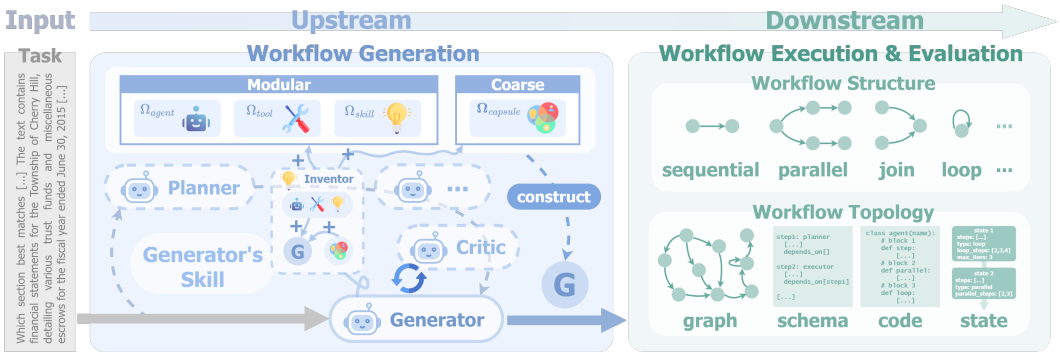


Figure 1: **Upstream-Downstream Framework.** Upstream, workflow generation can take on different structures, from *Generator* with its own *Inventor* skill to *Generator* working with other agents, drawing on pools of different granularity. Downstream, workflows take different structures and topologies to better solve the task.

### 2.3 What the Nodes Are Made of

A workflow is built from a shared *pool* $\Omega$ of reusable agents, tools, and skills. Each node of $G$ is an agent, a tool, or a skill from $\Omega$, so $\Omega$ delimits the space of workflows the *Generator* can produce.

## 3 FloWright: Learning to Wright Workflows

For a task $x_i \in \mathcal{X}$, upstream produces a workflow $G \in \mathcal{G}$ (§2, 3.1). FloWright (Fig. 2) grounds it into a measurable *optimization signal* $h(G, x_i)$ through a principled harness with both local crediting (§3.2) and a global hierarchy (§3.3). By learning to optimize this signal, FloWright enables LLMs to solve complex tasks by building better workflows, through self-evolution and co-evolution (§3.4) at both train and test time (§3.5).

### 3.1 Wire the Workflow: Topology & Pool Dynamics

Upstream construction turns on three choices: the topology of the workflow $G$, and the granularity and dynamics of the pool $\Omega$ it draws from.

**Topology is a non-linear control-flow *graph*.** Beyond passing progress forward in a chain, $\pi_g$ runs nodes in parallel when subtasks are independent, forks alternative branches that are later reconciled, and repeats a step until a stopping condition is met. A single workflow can thus express sequential, parallel, branching, and iterative structure as the task demands.

**Pool is presented to the upstream at a chosen *granularity* (Fig. 1).** A fine, modular granularity keeps agents, tools, and skills as separate units to enable flexible selection and composition: $\Omega = \Omega_{agent} \cup \Omega_{tool} \cup \Omega_{skill}$. A coarse granularity bundles each agent with necessary tools and skills into a self-contained capsule, leading to $\Omega = \Omega_{capsule}$.

**Pool is *dynamic*.** When a task calls for a capability that $\Omega$ does not contain, the pool is expanded with the missing component, a new tool, skill, or agent. The upstream thus grows its own components during construction in addition to drawing directly from the pool (Fig. 1). Holding the pool static disables this growth, a setting we ablate in our experiments (§5.2).

### 3.2 Credit the Structure: Localizing Credits to Roles

For the $j$-th workflow $G_j = (V_j, E_j)$ executed on $x_i$, the optimization signal $h(G_j, x_i)$ is a single scalar, but the action it measures spans an entire graph. By *harnessing the harness*, *i.e.*, leveraging the execution trace $\tau_j$ produced by $\mathcal{H}$, we localize it to the roles responsible. Credit localization therefore requires no additional execution and no new signal.

**Credit charges each role for the failures of its own nodes.** A role $\rho$ authors a subset $V_j^\rho \subseteq V_j$ of the workflow's nodes (Fig. 2), *e.g.*, *Generator* authors every node it designs and wires, *Inventor*

authors only the nodes that use a component it creates, and *Downstream* only the agent nodes it runs. From $\tau_j$ we recover $F_j^\rho \subseteq V_j^\rho$, the role's own nodes that root-cause a failure, and charge the role in proportion to the fraction of its footprint at fault:

$$c_j^\rho = -\frac{|F_j^\rho|}{|V_j^\rho|} \in [-1, 0] \tag{2}$$

If $F_j^\rho = \varnothing$, *e.g.*, when no node fails, then $c_j^\rho = 0$; otherwise the term is negative, steering the policy away from failures (§E.1).

### 3.3 Harness the Workflow: A Hierarchical Reward

The harness does not reduce an execution to a single pass-or-fail outcome. Instead, it stages the optimization signal into a principled hierarchy of auto-verified terms, so the policy sees a meaningful gradient even when the workflow falls short. As such, we propose our hierarchical reward as a generic paradigm: one *shared harness* from which every optimizable role adaptively instantiates its own reward.

**Optimization signal is a staged ladder.** Reward terms accumulate from the role's own output to the task outcome: whether the role's output $o_j^\rho$ *parses* ($f$), whether the role's contribution is *valid* ($v^\rho$), whether the workflow *executes* to a final answer ($e$), and how correct that *answer* is ($a$). The local credit of §3.2 enters as a fifth, structure-aware term, extending the ladder across local and global. As such, the signal for role $\rho$ is:

$$h_j^\rho = h^\rho(G_j, x_i) = w_f\, f + w_v\, v^\rho + w_e\, e + w_a\, a + w_c\, c_j^\rho \tag{3}$$

where $\{w_f, w_v, w_e, w_a, w_c\}$ are weighting coefficients configured per role (§F.2). Our hierarchical ladder thus composes one optimization signal across two axes: *from the role's own output to the task outcome*, the staged four terms measure how far a workflow reaches; *from local to global*, the credit term (§3.2) localizes failures to roles at fault. **FlowWright** aims to maximize this signal at train and test time (§3.4, 3.5).

### 3.4 Harness for Evolution: One Role or Mutual Curriculum

Our reward paradigm serves every role from one shared harness, enabling one role to self-evolve or several to co-evolve.

**Any single role self-evolves inside the harness.** To train a role, we optimize its role-specific reward $h^\rho$ inside the harness as it runs, so the role evolves in the same flow that will use it (§4a). The same procedure fits any role, since all roles share one harness.

**Several roles co-evolve inside the harness through a mutual curriculum.** When several roles train on one substrate inside one harness, each role's behavior shapes what the others encounter. Improving one role therefore reshapes the experience the rest learn from. The co-evolving roles can thus share one policy or each keep their own, and in either case each role learns against the latest evolved policies of the others. For example, co-evolution can pair an agent with its own skill (§4b), couple an upstream agent that builds the workflow with a downstream agent that executes it (§4c), or co-train any two or more agents (§4d). In each case, the coupling is a *mutual curriculum*: each role shapes the training distribution the others learn from. The structure-aware credit of §3.2 charges each role only for the failures of its own nodes in the shared execution, which separates the roles' credit wherever their footprints differ.

### 3.5 Maximize the Harness: Train-Time RL, Test-Time Distillation

**FlowWright** optimizes the signal in two stages: train-time reinforcement and test-time distillation.

**Train time: maximizing the signal.** We optimize a role's policy $\pi_\rho$ to maximize its expected signal over tasks:

$$\mathcal{J}(\pi_\rho) = \mathbb{E}_{x_i \sim \mathcal{X},\, \{\pi_{\rho'}\}_{\rho' \in \mathcal{R}}} \left[\, h^\rho(G_j, x_i) \,\right] \tag{4}$$

where $\mathcal{J}$ optimizes $\pi_\rho$ while the expectation runs over the policies $\pi_{\rho'}$ of all roles $\rho' \in \mathcal{R}$ in the flow. As **FlowWright** contributes a training paradigm rather than an optimizer, $\mathcal{J}$ admits any

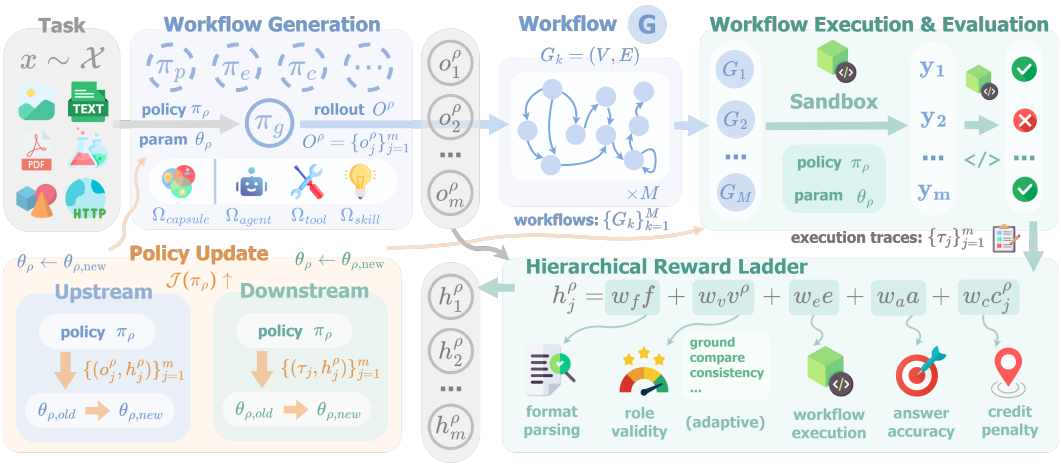


Figure 2: **FLOWRIGHT for reinforcement learning via self-evolving harness.** $\pi_\rho$ is the policy of a role $\rho \in \mathcal{R}$. For the $j$-th of $m$ rollouts of a role being optimized, $G_j \in \mathcal{G}$ is the workflow that rollout is graded on, and the $m$ rollouts span $M$ distinct workflows (Tab. 7). Workflow generation and execution form a coupled on-policy learning loop: upstream proposes workflows, downstream executes them, and the verifiable harness turns execution outcomes into learning signals for the policies being optimized. Policy updates produce the next generation of workflows and executions, enabling self-evolution by harnessing the system's own experience without additional supervision.

policy-gradient method: we anchor on the proximal policy optimization (Schulman et al., 2017) clipped surrogate and leave the advantage estimator and policy loss interchangeable, so recent variants such as GRPO (Shao et al., 2024), DAPO (Yu et al., 2025a), and CISPO (MiniMax et al., 2025) all instantiate it, and we ablate the choice in our experiments (§5.2).

**Test time: meta distillation without gradients.** The same shared harness also optimizes workflows at test time, where no weights change. In place of the policy parameters, FLOWRIGHT optimizes a reusable prior $P$ that optimize generation:

$$P^\star = \arg\max_P \; \mathbb{E}_{x_i \sim \mathcal{X}} \big[\, h(G, x_i \mid P) \,\big] \tag{5}$$

This prior is distilled *internally* from the system's own experiences, or *externally* from a stronger teacher, or both. We additionally distill successful experiences into few-shot cases, from which roles evolve by *meta learning*. We define this *meta distillation*: the harness turns what its own successful experiences teach into the prior that later tasks reuse. As such, FLOWRIGHT keeps improving on new tasks, driven by the very signal it optimizes during training.

## 4 EXPERIMENTS

**Setup.** We evaluate FLOWRIGHT in both objectives it optimizes (§3.5): reinforcement learning at train time, and meta distillation at test time. At train time, we cover four evolution modes (§3.4): (a) single-role self-evolution, (b) agent-skill co-evolution, (c) upstream-downstream co-evolution, and (d) multi-agent co-evolution. We compare them against two single-agent baselines, with or without the tool and skill pools (Fig. 1). Each baseline runs with the same compute budget as our FLOWRIGHT counterpart it is compared against. All evaluations are on the held-out test sets, non-overlapped with training data (§D.5). We train `Qwen3.5-4B` and `Qwen3.5-9B` (Team, 2026) on the *paired* strategy at $\ell = 3$, and evaluate two open-source models, `Qwen3.5-4B` and `Qwen3.5-9B`, together with two close-source models, `GPT-5-mini` (OpenAI, 2025) and `GPT-5.4-mini` (OpenAI, 2026b), on held-out test sets, using `GPT-5.4` (OpenAI, 2026a) as the distillation teacher (§3.5) and `GPT-5.6-luna` (OpenAI, 2026c) as LLM judge (§D.4). As a role can be powered by a different backbone than the rest, we additionally evaluate settings that pair a smaller upstream backbone with a larger downstream one as well as the reverse. We summarize our implementation details in §F.

**Data.** We evaluate on the hardened tasks of DATAWRIGHT (§D), which converts 12 datasets over 7 domains into workflow-level tasks under three hardening strategies (§D.1), at harden-

Table 1: **Evaluation on Four Evolution Modes.** Mean accuracy (%) over the hardening strategies and levels each dataset supports (§D.1). FLOWRIGHT trains on the *paired* strategy at $\ell$=3 of four datasets, which defines three regimes (§D.5): **in-distribution** is those four training arms; **out-of-distribution** is the same four datasets under a strategy, a level, or both that training never sees (13 arms); and **out-of-domain** is eight datasets FLOWRIGHT never trains on (27 arms). In-distribution is $\ell$=3 by construction, since $\ell$=3 is the training level and $\ell$=5 is held out. The green shading is proportional to the gain over untrained backbones. `4B` and `9B` abbreviate `Qwen3.5-4B` and `Qwen3.5-9B`. (a)-(d) denote the four evolution modes defined in §4.

| Method | In-Dist. | Out-of-Distribution | | | Out-of-Domain | | | Overall |
|---|---|---|---|---|---|---|---|---|
| | $\ell$=3 | $\ell$=3 | $\ell$=5 | Mean | $\ell$=3 | $\ell$=5 | Mean | |
| Baseline: Single Agent (*without tool or skill*) | | | | | | | | |
| **Qwen3.5-4B** | 10.87 | 11.39 | 6.12 | 8.15 | 7.89 | 5.12 | 6.56 | 7.42 |
| **Qwen3.5-9B** | 16.17 | 14.45 | 9.35 | 11.31 | 13.55 | 9.53 | 11.61 | 11.94 |
| Baseline: Single Agent (*with tools and skills*) | | | | | | | | |
| **Qwen3.5-4B** | 21.13 | 20.75 | 15.12 | 17.28 | 25.02 | 19.35 | 22.29 | 20.70 |
| **Qwen3.5-9B** | 28.53 | 31.14 | 23.87 | 26.67 | 35.79 | 30.08 | 33.04 | 30.75 |
| Baseline: FLOWRIGHT (*untrained*) | | | | | | | | |
| **Qwen3.5-4B** | 31.91 | 34.91 | 25.70 | 29.24 | 36.17 | 30.58 | 33.48 | 32.09 |
| **Qwen3.5-9B** | 50.14 | 44.47 | 40.48 | 42.02 | 52.44 | 46.84 | 49.74 | 47.50 |
| FLOWRIGHT: Train-Time Optimization | | | | | | | | |
| **+ (a) 4B** | 35.16 | 37.78 | 29.15 | 32.47 | 38.38 | 33.58 | 36.07 | 34.92 |
| **+ (a) 9B** | **54.51** | **48.28** | **43.66** | **45.44** | **56.99** | **50.36** | **53.80** | **51.39** |
| **+ (b) 4B** | 39.32 | 39.50 | 30.94 | 34.23 | 39.27 | 33.74 | 36.61 | 36.15 |
| **+ (c) 4B** | 36.23 | 38.75 | 29.88 | 33.29 | 38.89 | 32.65 | 35.89 | 35.15 |
| **+ (d) 4B** | 39.23 | 40.71 | 31.91 | 35.29 | 40.48 | 34.67 | 37.68 | 37.12 |

ing levels $\ell \in \{3, 5\}$. Each dataset is split from the source before hardening to preclude train-test overlap. Tab. 3 summarizes DATAWRIGHT data statistics. We refer to each $<$ *dataset*, *hardening strategy*, *hardening level* $>$ triple as an evaluation *arm*, yielding 44 arms in total (Tab. 1, §D.5). Each arm falls into one of three evaluation *regimes* by its distance from the arms FLOWRIGHT trains on: *in-distribution*, *out-of-distribution*, and *out-of-domain* (Tab. 1).

**Evaluation Metrics.** A hardened task is scored as the mean over its sub-questions, and a dataset's accuracy is the mean over its test tasks (§D.4). Every sub-question keeps the metric of its own answer type (Tab. 2), so bundling changes how hard a task is without changing how its answers are graded.

## 5 RESULTS

### 5.1 EVOLVE THE ROLES: OPTIMIZE ANY ROLE BY HARNESSING THE WORKFLOW

**FLOWRIGHT improves performance in every regime at train time.** Comparing with single-agent baselines, untrained FLOWRIGHT showcases improved performance ($\geq +11.39\%$ overall). This reveals that models with smaller sizes are able to build and run workflows with gains better than its size acting alone. At matched model size, every evolution mode exceeds its untrained counterpart in every regime and overall by up to $+7.41\%$ (Tab. 1), and every arm improves under at least one mode. Moreover, the gain also increases when more roles co-evolve. Multi-agent co-evolution gains $+7.32\%$ in-distribution, $+6.05\%$ out-of-distribution, and $+4.20\%$ out-of-domain, staying positive across all datasets, strategies, and levels (Fig. 12). Among the four `Qwen3.5-4B` modes, multi-agent co-evolution leads $7/8$ columns of Tab. 1 and gains most overall at $+5.03\%$, ahead of agent-skill co-evolution at $+4.06\%$, upstream-downstream co-evolution at $+3.06\%$ and single-role self-evolution at $+2.83\%$. These improvements indicate that co-evolving more roles yields more than optimizing

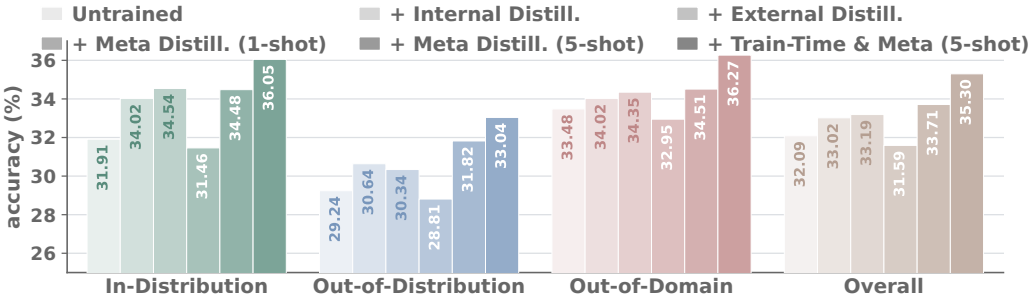


Figure 3: **Test-Time Optimization with Reusable Prior.** The leftmost bar of each group is the untrained `Qwen3.5-4B` workflow baseline, and every other bar generates workflows by distilling a reusable prior (§3.5). *Internal* and *external* distillation draw that prior from own experiences and from a stronger teacher (GPT-5.4) respectively. *Meta* distillation additionally distills successful experiences into few-shot cases, from which roles self-evolve via meta learning (1-shot and 5-shot). The rightmost bar composes train-time evolution with the 5-shot prior.

one of them alone. Harnessing the workflow therefore carries what FLOWRIGHT learns on four datasets at one hardening strategy and one level to the strategies, levels, and domains held out from it.

**FLOWRIGHT also improves performance at test time.** In place of policy parameters, FLOWRIGHT additionally distills a reusable prior at test time (§3.5) for improved adaptability on new tasks. Distilling this prior from its own experiences gains $+0.93\%$ overall, and from a stronger teacher (GPT-5.4) $+1.10\%$ (Fig. 3). Meta distillation gives the largest test-time gain at five shots with up to $+2.58\%$, in contrast to one shot staying below the untrained baseline. *A prior helps only when it carries enough evidence to generalize beyond the single case it comes from.* Test-time and train-time optimization then compound, reaching $+3.21\%$ overall, $+4.14\%$ in-distribution, $+3.80\%$ out-of-distribution, and $+2.79\%$ out-of-domain, exceeding the five-shot prior alone by $+1.59\%$ and leading other settings across regimes. The same harness (§3.5) therefore optimizes workflows whether or not the policy weights update.

**FLOWRIGHT generalizes across roles and backbones at both train and test time.** At train time, optimizing any single role improves the performance, and co-evolving more roles improves it further (Fig. 14). Also, an optimized `4B` role is able to transfer its gain when other roles run on `Qwen3.5-9B`, even raising accuracy above the all-`9B` baseline (Fig. 13). At test time, the meta-distilled prior demonstrates persistent gain in every regime on all four open-source and close-source model backbones (Fig. 15). *One harness therefore optimizes any role and backbone a workflow is built and executed upon*. We extend our discussion in §G.2.

### 5.2 ABLATE THE DESIGN: ALGORITHM, REWARD, POOL, AND TOPOLOGY

**FLOWRIGHT is generalizable to different RL algorithms.** Optimizing *Generator* consistently outperforms the untrained baseline with different RL algorithms (Fig. 4): $+2.83\%$ overall for GRPO, $+3.23\%$ for CISPO, and $+3.84\%$ for DAPO. DAPO leads in every regime, and its margin widens in out-of-domain evaluation, where it gains $+3.69\%$ against $+2.82\%$ for CISPO and $+2.59\%$ for GRPO. On the other hand, for in-distribution, it stays within $0.01\%$ of CISPO. *The harness provides a learning signal generalizable to different RL algorithms.*

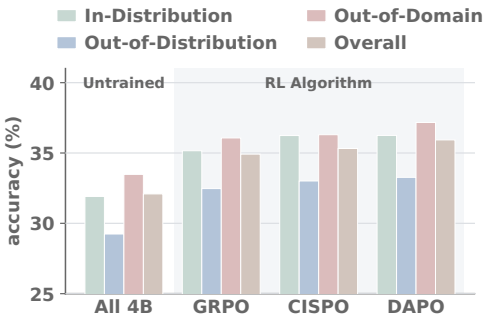


Figure 4: **Algorithm Generalizability.** Each algorithm optimizes *Generator* on `Qwen3.5-4B`.

**Every layer of the reward ladder contributes.** As shown in Fig. 5, grading format and answer alone stays close to the untrained baseline, with $+0.07\%$ under GRPO and $+0.80\%$ under DAPO. Adding validity reward contributes the largest single increment at $+2.07\%$ under GRPO and $+2.29\%$ under DAPO, respectively. The structure-aware credit of §3.2 adds a further $+0.69\%$ and $+0.75\%$, reaching $+2.83\%$ and $+3.84\%$ overall, respectively. *Each layer of the reward ladder contributes, with validity contributing the most.*

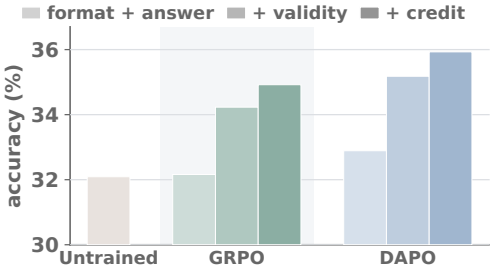


Figure 5: **Reward Ladder.** Each layer adds one term (§3.3): format and answer alone, plus validity (§E.3), plus the structure-aware credit (§3.2).

**FLOWRIGHT is generalizable to different workflow topologies.** As shown in Fig. 6, using *code* to represent workflows reaches $34.12\%$ untrained and $35.86\%$ with an optimized *Generator*, ahead of the other three representations in both. Optimization improves every topology, by $+2.83\%$ for *schema*, $+2.62\%$ for *state machine*, $+1.74\%$ for *code*, and $+1.25\%$ for *graph*, narrowing the spread across the four from $3.91\%$ to $3.03\%$. *Optimization improves across workflow representations, so the gains hold whichever topology the Generator writes in.*

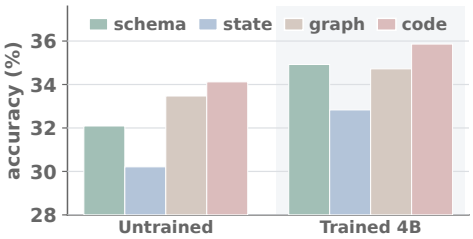


Figure 6: **Workflow Topology.** *Generator* writes workflows in schema, state machine, graph, or code (§E.4). Both use the `Qwen3.5-4B` backbone.

**Capsules lead at every backbone, and optimization narrows the gap.** As shown in Fig. 7, bundling each agent with the tools and skills it needs gains $+3.83\%$ on `Qwen3.5-4B` and $+3.87\%$ on `Qwen3.5-9B` over a modular pool (§3.1). Optimizing *Generator* improves the modular pool by $+4.50\%$ against $+2.83\%$ for capsules, narrowing the gap between the two to $+2.16\%$. *Composing a workflow from separate units demands more fine-grained workflow design from the upstream, and training strengthens upstream agents to better satisfy that demand.*

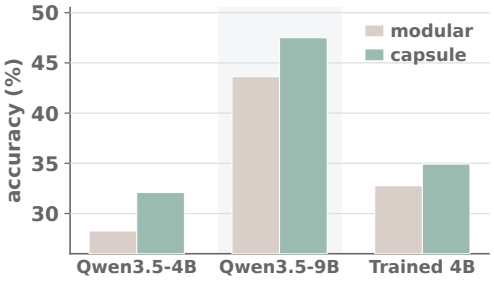


Figure 7: **Pool Granularity.** Backbones are compared at two pool granularities (§3.1), and *Trained 4B* carries RL-trained *Generator*.

**Growing the pool from successful experiences adds the most.** As shown in Fig. 8, a pool that grows on demand gains $+4.83\%$ over a pool held fixed. This also gives the largest effect among all the single design choices in this ablation study. Distilling past experiences into the pool adds a further $+1.44\%$ when only successful ones are kept, against $+0.53\%$ when both successful and failed ones are kept. The performance ranking among the four pool dynamics modes holds before and after optimization, with gains up to $+2.83\%$ from training. *The workflow therefore learns from its own experience, and learns most from the experiences that succeed.*

## 6 RELATED WORK

We summarize four lines of work below and extend each in §A.

**Workflow Construction and Optimization.** Early multi-agent frameworks coordinate agents through manually specified roles and procedures (Hong et al., 2024; Wu et al., 2023). Later methods automate this design while keeping every model fixed, searching for a workflow writ-

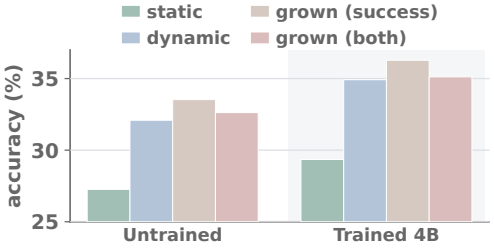


Figure 8: **Pool Dynamics.** A *static* pool stays fixed, a *dynamic* pool expands on demand, and a *grown* pool also carries components distilled from past experiences (§3.1).

ten as code (Zhang et al., 2025d; Hu et al., 2025b) or generating a graph for each task and revising it during execution (Niu et al., 2025), and a further line trains the *generator* itself from execution outcomes (Zhang et al., 2025a; Nie et al., 2025; Nielsen et al., 2026). A separate line improves the workflow without updating any weights, inducing reusable routines from past experience (Wang et al., 2024), evolving the context a model conditions on (Zhang et al., 2026c), or replacing reinforcement learning with reflective prompt evolution (Agrawal et al., 2026). *However, the models that solve the workflow's subtasks remain fixed, limiting how much the workflow as a whole can improve.*

**Multi-Agent Training and Co-Evolution.** Reinforcement learning increasingly optimizes several interacting agents, yet within an interaction structure *fixed in advance*, such as a discussion among agents that play one role (Park et al., 2025) or a workflow designed once per task domain (Zhao et al., 2026). Co-evolution likewise fixes how the models interact, pairing a challenger that proposes problems with a solver that answers them (Huang et al., 2026), or scoring proposals in a multi-round discussion (Xue et al., 2026). Other works train each agent separately, in its own reinforcement learning with a role-specific dataset and reward (Guo et al., 2026; Motwani et al., 2025). *However, agents then learn in isolation or in one fixed role, rather than over the workflows constructed for each task.*

**Credit Assignment and Failure Attribution.** Credit finer than a single outcome typically comes from *learned models*, such as a process reward model trained alongside the policy (Liu et al., 2025) or a local reward for each component, estimated by re-executing what follows it (Wu et al., 2026). Failure attribution instead names the responsible agent and step with LLM judges (Zhang et al., 2025f), an attributor trained on labeled failures (Zhang et al., 2025b), or re-execution with edited messages (Ma et al., 2026), yet identifies the decisive step in only about a third of failed runs (Chen et al., 2026). Credit that adds *no model or rollout* compares actions taken from states that recur across a single agent's rollouts of the same task (Feng et al., 2025; Cheng et al., 2026). *However, such attribution costs models, labels, or executions beyond the workflow run itself, and it can still misattribute where a failure originates.*

**Evaluation of Multi-Agent Workflows.** Methods that construct or optimize workflows are commonly evaluated on single-question math, function-level code, and short question answering (Zhang et al., 2025d;a; Nie et al., 2025). On such data, automatically designed systems improve on a self-consistent single agent by at most a few points (Singh et al., 2025), and majority voting alone accounts for most of the gains attributed to multi-agent debate (Choi et al., 2025). Training on it can also collapse a system into *one dominant agent* (Zhang et al., 2025h), and failure analyses find the reported gains often minimal (Cemri et al., 2025). *However, such data can neither reveal nor optimize what a workflow adds beyond a single agent.*

## 7 Conclusions

In this work, we introduce **FlowWright**, which leverages the workflow as harness (§3.1) to provide the optimization signal without additional annotations: one signal that evaluates a workflow, trains any role that builds or executes it, and evolves a reusable prior at test time (§2.1, 3.5). A hierarchical, structure-aware reward (§3.3) turns the sparse outcome of a workflow into role-level feedback, with role-specific credit derived from the execution trace (§3.2). **FlowWright** enables train-time optimization in four modes: (1) single role self-evolution, (2) agent-skill co-evolution, (3) upstream-downstream co-evolution, and (4) two or more roles co-evolve; and test-time optimization via meta distillation (§3.4). To supply reliable workflow-level evaluation, we further propose **DataWright** to convert single agent level datasets into workflow-level tasks through three hardening strategies at two levels, yielding the 44 arms our experiments span (§D). Across those arms, **FlowWright** outperforms the single-agent baselines by $\geq +11.39\%$, adds a further $+2.58\%$ through meta distillation and $+5.03\%$ through reinforcement optimization (§5.1). These gains hold across RL algorithms, workflow topologies, pool granularities, and pool dynamics (§5.2), and transfer to backbones the harness never trains (§G.2).

### AI USE STATEMENT

In this work, we used generative AI tools to *clean and reformat dataset*. We have **not** used generative AI tools to: *help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, interpret results*. And *generate synthetic data sets, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, assist with translation, support qualitative and thematic data analysis* are not applicable to this work. Additionally, we used generative AI tools for *correct grammar and polish writing*. We have reviewed all AI-assisted work. For example, we have manually go through all the corrected grammar errors to ensure they are not false alarms, and we also manually code and check LLM-assisted quality control during dataset cleaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

This work focuses on optimizing multi-agent workflows at both train time and test time via reinforcement learning and meta distillation, respectively. Our work raises no concerns involving human subjects, personal data, or sensitive attributes.

### REPRODUCIBILITY STATEMENT

In this work, we introduce **FLOWRIGHT** with methodology details in §3. We also show full dataset construction, source selection, and quality control in §D, and every metric used to grade a task is defined in §D.4. We also present our experiment setup, including backbones, baselines, and experimented modes in §4, and further elaborate on our implementation details in §F. We also summarize our experiment configuration details in Tab. 6. Our full code for **FLOWRIGHT**, dataset construction, and evaluation sandbox will all be made public upon acceptance.

## REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025. URL https://arxiv.org/abs/2504.00906.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems, 2026. URL https://arxiv.org/abs/2604.22708.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2605.26684.

Hyeong Kyu Choi, Xiaojin Zhu, and Sharon Li. Debate or vote: Which yields better decisions in multi-agent large language models?, 2025. URL https://arxiv.org/abs/2508.17536.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration, 2025. URL https://arxiv.org/abs/2505.19591.

Chao Deng, Jiale Yuan, Pi Bu, Peijie Wang, Zhong-Zhi Li, Jian Xu, Xiao-Hui Li, Yuan Gao, Jun Song, Bo Zheng, and Cheng-Lin Liu. Longdocurl: a comprehensive multimodal long document benchmark integrating understanding, reasoning, and locating, 2025. URL https://arxiv.org/abs/2412.18424.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training, 2025. URL https://arxiv.org/abs/2505.10978.

Apurva Gandhi, Satyaki Chakraborty, Xiangjun Wang, Aviral Kumar, and Graham Neubig. Recursive agent optimization, 2026. URL https://arxiv.org/abs/2605.06639.

Xuehang Guo, Zhiyong Lu, Tom Hope, and Qingyun Wang. Anagent for enhancing scientific table & figure analysis, 2026. URL https://arxiv.org/abs/2602.10081.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025a. URL https://arxiv.org/abs/2505.23885.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025b. URL https://arxiv.org/abs/2408.08435.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data, 2026. URL https://arxiv.org/abs/2508.05004.

Yiqiao Jin, Rachneet Kaur, Zhen Zeng, Sumitra Ganesh, and Srijan Kumar. Slideagent: Hierarchical agentic framework for multi-page visual document understanding, 2026. URL https://arxiv.org/abs/2510.26615.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks, 2026. URL https://arxiv.org/abs/2601.14652.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022. URL https://arxiv.org/abs/2211.11501.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset, 2023. URL https://arxiv.org/abs/2312.14852.

Zijian Li, Xin Guan, Bo Zhang, Shen Huang, Houquan Zhou, Shaopeng Lai, Ming Yan, Yong Jiang, Pengjun Xie, Fei Huang, Jun Zhang, and Jingren Zhou. Webweaver: Structuring web-scale evidence with dynamic outlines for open-ended deep research, 2025. URL https://arxiv.org/abs/2509.13312.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Junge Zhang, and Jianbin Jiao. Agentic reinforcement learning with implicit step rewards, 2025. URL https://arxiv.org/abs/2509.19199.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems, 2026. URL https://arxiv.org/abs/2512.06749.

Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, Pan Zhang, Liangming Pan, Yu-Gang Jiang, Jiaqi Wang, Yixin Cao, and Aixin Sun. Mmlongbench-doc: Benchmarking long-context document understanding with visualizations, 2024. URL https://arxiv.org/abs/2407.01523.

MiniMax, :, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du, Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui Du, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Enwei Jiao, Haigang Zhou, Haimo Zhang, Han Ding, Haohai Sun, Haoyu Feng, Huaiguang Cai, Haichao Zhu, Jian Sun, Jiaqi Zhuang, Jiaren Cai, Jiayuan Song, Jin Zhu, Jingyang Li, Jinhao Tian, Jinli Liu, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kaiyi Feng, Ke Yang, Kecheng Xiao, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Li, Lin Zheng, Linge Du, Lingyu Yang, Lunbin Zeng, Minghui Yu, Mingliang Tao, Mingyuan Chi, Mozhi Zhang, Mujie Lin, Nan Hu, Nongyu Di, Peng Gao, Pengfei Li, Pengyu Zhao, Qibing Ren, Qidi Xu, Qile Li, Qin Wang, Rong Tian, Ruitao Leng, Shaoxiang Chen, Shaoyu Chen, Shengmin Shi, Shitong Weng, Shuchang Guan, Shuqi Yu, Sichen Li, Songquan Zhu, Tengfei Li, Tianchi Cai, Tianrun Liang, Weiyu Cheng, Weize Kong, Wenkai Li, Xiancai Chen, Xiangjun Song, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xinzhu Hou, Xuan Lu, Xun Zou, Xuyang Shen, Yan Gong, Yan Ma, Yang Wang, Yiqi Shi, Yiran Zhong, Yonghong Duan, Yongxiang Fu, Yongyi Hu, Yu Gao, Yuanxiang Fan, Yufeng Yang, Yuhao Li, Yulin Hu, Yunan Huang, Yunji Li, Yunzhi Xu, Yuxin Mao, Yuxuan Shi, Yuze Wenren, Zehan Li, Zelin Li, Zhanxu Tian, Zhengmao Zhu, Zhenhua Fan, Zhenzhen Wu, Zhichao Xu, Zhihang Yu, Zhiheng Lyu, Zhuo Jiang, Zibo Gao, Zijia Wu, Zijian Song, and Zijun Sun. Minimax-m1: Scaling test-time compute efficiently with lightning attention, 2025. URL https://arxiv.org/abs/2506.13585.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training, 2025. URL https://arxiv.org/abs/2412.01928.

Fan Nie, Lan Feng, Haotian Ye, Weixin Liang, Pan Lu, Huaxiu Yao, Alexandre Alahi, and James Zou. Weak-for-strong: Training weak meta-agent to harness strong executors, 2025. URL https://arxiv.org/abs/2504.04785.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor, 2026. URL https://arxiv.org/abs/2512.04388.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation, 2025. URL https://arxiv.org/abs/2501.07834.

OpenAI. GPT-5 Mini Model, 2025. URL https://developers.openai.com/api/docs/models/gpt-5-mini.

OpenAI. GPT-5.4 Model, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.4.

OpenAI. GPT-5.4 Mini Model, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.4-mini.

OpenAI. GPT-5.6 Luna Model, 2026c. URL https://developers.openai.com/api/docs/models/gpt-5.6-luna.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning, 2025. URL https://arxiv.org/abs/2502.18439.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs, 2025. URL https://arxiv.org/abs/2504.13958.

Runqi Qiao, Qiuna Tan, Guanting Dong, Minhui Wu, Chong Sun, Xiaoshuai Song, Zhuoma GongQue, Shanglin Lei, Zhe Wei, Miaoxuan Zhang, Runfeng Qiao, Yifan Zhang, Xiao Zong, Yida Xu, Muxi

Diao, Zhimin Bao, Chen Li, and Honggang Zhang. We-math: Does your large multimodal model achieve human-like mathematical reasoning?, 2024. URL https://arxiv.org/abs/2407.01284.

Varshini Reddy, Rik Koncel-Kedziorski, Viet Dac Lai, Michael Krumdick, Charles Lovering, and Chris Tanner. Docfinqa: A long-context financial reasoning dataset, 2025. URL https://arxiv.org/abs/2401.06915.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space, 2025. URL https://arxiv.org/abs/2410.06153.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Mayank Singh, Vikas Yadav, Shiva Krishna Reddy Malay, Shravan Nayak, Sai Rajeswar, Sathwik Tejaswi Madhusudhan, and Eduardo Blanco. Grammar search for multi-agent systems, 2025. URL https://arxiv.org/abs/2512.14079.

Namyoung So, Seokgyu Jang, and Taeuk Kim. Superficial success vs. internal breakdown: An empirical study of generalization in adaptive multi-agent systems, 2026. URL https://arxiv.org/abs/2604.18951.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

Ryota Tanaka, Kyosuke Nishida, Kosuke Nishida, Taku Hasegawa, Itsumi Saito, and Kuniko Saito. Slidevqa: A dataset for document visual question answering on multiple images, 2023. URL https://arxiv.org/abs/2301.04883.

Qwen Team. Qwen3.5-omni technical report, 2026. URL https://arxiv.org/abs/2604.15804.

Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. Rema: Learning to meta-think for llms with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09501.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. Mas$^2$: Self-generative, self-configuring, self-rectifying multi-agent systems, 2025a. URL https://arxiv.org/abs/2509.24323.

Yanbo Wang, Zixiang Xu, Yue Huang, Xiangqi Wang, Zirui Song, Lang Gao, Chenxi Wang, Xiangru Tang, Yue Zhao, Arman Cohan, Xiangliang Zhang, and Xiuying Chen. Dyflow: Dynamic workflow framework for agentic reasoning, 2025b. URL https://arxiv.org/abs/2509.26062.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks, 2025c. URL https://arxiv.org/abs/2504.06821.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Shirley Wu, Parth Sarthi, Shiyu Zhao, Aaron Lee, Herumb Shandilya, Adrian Mladenic Grobelnik, Nurendra Choudhary, Eddie Huang, Karthik Subbian, Linjun Zhang, Diyi Yang, James Zou, and Jure Leskovec. Optimas: Optimizing compound ai systems with globally aligned local rewards, 2026. URL https://arxiv.org/abs/2507.03041.

Tianchi Xie, Minzhi Lin, Mengchen Liu, Yilin Ye, Changjian Chen, and Shixia Liu. Infochartqa: A benchmark for multimodal question answering on infographic charts, 2025. URL https://arxiv.org/abs/2505.19028.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. Comas: Co-evolving multi-agent systems via interaction rewards, 2026. URL https://arxiv.org/abs/2510.08529.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems, 2025. URL https://arxiv.org/abs/2503.03686.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025a. URL https://arxiv.org/abs/2503.14476.

Xinlei Yu, Chengming Xu, Zhangquan Chen, Yudong Zhang, Shilin Lu, Cheng Yang, Jiangning Zhang, Shuicheng Yan, and Xiaobin Hu. Visual document understanding and reasoning: A multi-agent collaboration framework with agent-wise adaptive test-time scaling, 2025b. URL https://arxiv.org/abs/2508.03404.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark, 2025. URL https://arxiv.org/abs/2409.02813.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025a. URL https://arxiv.org/abs/2502.04180.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025b. URL https://arxiv.org/abs/2509.03312.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks, 2025c. URL https://arxiv.org/abs/2410.11782.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita-Velez, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng Yan, Philip Torr, and Lei Bai. The landscape of agentic reinforcement learning for llms: A survey, 2026a. URL https://arxiv.org/abs/2509.02547.

Jiamu Zhang, Lingxi Zhang, Pengjun Lu, Qiyue Zhang, Yu-Neng Chuang, Zhengchen Li, Shuai Xu, Vipin Chaudhary, and Hanjie Chen. Rethinking the evaluation of efficiency methods for multi-agent systems, 2026b. URL https://arxiv.org/abs/2609.05933.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2025d. URL https://arxiv.org/abs/2410.10762.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026c. URL https://arxiv.org/abs/2510.04618.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning, 2025e. URL https://arxiv.org/abs/2505.00024.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems, 2025f. URL https://arxiv.org/abs/2505.00212.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines, 2025g. URL https://arxiv.org/abs/2507.22606.

Zhiwei Zhang, Xiaomin Li, Yudi Lin, Hui Liu, Ramraj Chandradevan, Linlin Wu, Minhua Lin, Fali Wang, Xianfeng Tang, Qi He, and Suhang Wang. Unlocking the power of multi-agent llm for reasoning: From lazy agents to deliberation, 2025h. URL https://arxiv.org/abs/2511.02303.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. Chartcoder: Advancing multimodal large language model for chart-to-code generation, 2025. URL https://arxiv.org/abs/2501.06598.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-mas: Multi-agent reinforcement learning for collaborative llms, 2026. URL https://arxiv.org/abs/2510.11062.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö. Arık. Multi-agent design: Optimizing agents with better prompts and topologies, 2026. URL https://arxiv.org/abs/2502.02533.

Dawei Zhu, Rui Meng, Jiefeng Chen, Sujian Li, Tomas Pfister, and Jinsung Yoon. Doclens : A tool-augmented multi-agent framework for long visual document understanding, 2025. URL https://arxiv.org/abs/2511.11552.

Xiaochen Zhu, Caiqi Zhang, Yizhou Chi, Tom Stafford, Nigel Collier, and Andreas Vlachos. Demystifying multi-agent debate: The role of confidence and diversity, 2026. URL https://arxiv.org/abs/2601.19921.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs, 2024. URL https://arxiv.org/abs/2402.16823.

# A Extended Discussion of Related Work

## A.1 Workflow Construction and Optimization

Early multi-agent frameworks coordinate agents through manually specified roles, procedures, and conversation patterns (Hong et al., 2024; Wu et al., 2023), and recent visual document understanding systems run agent pipelines designed by hand for their domain on fixed, prompted backbones (Zhu et al., 2025; Jin et al., 2026). Other methods automate this design while keeping every model fixed, searching on each benchmark's validation data for a single workflow or agent design written as code (Zhang et al., 2025d; Hu et al., 2025b), or generating a workflow graph for each task and revising it during execution (Niu et al., 2025). Methods that train the workflow generator use supervised fine-tuning on query-specific multi-agent systems (Ye et al., 2025) or learn from execution outcomes, training lightweight controllers that select operators (Zhang et al., 2025a) or communication edges (Zhang et al., 2025c) for fixed LLMs, a weak meta-agent that designs workflows for strong executors (Nie et al., 2025), or orchestrators that assign subtasks to fixed worker models (Nielsen et al., 2026) or select the next agent step by step (Dang et al., 2025). Such training also extends to several meta-agents that generate, configure, and rectify a multi-agent system whose agents run on a fixed model pool (Wang et al., 2025a). A separate line improves the workflow without updating weights, leveraging reusable routines from past experience (Wang et al., 2024), evolving the context a model conditions on (Zhang et al., 2026c), or replacing reinforcement learning with reflective prompt evolution (Agrawal et al., 2026). *However, in all these methods, the models that solve the workflow's subtasks remain fixed, which limits how much the workflow as a whole can improve.*

## A.2 Multi-Agent Training and Co-Evolution

Reinforcement learning increasingly optimizes multiple interacting LLM agents, but within an interaction structure fixed in advance for all task instances, such as a multi-turn discussion among agents that play the same role (Park et al., 2025), a meta-thinking agent that guides a reasoning agent (Wan et al., 2025), a workflow designed once per task domain (Zhao et al., 2026), or a document-understanding pipeline of planning, execution, judgment, and answering agents (Yu et al., 2025b). Co-evolution methods likewise fix how the models interact, pairing a challenger that proposes problems with a solver that answers them (Huang et al., 2026), or letting agents drawn at random propose, critique, and score solutions in a multi-round discussion whose scores serve as their rewards (Xue et al., 2026). Recursive agent training lets the task decomposition vary per instance, but trains a single policy as a root agent and as the copies of itself to which it delegates, all in the same role (Gandhi et al., 2026). Other works train each agent separately, optimizing it in its own reinforcement learning with a role-specific dataset and reward (Guo et al., 2026), or fine-tuning it to draft, verify, or refine an answer on trajectories sampled once from the untrained model (Motwani et al., 2025). *However, these agents learn either in isolation or only within the one structure or role they are trained in, which leaves them unprepared for tasks whose outcome depends on different agents working together in different ways.*

## A.3 Credit Assignment and Failure Attribution

Credit finer than a single outcome often comes from learned models, such as a process reward model trained alongside a single agent's policy to reward each action (Liu et al., 2025), or a local reward for each component of a compound system, learned from outcomes estimated by re-executing the components downstream of it (Wu et al., 2026). Other methods credit the meta-agents that design a multi-agent system by averaging the outcomes of executions branched at each of their decisions (Wang et al., 2025a). Failure attribution for multi-agent systems names the responsible agent and step with LLM judges (Zhang et al., 2025f) or with an attributor trained on failures labeled through counterfactual replay and fault injection (Zhang et al., 2025b), or tests failure hypotheses by re-executing the system with edited messages or plans (Ma et al., 2026). Even with full execution traces, such attributors identify the decisive failure step in only about a third of failed runs (Chen et al., 2026). Credit that adds no model or rollout instead compares actions taken from states that recur across a single agent's rollouts of the same task, and falls back to trajectory-level credit when states do not recur (Feng et al., 2025; Cheng et al., 2026). *However, such credit either costs extra models, labels, or executions, or holds only where a single agent revisits the same states, and it can*

*still misattribute failures. Consequently, agents that share one outcome lack a cheap and reliable signal of their own mistakes.*

### A.4 Evaluation of Multi-Agent Workflows

Methods that construct or optimize multi-agent workflows are commonly evaluated on single-question math, function-level code, and short question answering (Zhang et al., 2025d;a; Nie et al., 2025). On such data, automatically designed multi-agent systems improve on a self-consistent single agent by at most a few points (Singh et al., 2025), many reported successes of efficiency methods for multi-agent systems appear in settings where a single agent already preserves strong performance (Zhang et al., 2026b), and majority voting alone accounts for most of the gains attributed to multi-agent debate (Choi et al., 2025). When multiple agents are trained on single-question math, one agent comes to dominate while the other contributes little, collapsing the system into a single agent (Zhang et al., 2025h). Final answer accuracy can also remain high while the interactions among agents diverge from their intended behavior (So et al., 2026), and failure analyses find the gains of multi-agent systems on popular benchmarks often minimal (Cemri et al., 2025). *However, as such data can be solved without coordinating agents, evaluating on it cannot reliably assess whether a workflow's structure helps, and training on it provides little signal for learning beyond a single agent's effort.*

## B Limitations

By proposing **FloWright**, there are also limitations that confine the scope of what we demonstrate. First, our reinforcement learning experiments are mainly evaluated on two open models of a single family, `Qwen3.5-4B` and `Qwen3.5-9B` (§4). **FloWright** leverages the same signal whichever model powers a role, and the meta-distillation improves all four backbones we evaluate, including the two it never optimizes (§G.2). Nevertheless, we aim to incorporate more model families into our evaluation to strengthen our work. Second, our evaluation spans 44 arms over 12 datasets and 7 domains (§D.5), each a task whose inputs are fixed once the task begins. In future work, we plan to train roles at a larger scale, and to extend **FloWright** to dynamic environments, where the inputs a workflow reads and the components it needs change while the workflow runs.

## C Preliminary Study: The Gap in Existing Benchmarks

A multi-agent workflow is only warranted on a task that a single agent cannot already solve. We therefore ask two questions: what existing benchmarks demand of a system, whether a single agent or a workflow, and how much a workflow gains on them. We answer the first by surveying the evaluations that published workflow generation methods use, and the second by a controlled comparison in which the workflow and the single agent both run the untrained backbone `Qwen3-VL-8B`.

**What Existing Benchmarks Demand.** We collect every dataset that each of 20 workflow generation methods published at top-tier venues evaluates on (Zhang et al., 2025d; Hu et al., 2025b; Zhang et al., 2025a; Zhuge et al., 2024; Shang et al., 2025; Nie et al., 2025; Ye et al., 2025; Wang et al., 2025a; Dang et al., 2025; Ke et al., 2026; Nielsen et al., 2026; Su et al., 2025; Zhao et al., 2026; Zhang et al., 2025c; Niu et al., 2025; Zhang et al., 2025g; Zhou et al., 2026; Wang et al., 2025b; Hu et al., 2025a; Singh et al., 2025), which gives 121 datasets in total, counting a dataset once for each method that uses it. We classify these 121 datasets into six categories according to what a system needs to do to answer each question (Fig. 9a): single-question math (39), short-answer QA (25), function-level coding (21), tool use or agentic environments (16), reading comprehension (4), and other (16). The first three categories account for 85 of the 121, and a single competent agent is able to answer their questions in one pass [1].

**How Much a Workflow Gains on Them.** Leveraging `Qwen3-VL-8B` as the backbone, we compare LLM-generated multi-agent workflow against a single agent on two datasets of this kind (MMLongBench-Doc (Ma et al., 2024) and DocFinQA (Reddy et al., 2025)). We give the workflow and the single agent the same compute budget, so that any difference in accuracy is attributable to how the computation is organized rather than to how much of it each system is allowed (Fig. 9b). On

[1] These counts describe the 20 methods we survey, and we do not extrapolate them to the field.

the original MMLongBench-Doc (200-sample subset) the workflow reaches 40.50% and the single agent 48.00%. On the original DocFinQA (200-sample subset) the workflow reaches 33.00% and the single agent 38.00%. Neither dataset rewards multi-agent coordination. Each sample presents one document and one question, so a system reads through the document and extracts one answer, and no decision in the task requires ability or coordination beyond a single agent. The outcome therefore reflects what these benchmarks ask for, not a limitation of the workflow.

**Hardening Changes the Outcome.** On MMLongBench-Doc hardened by DATAWRIGHT, the same workflow reaches 51.61% against the single agent's 12.33% over 150 test samples, an improvement of 39.28 points. Through hardening (§D.1), the evidence a system needs to examine exceeds one effective context window. Therefore, the task requires deciding which input to read in what order, and spending the same number of turns on a single input no longer suffices. This is the condition under which the structure and coordination a workflow adds carry value [2].

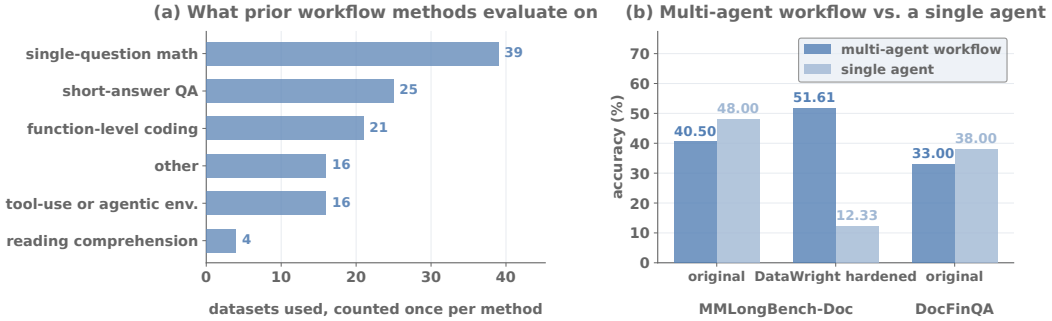


Figure 9: **Preliminary Study.** (a) Among the 121 datasets that the 20 workflow generation methods evaluate on: 85 of the 121 are single-question math, short-answer QA, or function-level coding, each of which a single competent agent can answer in one pass. (b) Comparing multi-agent workflows against a single agent given the same compute budget: workflows underperform the single agent on the original datasets, yet gain ↑ 39.28 on DATAWRIGHT-hardened dataset.

# D DATAWRIGHT: WORKFLOW-LEVEL DATA HARDENING

Datasets that a single agent can already handle can neither reveal nor optimize what a workflow adds beyond it (§C). To this end, we propose DATAWRIGHT to convert existing datasets into workflow-level tasks. Particularly, it builds each task by a rule-based *hardening strategy* at a *hardening level* $\ell$ (§D.1), adaptively selects the strategies each dataset supports (§D.2), and filters the data with quality control (§D.3). Each hardened task is graded as the mean of its sub-question scores, and the accuracy of each hardened dataset is calculated as the mean accuracy over its test tasks (§D.4). Alg. 1 summarizes the workflow-level data hardening procedure of DATAWRIGHT. We also summarize our data statistics in §D.5.

## D.1 HARDENING STRATEGIES

A hardening strategy converts samples of an existing dataset, each a question paired with the input it asks about, such as a document, a slide deck, a chart, *etc.*, into harder tasks by its rule, without any model in the loop. Solving a hardened task draws on the capabilities that a workflow brings beyond a single agent, including decomposing the task into sub-questions, routing each sub-question to a suitable input and agent, localizing the evidence that answers it, and aggregating the answers. Every strategy calls on these capabilities together, while each emphasizes a different one by its construction.

**Shared** bundles $\ell$ questions about the same input into one task. Solving it mainly exercises *decomposition*: the task splits into sub-questions that can be assigned and answered separately.

[2] We observe this reversal in our preliminary study, so we analyze it as an existence result that motivates DATAWRIGHT (§D), and not as a rate at which hardening reverses an outcome.

**Algorithm 1 DATAWRIGHT: Workflow-Level Data Hardening.** A sample $(q, z, y^\star)$ is a question $q$ with its input $z$ and reference answer $y^\star$; $\mathcal{B}_z$ denotes the samples of $\mathcal{B}$ on input $z$, and $|x|$ the number of original samples in task $x$. GROUP partitions samples into disjoint tasks of $\ell$ questions, and DECOY draws $\ell - 1$ inputs other than $z$, using every input a near-equal number of times.

**function** DATAWRIGHT($\mathcal{B}, \ell$)
  $\mathcal{B} \leftarrow \{(q, z, y^\star) \in \mathcal{B} \mid \text{VALID}(q, z, y^\star)\}$ ▷ §D.3
  $(\mathcal{B}^{\text{train}}, \mathcal{B}^{\text{test}}) \leftarrow \text{SPLIT}(\mathcal{B})$
  **for** $\sigma \in \{\sigma' \mid \text{SUPPORTS}(\sigma', \mathcal{B})\}$ **do** ▷ §D.2
    $\mathcal{T}_\sigma^{\text{train}} \leftarrow \text{HARDEN}(\mathcal{B}^{\text{train}}, \sigma, \ell)$, $\mathcal{T}_\sigma^{\text{test}} \leftarrow \text{HARDEN}(\mathcal{B}^{\text{test}}, \sigma, \ell)$
    **if** $\sum_{x \in \mathcal{T}_\sigma^{\text{test}}} |x| < 100$ **then**
      $\mathcal{T}_\sigma^{\text{train}}, \mathcal{T}_\sigma^{\text{test}} \leftarrow \varnothing$
    **end if**
  **end for**
  **return** $\{(\mathcal{T}_\sigma^{\text{train}}, \mathcal{T}_\sigma^{\text{test}})\}_\sigma$
**end function**

**function** HARDEN($\mathcal{B}, \sigma, \ell$) ▷ §D.1
  **if** $\sigma =$ Shared **then**
    **return** $\bigcup_z$ GROUP($\mathcal{B}_z, \ell$)
  **else if** $\sigma =$ Paired **then**
    **return** GROUP($\mathcal{B}, \ell$) ▷ distinct $z$ within a task
  **else**
    **return** $\{(q, \text{SHUFFLE}(\{z\} \cup \text{DECOY}(z, \ell - 1)), y^\star) \mid (q, z, y^\star) \in \mathcal{B}\}$
  **end if**
**end function**

**Paired** bundles $\ell$ questions, each with its own distinct input, into one task. Solving it mainly exercises *routing*: each sub-question is matched to the input that answers it.

**Decoy** keeps one question and places its input among $(\ell - 1)$ decoy inputs drawn from the same dataset. Solving it mainly exercises *localization*: finding the single input that answers the question. Every input serves as a decoy a near-equal number of times, never for its own question.

**Hardening level.** The level $\ell$ sets the number of questions per task for *shared* and *paired*, and the number of inputs per task for *decoy*. Each original question enters at most one task, and every dataset is split into training and test sets before hardening. Therefore, no question appears on both sides at any level. Accordingly, a hardened task's score is the mean of its sub-question scores (§D.4).

### D.2 ADAPTIVE STRATEGY SELECTION

Not every dataset supports every hardening strategy, so **DATAWRIGHT** selects the suitable strategies for each dataset from what its samples allow (SUPPORTS in Alg. 1). Each strategy has one prerequisite.

**Shared** requires inputs that each carry several independent questions. It is therefore unavailable for (1) datasets that pair every input with a single question, such as one question per coding problem (*e.g.*, TACO (Li et al., 2023)), per image (*e.g.*, MMMU-Pro (Yue et al., 2025)), per chart (*e.g.*, ChartCoder (Zhao et al., 2025)), *etc.*, and for (2) datasets whose questions about one input are steps of the same problem (*e.g.*, We-Math (Qiao et al., 2024)), where one step reveals another's answer.

**Paired** requires only questions with distinct inputs, which every dataset provides.

**Decoy** requires decoys that cannot answer the question, which holds only when the question identifies its own input, as questions about a specific document, slide deck, or financial filing do. For charts, images, and coding problems, a decoy of the same kind can match everything the question states, which would make the task ambiguous rather than harder, so **DATAWRIGHT** does not apply this strategy to them.

**Sample-size requirement.** A hardening level of a strategy is kept for a dataset only if its test set at that level holds at least 100 original samples, so that every accuracy score rests on enough samples to give a valid evaluation. Tab. 3 summarizes the strategies and levels selected for each dataset.

### D.3 Quality Control

Existing datasets contain defective samples, such as a question that its input cannot answer, an input that does not match its question, or a reference answer that the input contradicts. **DataWright** therefore validates the samples of each dataset before hardening (Valid in Alg. 1), mainly by quality filtering and automatic checks.

**Quality filtering.** On the training data, GPT-5.4 serves as an LLM judge: it examines each sample together with its reference answer and flags the answers it finds wrong. The flagged samples are removed, 203 of the 21,570 training samples (1.9%). For example, on DocFinQA, the judge removes 79 unanswerable questions, and on TACO it removes 36 problems whose statement relies on a figure that the dataset no longer provides.

**Automatic checks.** For code generation datasets, we implement automatic checks by running the reference solution of each sample, keeping the samples that can pass their own tests. For example, a TACO problem requires at least 10 tests, so that an incorrect code answer cannot pass by chance.

### D.4 Evaluation Metrics

Hardening changes how hard a task is, not how its answers are graded: every sub-question keeps the metric of its own answer type, and a task's score is the mean over its sub-questions. A hardened task $x$ holds $|x|$ sub-questions, the $i$-th carrying the reference answer $y_i^\star$ and receiving the answer $y_i$ that a workflow returns: $|x| = \ell$ under *shared* and *paired*, which bundle $\ell$ questions per task, and $|x| = 1$ under *decoy*, whose level counts inputs rather than questions. With $s(\cdot,\cdot) \in [0,1]$ denoting the metric of that sub-question's answer type, a task's score and a dataset's accuracy over its test tasks are:

$$S(x) = \frac{1}{|x|}\sum_{i=1}^{|x|} s(y_i, y_i^\star), \qquad \mathrm{Acc}(\mathcal{T}_\sigma^{\mathrm{test}}) = \frac{1}{|\mathcal{T}_\sigma^{\mathrm{test}}|}\sum_{x\in\mathcal{T}_\sigma^{\mathrm{test}}} S(x) \tag{6}$$

where a sub-question the workflow leaves unanswered scores $s = 0$. Tab. 2 summarizes the evaluation metrics for different types of tasks **FloWright** cover. We also show the task types of each dataset in Tab. 3.

**Multiple choice and lists.** Let $Y$ and $Y^\star$ be the sets of options, or of list items, that the answer and the reference name. The score is the overlap of the two sets against the larger of them:

$$s_{\mathrm{set}}(Y, Y^\star) = \frac{|Y \cap Y^\star|}{\max\left(|Y^\star|, |Y|\right)} \tag{7}$$

which reaches 1 only when the two sets agree. Extra names are penalized, just as missing ones are, so a workflow cannot gain by naming every option. A single-answer question is the case $|Y^\star| = 1$, where Eq. 7 is an exact match.

**Numeric.** A numeric answer is first normalized: thousands separators and currency symbols are removed, units are mapped to a canonical name, and a percentage and its fraction become one quantity. It is then compared at the precision the reference itself states, with $\mathrm{round}_{y^\star}(\cdot)$ rounding to that precision:

$$s_{\mathrm{num}}(y, y^\star) = \mathbb{1}[\mathrm{round}_{y^\star}(y) = \mathrm{round}_{y^\star}(y^\star)] \tag{8}$$

No relative tolerance is allowed, which is stricter than some of these benchmarks.

**Open-ended.** Let $\nu(\cdot)$ normalize case, punctuation, and spacing, and $\mathrm{tok}(\cdot)$ be the multiset of answer tokens. An exact match after normalization scores 1, and otherwise the score is the token-level F1 with the reference:

$$s_{\mathrm{text}}(y, y^\star) = \begin{cases} 1, & \nu(y) = \nu(y^\star), \\ \dfrac{2\,|\mathrm{tok}(y) \cap \mathrm{tok}(y^\star)|}{|\mathrm{tok}(y)| + |\mathrm{tok}(y^\star)|}, & \text{otherwise} \end{cases} \tag{9}$$

so a sub-answer that overlaps its reference without matching it earns partial credit. The second case applies inside *shared* and *paired* tasks. A *decoy* task holding a single question is scored by the first case alone.

**Code generation.** A code is graded by running it, never by comparing its text to a reference:

$$s_{\text{code}}(y) = \mathbb{1}[\textsc{Check}(y)] \tag{10}$$

where CHECK is a strict execution-based evaluation test: every unit test of the problem passes, the benchmark's own checker test accepts the code, or, for a dataset that provides no evaluation protocol, the code executes without errors and the code-rendered output is accurate against the reference answer.

**LLM as Judge.** An answer that no rule can decide is graded by an LLM judge that evaluates the answer directly against the reference. As such, we define an LLM judge as a grader that receives the reference answer and the predicted answer, and evaluates with one integer on a $\{0, \ldots, L\}$ scale whose every level its prompt pre-defines:

$$s_{\text{judge}}(y, y^\star) = \frac{1}{L}\,\textsc{Judge}(y, y^\star) \tag{11}$$

where the judge returns an integer in $\{0, \ldots, L\}$ and $L$ is the top of the scale its prompt defines. The prompt and the normalization thus stay in step. Each judge makes exactly one judgment, which keeps an answer self-consistent: a verdict paired with a separate similarity admits a pair that disagrees with itself, while anchored levels calibrate. A judge reads text or images, whichever the answer is made of, and a dataset that needs several judges gets one per aspect of the same answer, which the metric averages into that task's score.

For ChartCoder, we propose its evaluation as the mean of two LLM judged scores gated by execution: two judges at $L{=}5$ (Fig. 10) alongside the execution term of Eq. 10. The first LLM judge evaluates code by assessing the predicted code against the reference code. The second LLM judge evaluates the predicted chart rendered by the predicted code against the reference chart as two images. When execution fails, the score becomes 0:

$$\begin{aligned}
s^{\text{code}}_{\text{chart}}(y, y^\star) &= \tfrac{1}{5}\,\mathbb{1}[\textsc{Exec}(y)]\,\textsc{Judge}_{\text{code}}(y, y^\star)\,, \\
s^{\text{render}}_{\text{chart}}(y, y^\star) &= \tfrac{1}{5}\,\mathbb{1}[\textsc{Exec}(y)]\,\textsc{Judge}_{\text{render}}(\textsc{Render}(y), \textsc{Render}(y^\star))\,, \\
s_{\text{chart}}(y, y^\star) &= \tfrac{1}{2}\left(s^{\text{code}}_{\text{chart}}(y, y^\star) + s^{\text{render}}_{\text{chart}}(y, y^\star)\right)
\end{aligned} \tag{12}$$

where EXEC marks a code that executes successfully, and RENDER renders one to an image. Both judges score 0 unless the code executes without error. The two judges cover two different aspects of one answer: one evaluating the code from its source and the other the image that code renders, and their mean is the accuracy we present for each ChartCoder task. We use `GPT-5.6-luna` as the judge (§4). ChartCoder is the one dataset that requires a judge, so every ChartCoder number in our experiments is a judge score, and every other dataset carries the rule-based metric of its answer type defined above.

**Code judge** text only, comparing the code a workflow writes against the reference code

**System**

```
You are an impartial grader of matplotlib code. Your task is to judge whether the chart MODEL CODE would draw matches the chart REFERENCE CODE would draw. # How To Judge  Reason about what each program would RENDER, and judge only that. Work out the chart type each program produces, the data it plots, the number and order of its series and categories, and the title, axis labels, tick labels and legend text it sets. A program that reaches the same rendered chart by different means is fully correct. Ignore everything that does not change the rendered chart: variable and function names, comments, import order, whitespace [...]. Ignore colour, line style and marker choices unless REFERENCE CODE sets them explicitly. # Scores  5: would render the same chart. 4: same chart and data, a minor difference. 3: same chart type and most of the data. 2: same chart type, largely different data. 1: a different chart type. 0: no chart content at all. # Output Format Respond with ONLY a JSON object: {"score": <an integer from 0 to 5>, "reason": "<one short sentence>"}
```

**User**

```
# REFERENCE CODE: {reference}    # MODEL CODE: {prediction}
```

---

**Chart judge** two images, comparing the render of that code against the reference chart

**System**

```
You are an impartial grader of chart images.  Your task is to judge how closely a MODEL CHART reproduces a REFERENCE CHART. # How To Judge  Compare the two images on each of these aspects, then give one score for the whole chart.  Chart type [...] including whether subplots are used and how many.  Layout and element position [...]. Data: the number and order of series and categories, and the relative magnitudes, trends and proportions the chart shows. Text: the title, axis labels, tick labels and legend entries. Colour: whether series are distinguishable and whether the palette and any colour encoding carry the same meaning. Judge the data as drawn, and do not infer values the chart does not show. Ignore image resolution, figure size, font family, and small differences in margins or spacing. # Scores  5: the same chart on every aspect. 4: same type, layout, data and text, cosmetic differences only. 3: same type and layout, some data, text or colour differs. 2: same chart type only. 1: a different chart type, or unreadable. 0: the image is blank. # Output Format  Respond with ONLY a JSON object: {"score": <an integer from 0 to 5>, "reason": "<one short sentence>"}
```

**User**

```
The first image is the REFERENCE CHART and the second is the MODEL CHART. Score how closely the MODEL CHART reproduces the REFERENCE CHART.  [image: {reference chart}]  [image: {render of model code}]
```

Figure 10: **LLM Judge Prompts.** Each judge prompt carries its system instruction and its user message (§D.4), trimmed with `[...]`. Scores are graded to 0 only when code execution failed (Eq. 12).

Table 2: **Evaluation Metrics.** We summarize the evaluation metrics of different types of tasks covered by DATAWRIGHT (§D.4).

| Task Type | Evaluation Metric | Definition |
|---|---|---|
| | ***Question Answering*** | |
| multiple choices, one answer | the chosen option equals the reference option | Eq. 7 |
| multiple choices, multiple answers | the share of reference options named, lowered by every extra one | Eq. 7 |
| list of items | the share of reference items named, lowered by every extra one | Eq. 7 |
| numeric | equality at the reference's precision, after normalizing format and units | Eq. 8 |
| open-ended | exact match after normalization, otherwise the token-level F1 with the reference | Eq. 9 |
| | ***Code Generation*** | |
| code with unit tests | every test of the problem passes | Eq. 10 |
| code with a reference checker | the benchmark's own checker accepts the code | Eq. 10 |
| plotting code | rendered plot accuracy | Eq. 11-12 |
| | ***Aggregation*** | |
| a hardened task | the mean over its sub-questions, an unanswered one scoring zero | Eq. 6 |
| a dataset | the mean over its test tasks | Eq. 6 |

Table 3: **Data Statistics.** Every dataset DATAWRIGHT hardens, the strategies it supports, the task type its answers take, the input its questions carry, and the number of original questions each hardened set holds at level $\ell = 3$ and $\ell = 5$ (written $\ell = 3 / \ell = 5$). Counts are questions rather than bundles, since a task carries $\ell$ of them under *Shared* and *Paired*: bundling regroups the same questions into fewer and harder tasks as the level rises, and under *Decoy* the level changes only how many inputs surround the one that answers the question. A level is kept only when its test side holds at least 100 original questions (§D.2), which is why two *Shared* levels are absent. A dash "–" in the training column marks a dataset held out for evaluation only, whose questions all serve the test side.

| Dataset | Strategy | Task Type | Input | Test Questions | Training Questions |
|---|---|---|---|---|---|
| ***Document Understanding*** | | | | | |
| MMLongBench-Doc (Ma et al., 2024) | Shared | mixed | file | 960 / 820 | – |
| | Paired | mixed | file | 1,089 / 1,090 | – |
| | Decoy | mixed | file | 1,091 / 1,091 | – |
| LongDocURL (Deng et al., 2025) | Shared | mixed | file | 456 / 405 | 1,443 / 1,160 |
| | Paired | mixed | file | 540 / 540 | 1,782 / 1,780 |
| | Decoy | mixed | file | 541 / 541 | 1,783 / 1,783 |
| ***Finance Understanding*** | | | | | |
| DocFinQA (Reddy et al., 2025) | Shared | numeric | text | 609 / 275 | 5,031 / 4,250 |
| | Paired | numeric | text | 843 / 840 | 5,733 / 5,735 |
| | Decoy | numeric | text | 843 / 843 | 5,735 / 5,735 |
| ***Multimodal Slide Understanding*** | | | | | |
| SlideVQA-Doc (Tanaka et al., 2023) | Shared | open-ended | file | 1,791 / 1,390 | 8,637 / 7,230 |
| | Paired | open-ended | file | 2,214 / 2,215 | 10,617 / 10,615 |
| | Decoy | open-ended | file | 2,215 / 2,215 | 10,617 / 10,617 |
| ***Chart Understanding*** | | | | | |
| InfoChartQA-Text (Xie et al., 2025) | Shared | mixed | image | 2,610 / 2,365 | – |
| | Paired | mixed | image | 3,129 / 3,125 | – |
| InfoChartQA-Visual (Xie et al., 2025) | Shared | mixed | image | 132 / – | 576 / – |
| | Paired | mixed | image | 1,065 / 1,735 | 4,401 / 7,180 |
| InfoChartQA-Metaphor (Xie et al., 2025) | Shared | multiple-choice | image | 309 / – | – |
| | Paired | multiple-choice | image | 333 / 330 | – |
| ***Science*** | | | | | |
| MMMU-Pro (Yue et al., 2025) | Paired | multiple-choice | image | 1,728 / 1,730 | – |
| ***Math*** | | | | | |
| We-Math (Qiao et al., 2024) | Paired | multiple-choice | image | 1,740 / 1,740 | – |
| ***Coding*** | | | | | |
| TACO (Li et al., 2023) | Paired | code generation | text | 1,362 / 1,360 | 5,424 / 5,425 |
| DS-1000 (Lai et al., 2022) | Paired | code generation | text | 873 / 870 | – |
| ChartCoder (Zhao et al., 2025) | Paired | code generation | image | 999 / 1,000 | – |

### D.5 DATA STATISTICS

Tab. 3 summarizes all the datasets **DATAWRIGHT** hardens, together with their task types, input types, and the size of each hardened set at both levels. The level shapes these sizes directly: *shared* and *paired* divide a split's questions into tasks of $\ell$ questions, so raising the level yields fewer and harder tasks out of nearly the same pool of questions; while *decoy* keeps one question per task and raises the number of inputs it hides the gold one among, which leaves its task count unchanged. Because a task carries every question in its bundle, the test side of a kept level holds at least 100 original questions to form a valid test set.

**DATAWRIGHT** hardens 12 datasets across 7 domains. We only use four datasets to supply training datasets, which are split before hardening, so no hardened test task shares a question with training. We keep the remaining eight datasets to be evaluation-only. The hardened datasets span various types of answers, including numeric, multiple-choice, open-ended, and code, and a single task may mix several types. Input types range from slides and documents to charts and figures.

Each dataset supports only the hardening strategies its source data supports (§D.2), so the number of evaluation arms differs across datasets. **DATAWRIGHT** provides 44 arms for three regimes:

1. **In-distribution (4 arms).** The four datasets **FLOWRIGHT** trains on, each taken at the *paired* strategy and $\ell$=3: InfoChartQA-Visual, LongDocURL, SlideVQA-Doc, and TACO. Training sees no other arm.
2. **Out-of-distribution (13 arms).** The same four datasets under a strategy, a level, or both that training never sees, yielding 5 arms at $\ell$=3 and 8 arms at $\ell$=5.
3. **Out-of-domain (27 arms).** Eight datasets **FLOWRIGHT** never trains on: InfoChartQA-Text, InfoChartQA-Metaphor, DS-1000, MMLongBench-Doc, DocFinQA, MMMU-Pro, We-Math, and ChartCoder, yielding 14 arms at $\ell$=3 and 13 arms at $\ell$=5.

Fig. 12 shows every method on each of these arms.

## E FLOWRIGHT: INSIDE THE LEARNING

### E.1 CREDIT THE STRUCTURE: TRACED CREDIT, LEARNABLE CO-EVOLUTION

**Only genuine root causes are charged.** A failed node that has a failed dependency, direct or indirect, is treated as a cascaded failure, so its credit propagates to the root failure along the workflow's edges, for any topology. Infrastructure faults, such as a provider connection error or a content filter, are not the policy's fault and thus excluded. A node is judged by its final occurrence in $\tau_j$, so a step that fails early in a loop but recovers later is not charged. A fault that never surfaces as a node failure is left to the global signal.

**Per-role credit makes co-evolution learnable.** For example, when a workflow runs both the *Generator*'s design and the *Inventor*'s components, one scalar entangles their contributions. Localized credit restricts each role's credit term to its own nodes, so a role whose footprint is narrower than the workflow is not charged for failures outside it, *e.g.*, the *Inventor* is not charged for failures on nodes that use none of the components it creates, as the roles co-evolve (§3.4). This local credit is a bounded term of the global hierarchy in §3.3, alongside the role-specific validity $v^\rho$ that measures each role's own contribution (Tab. 4)

### E.2 HARNESS THE WORKFLOW: RICH IN SIGNAL, BROAD IN REACH

**Every step of progress earns signal.** The ladder makes the signal rich: it reflects every step of a role's progress, from a well-formed output through a valid contribution to a correct answer. Because each stage with a nonzero weight adds its own term, the policy learns from the full hierarchy of outcomes. Thus, even a partially successful workflow yields a graded, informative gradient.

**One shared harness generalizes to every role.** The invariant tail $w_e\, e + w_a\, a$ carries across every role: the harness always runs a workflow and evaluates its answer, and only the *source* of the workflow or of its execution differs. Every role's own quality enters through its own *validity* reward $v^\rho$ at the same stage of the ladder, deterministic and never a learned judge. Because the whole framework is

Table 4: **Per-Role Validity.** What each role's own output needs to satisfy, split into the upstream roles that build a workflow and the downstream agent that executes it (Fig. 1). Every reward is deterministic, so no learned judge enters the reward (§E.2).

| Role | Format Reward ($f$) | Validity Reward ($v^\rho$) |
|---|---|---|
| | ***Upstream: Building a Workflow*** | |
| *Planner* | its plan parses | *helpfulness*: the graded gain its plan brings over the same flow without one |
| *Generator* | its workflow parses | *structure*: the workflow is a valid graph over the pool(s) |
| *Inventor* (*Generator Skill*) | every creation decision and authored component parses | *deciding* and *authoring*: reused components resolve to the pool, and newly authored ones instantiate and run |
| | ***Downstream: Executing a Workflow*** | |
| *Downstream* agent | its output parses | *step format* and *step liveness*: every node it runs reaches an answer of its own |

shared (Fig. 1), the harness generalizes to any role, which is what makes co-evolution across arbitrary roles practical (§3.4).

### E.3 Define the Reward: Validity Reward in the Hierarchical Ladder

The reward of Eq. 3 is built from five terms. Whether the role's output *parses* ($f$) gates the rest: an output that cannot be read leaves nothing to execute. Whether the role's contribution is *valid* ($v^\rho$) is a role-specific term, and §E.2 states why the remaining terms are not: the harness always runs a workflow and grades its answer, so whether the workflow *executes* to a final answer ($e$) and how *correct* that answer is ($a$) are shared by every role (Eq. 3). The credit term $c_j^\rho$ (§E.1) then charges a role only for the nodes it authors. Thus, only a parse failure gates the reward, while validity rewards without gating. A correct answer requires an execution that reaches one, so a role earns dense partial credit for how far its contribution carries the workflow.

A role earns its validity reward when its own output effectively contributes to the workflow, and Tab. 4 shows what a contribution amounts to for each role. Four forms arise: (1) *structural correctness* rewards a workflow that is well formed against the pool; (2) *deciding and authoring* reward components that resolve to an existing pool entry or, when newly authored, instantiate and run; (3) *helpfulness* rewards what the role's intervention adds, calculated as the graded gain over the same flow without it, suiting a role that steers the flow; and (4) *step format* and *step liveness* reward every node the role runs for reaching an answer of its own. A reward resting on a semantic judgment (*e.g.*, the coherence of a plan) is deliberately excluded: a semantically correct output can also lead to the wrong final answer, whereas an outcome-grounded gain measures the actual contribution.

### E.4 Wire the Workflow: Representing Workflows In Different Topologies

A workflow needs to be properly written down to enable seamless upstream generation and downstream execution. **FloWright** allows four workflow topologies (Fig. 11): a step schema, an agent graph, a state machine, and a block code. The four topologies differ only in where the control flow is recorded. A schema states it in a separate block, a graph types each edge, a state machine attaches conditions and bounds to transitions, and a code carries it in the type of a block. Each is converted into the same directed graph before harness execution: nodes hold the agent with its instruction and inputs, and edges hold sequence, parallel, conditional and bounded-loop relations, and a cycle that no topology marks as a loop is reclassified as a bounded one so the graph stays schedulable. The control operations convert the same way in every topology. Execution, credit, and evaluation therefore read one graph and never observe which topology produces it. For this reason, **FloWright** wires different topologies into the shared workflow graph (§3.1), and thus the topology a generator writes in cannot change what the harness measures.

**Task** ($\ell = 3$) `Please answer all the following three questions below. Each question has its own image(s) [...]`

**Schema** steps with dependencies, and one `control_flow` block


```
{"steps": [{"step_id": "step_1", "agent": "Visual Analyst", "instruction": "Analyze the images for Question 1 [...] return only the letter", "inputs": {"image_1.jpg": [...]}, "outputs": ["Answer_Q1"], "depends_on": []}, [...]],
"control_flow": {"type": "sequential", "loop_steps": [], "max_iterations": null}}
```


**Graph** agent nodes, and typed edges that carry the control flow


```
{"agents": [{"agent_id": "visual_analyst_agent", "agent": "visual_analyst_agent", "instruction": "Analyze the images to answer three questions [...]", "inputs": [...], "outputs": ["Answer_Q1", "Answer_Q2", "Answer_Q3"]}],
"flows": [{"source": "visual_analyst_agent", "target": "visual_analyst_agent", "type": "sequence", "condition": "true", "max_iterations": 1}]}
```


**State machine** states, and transitions that carry conditions and bounds


```
{"states": [{"state_id": "Q1_Analysis", "agent": "VisualDataAnalyzer", "action": "Analyze [...] select the correct option letter", "outputs": ["Answer for Question 1"]}, [...]],
"transitions": [{"from_state": "Q1_Analysis", "to_state": "Q2_Analysis", "condition": "answer is a single letter", "max_iterations": 3}, [...]],
"start_state": "Q1_Analysis", "terminal_states": ["Final_Aggregation"]}
```


**Code** blocks whose `type` carries the control flow


```
{"program": {"language": "workflow_pseudocode", "blocks": [{"block_id": "block_1", "type": "step", "agent": "visual_analyst", "instruction": "Analyze the chart and the crop [...] return the option letter", "outputs": ["Answer_Q1"], "depends_on": [], "condition": null, "max_iterations": 3}, [...]]}}
```


**What the harness runs** every topology above becomes the same directed graph

`steps: {step_1, step_2, step_3, aggregate}` edges: `step_1` $\xrightarrow{\text{sequence}}$ `step_2` $\xrightarrow{\text{sequence}}$ `step_3` $\xrightarrow{\text{sequence}}$ `aggregate`

`policy: {start: step_1, terminal: aggregate, parallel: false}`

Figure 11: **Workflow Topologies.** We show an example of the same hardened task represented in four topologies: a step schema, an agent graph, a state machine, and a block code. Each topology example is directly extracted from our evaluation outputs and trimmed with `[...]`. The four topologies differ only in where the control flow is written down, *e.g.*, a separate `control_flow` block, a typed edge, a transition, a block type, *etc*. Each is then converted into a shared directed graph structure, so execution, credit, and evaluation stay uniform without knowing which topology produced it (§E.4).

What the choice does change is how reliably the generator writes a workflow worth running: whether its output parses, whether the structure it declares is valid, and how well the resulting workflow performs. Thus, this becomes an empirical question that we measure via an ablation over the four topologies in §5.2.

# F Implementation Details

## F.1 Reward Configuration

Every role instantiates one hierarchical ladder, so the reward weights differ only where the roles differ. We summarize per-role reward weights in Tab. 5. Format carries 0.1 for every role, since parsing is a gate rather than an achievement. Across different roles, the answer term always dominates at 0.6 to 0.8. We set the execution weight to 0.0 for every role in main experiments: a workflow that does not run cannot answer, so the answer term already covers it, and weighting execution on its own would pay a workflow for running cleanly while answering wrongly. Coding tasks reward the term at 0.1, since a workflow there reaches an answer only once its code executes, so the ladder sets this weight by task type (§G.3). The validity term holds what each role alone controls, and its components take the weight the answer term leaves: one component for the *Generator* and the steering roles, two for the *Inventor* and the *Downstream* agent. The credit penalty of §E.1 rides on top at 0.1, and the ablation that switches it off sets it to 0.0 (§5.2).

Table 5: **Per-Role Reward Configuration.** Every role instantiates the same hierarchical ladder (Eq. 3). The validity term is role-specific, and the credit penalty rides on top of the weighted sum (§E.1).

| Role | Format ($w_f$) | Validity ($w_v$) | Answer ($w_a$) | Credit ($w_c$) |
|---|---|---|---|---|
| *Planner* | 0.1 | *helpfulness* 0.1 | 0.8 | 0.1 |
| *Generator* | 0.1 | *structure* 0.1 | 0.8 | 0.1 |
| *Inventor* (*Generator Skill*) | 0.1 | *grounding* 0.1 + *authoring* 0.1 | 0.7 | 0.1 |
| *Critic* | 0.1 | *helpfulness* 0.3 | 0.6 | 0.1 |
| *Downstream* agent | 0.1 | *step format* 0.025 + *step liveness* 0.075 | 0.8 | 0.1 |

Table 6: **Experiment Configuration.** We summarize the experiment configurations for training and evaluation (§4). The same configuration applies across all four evolution modes and roles.

| Setting | Value | Note |
|---|---|---|
| ***Training Configuration*** | | |
| Policy optimization | GRPO / DAPO / CISPO | ablated; all three instantiate Eq. 4 |
| Learning rate | $1 \times 10^{-6}$ | constant, uniform for all settings |
| Group size $m$ | 8 | rollouts per task |
| Batch size | 8 | tasks per step |
| Epochs per data pass | 1 | constant, uniform for all settings |
| Clipping ratio | 0.2 | symmetric for GRPO; DAPO decouples the upper bound |
| KL loss coefficient | 0.001 | against the reference policy |
| Rollout sampling | $T$=1.0, top-$p$=0.95, top-$k$=20 | temperature kept at 1.0 for group diversity |
| ***Evaluation Configuration*** | | |
| Decoding | greedy, $T$=0 | deterministic, so a reported number is reproducible |
| Baseline compute budget | matched | each baseline receives the budget of the workflow it is compared against |
| Seed | 12345 | shared by hardening, splitting, and evaluation |

### F.2 Experiment Configuration

As **FloWright** contributes a training paradigm rather than an optimizer, the objective of Eq. 4 admits any policy-gradient method. We instantiate it with GRPO, DAPO, and CISPO, ablating the choice. All three draw $m$ rollouts per task and score them against their group rather than a learned critic, so they differ in how that group is turned into an update: GRPO normalizes token-level advantages within the group, DAPO decouples the clipping bounds and drops the groups that carry no signal, and CISPO clips the importance-sampling weight instead of the update, so a token far from the old policy is down-weighted rather than dropped from the gradient. We summarize our experiment configuration details in Tab. 6.

### F.3 Notation

We summarize the core notations in Tab. 7. An index $i$ denotes the $i$-th item of the list in scope, and in §3 it indexes tasks.

Table 7: **Notation.** We summarize notations used in our paper, grouped by the concept they denote, from tasks and workflows (§2) to rollouts, credit, reward, optimization (§3), and data hardening (§D). In §3, the indices $i$, $j$, and $k$ run over tasks, rollouts, and distinct workflows, respectively, and symbols indexed by $\rho$ are specific to role $\rho$, *e.g.*, $c_j^\rho$ is the credit of role $\rho$ in the $j$-th rollout.

| Symbol | Meaning | Introduced in |
|---|---|---|
| | ***Tasks and Workflows*** | |
| $x \sim \mathcal{X}$, $x_i$ | a task drawn from the task distribution; $x_i$ is the $i$-th task | §2, 3 |
| $G = (V, E)$ | a workflow: a directed graph whose nodes $V$ are agents, tools, or skills, and whose edges $E$ carry the data and control flow | §2 |
| $\mathcal{G}$ | the space of workflows | §3 |
| $y$ | the answer a workflow returns | Eq. 1 |
| | ***Harness and Signal*** | |
| $\mathcal{H}$ | the harness, which executes and grades a workflow | Eq. 1 |
| $h(G, x) \in [0, 1]$ | the harness signal, serving as the evaluation metric, the training signal, and the test-time objective | Eq. 1 |
| $\tau_j$ | the execution trace of $G_j$ produced by $\mathcal{H}$ | §3.2 |
| | ***Roles and Policies*** | |
| $\rho \in \mathcal{R}$ | a role, and the set of roles | §2 |
| $\pi_\rho$, $\theta_\rho$ | the policy of role $\rho$, and its parameters | §2 |
| $\pi_g$ | the policy of the *Generator* | Eq. 1 |
| $\rho'$ | ranges over all roles in the flow | Eq. 4 |
| | ***Pool*** | |
| $\Omega$ | the pool of reusable components a workflow is built from | §2 |
| $\Omega_{agent}$, $\Omega_{tool}$, $\Omega_{skill}$ | the agent, tool, and skill pools; $\Omega$ is their union under modular granularity | §3.1 |
| $\Omega_{capsule}$ | the capsule pool; $\Omega = \Omega_{capsule}$ under coarse granularity | §3.1 |
| | ***Rollouts*** | |
| $j$, $m$ | the rollout index, and the number of rollouts of a role being optimized | Fig. 2 |
| $k$, $M$ | the distinct-workflow index, and the number of distinct workflows the $m$ rollouts span | Fig. 2 |
| $G_j = (V_j, E_j)$ | the workflow the $j$-th rollout is graded on | Fig. 2 |
| $o_j^\rho$ | the output of role $\rho$ in rollout $j$ | §3.3 |
| $O^\rho = \{o_j^\rho\}_{j=1}^m$ | the $m$ rollouts of role $\rho$ being optimized | Fig. 2 |
| | ***Credit and Reward*** | |
| $V_j^\rho \subseteq V_j$ | the nodes of $G_j$ that role $\rho$ authors | §3.2 |
| $F_j^\rho \subseteq V_j^\rho$ | the nodes of role $\rho$ that root-cause a failure | §3.2 |
| $c_j^\rho \in [-1, 0]$ | the structure-aware credit of role $\rho$ | Eq. 2 |
| $f$, $v^\rho$, $e$, $a$ | whether the output parses, whether role $\rho$'s contribution is valid, whether the workflow executes to a final answer, and how correct the answer is | §3.3 |
| $w_f, w_v, w_e, w_a, w_c$ | per-role weighting coefficients | Eq. 3 |
| $h_j^\rho = h^\rho(G_j, x_i)$ | the hierarchical reward of role $\rho$ in rollout $j$ | Eq. 3 |
| | ***Optimization*** | |
| $\mathcal{J}(\pi_\rho)$ | the objective optimized for role $\rho$ | Eq. 4 |
| $P$, $P^\star$ | a reusable prior that conditions generation, and the optimal prior | Eq. 5 |

| Symbol | Meaning | Introduced in |
|---|---|---|
| | ***Data Hardening*** | |
| $\mathcal{B}$, $(q, z, y^\star)$ | a source dataset, and one of its samples: a question $q$, its input $z$, and its reference answer $y^\star$ | Alg. 1 |
| $\sigma$ | a hardening strategy: *Shared*, *Paired*, or *Decoy* | §D.1 |
| $\ell$ | the hardening level: questions per task for *Shared* and *Paired*, and inputs per task for *Decoy* | §D.1 |
| $\mathcal{T}_\sigma^{\text{train}}$, $\mathcal{T}_\sigma^{\text{test}}$ | the hardened tasks that strategy $\sigma$ builds from a dataset's training split, and those it builds from its test split | Alg. 1 |
| $\lvert x\rvert$ | the number of sub-questions a hardened task $x$ holds, one per original sample bundled into it | Alg. 1 |
| | ***Evaluation Metrics*** | |
| $y_i$, $y_i^\star$ | the answer a workflow returns for the $i$-th sub-question of a hardened task, and that sub-question's reference answer | Eq. 6 |
| $s(y_i, y_i^\star) \in [0, 1]$ | the score of one sub-question, given by the metric that its answer type uses | Tab. 2 |
| $s_{\text{set}}$, $s_{\text{num}}$, $s_{\text{text}}$, $s_{\text{code}}$ | the metrics of set-valued, numeric, open-ended, and code answers | Eq. 7–10 |
| $S(x)$ | a hardened task's score: the mean of the scores of its sub-questions | Eq. 6 |
| Acc | a dataset's accuracy: the mean task score over the test tasks of a strategy and level | Eq. 6 |
| $Y$, $Y^\star$ | the options, or list items, that an answer names, and those that its reference names | Eq. 7 |
| $\text{round}_{y^\star}(\cdot)$ | rounding to the precision that the reference answer states | Eq. 8 |
| $\nu(\cdot)$ | normalization of case, punctuation, and spacing in a text answer | Eq. 9 |
| $\text{tok}(\cdot)$ | the multiset of tokens of a text answer | Eq. 9 |
| $\mathbb{1}$ | the indicator: 1 when its condition holds, and 0 otherwise | Eq. 8 |
| CHECK | the strictest execution-based test that a code dataset provides for generated code | Eq. 10 |
| JUDGE | an LLM judge's verdict: one integer on the anchored scale its prompt fixes | Eq. 11 |
| $L$ | the top of that anchored scale, by which the verdict is normalized | Eq. 11 |
| $s_{\text{judge}}$ | the metric of an answer no rule decides: a judge's verdict over $L$ | Eq. 11 |
| EXEC | whether generated code executes successfully, beyond CHECK assessment above | Eq. 12 |
| RENDER | the chart image that executing the code produces | Eq. 12 |
| $s_{\text{chart}}^{\text{code}}$, $s_{\text{chart}}^{\text{render}}$, $s_{\text{chart}}$ | the code and render judges of a chart answer, and their mean | Eq. 12 |

# G A DEEPER DIVE BENEATH THE RESULTS

## G.1 UNFOLD THE MEANS: CONSISTENT GAINS ACROSS DATASETS, STRATEGIES, AND LEVELS

**FLOWRIGHT improves performance across diverse datasets, hardening strategies, and task complexities.** As shown in Fig. 12, the improvement holds on every dataset: under multi-agent co-evolution, all 12 datasets gain over the untrained workflow, with up to $+7.34\%$ on LongDocURL and $+5.13\%$ on ChartCoder. So no domain is traded away for another. It also holds on every hardening strategy, and grows as the strategy demands more of the workflow: $+3.57\%$ on *shared*, $+4.11\%$ on *paired*, and $+4.68\%$ on *decoy*. The improvement further holds at both levels of task complexity, with multi-agent co-evolution gaining $+4.31\%$ out-of-domain at $\ell$=3 and $+4.09\%$ at $\ell$=5, and $+5.80\%$ and $+6.21\%$ out-of-distribution (Tab. 1). Consequently, increased task complexities does not erode the gain. Harnessing the workflow therefore improves performance across different datasets, strategies, and levels DATAWRIGHT spans, rather than on a favorable subset of them.

## G.2 GENERALIZE THE HARNESS: OPTIMIZATION TRANSFERS TO DIFFERENT SETTINGS

**FLOWRIGHT generalizes to roles powered by backbones it never trains alongside.** As shown in Fig. 13, powering one upstream role with `Qwen3.5-4B` and other roles with `Qwen3.5-9B` performs

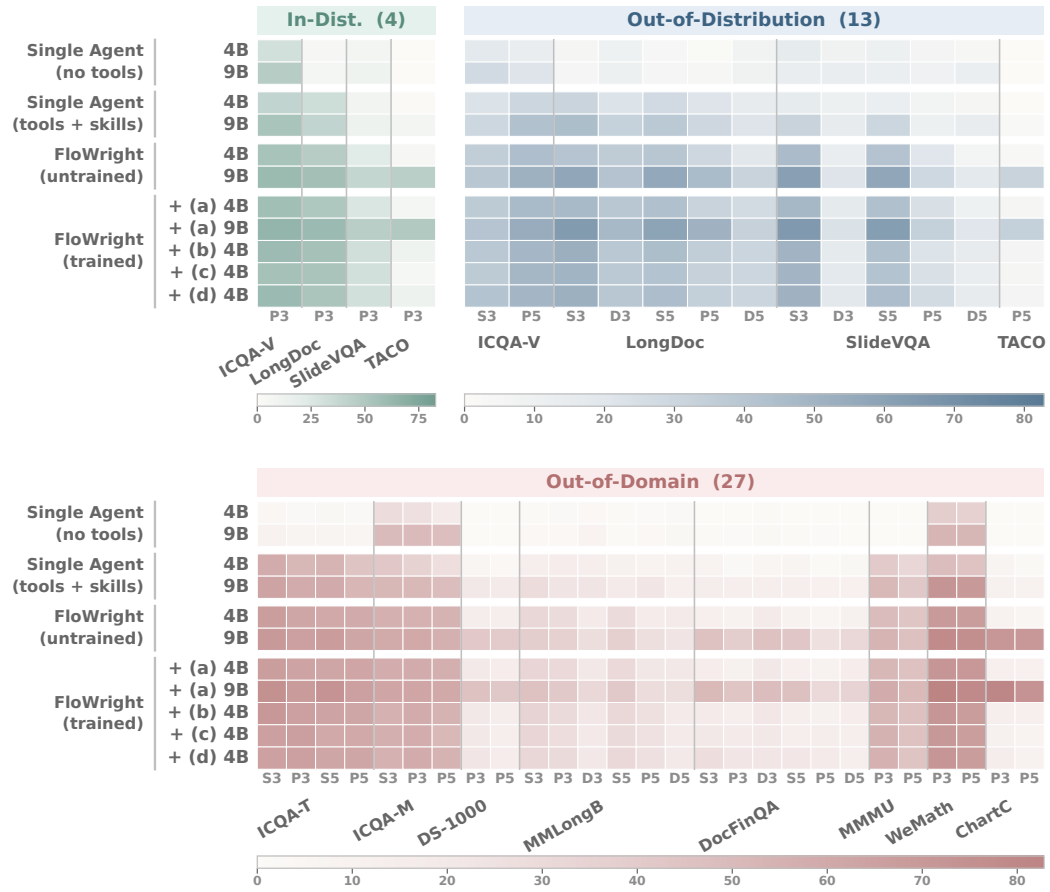


Figure 12: **Per-Arm Results Behind the Mean Scores.** Complementing the mean accuracy scores in Tab. 1, we show the accuracy (%) of each method on the 44 evaluation arms, grouped into the three regimes (Tab. 1, §D.5): in-distribution, out-of-distribution and out-of-domain. We denote the hardening strategy and level as: S/P/D = Shared/Paired/Decoy, 3/5 = level. Color intensity encodes accuracy on a shared scale, and hue identifies the regime.

between the two single-agent baselines, gaining $+11.91\%$ over the all-4B setting when `Qwen3.5-4B` powers the *Generator* and $+10.77\%$ when `Qwen3.5-4B` powers the *Inventor* skill. Exchanging the two models reverses the result: powering the *Generator* and the *Inventor* skill with `Qwen3.5-9B` and the downstream agent with `Qwen3.5-4B` reaches $29.57\%$, below the all-4B baseline of $32.09\%$. A stronger upstream model thus builds workflows for a downstream agent of its own capability, which a less capable downstream agent executes less effectively. Consequently, when using different backbones to power upstream and downstream roles, the downstream agent bounds what a workflow performs however well it is built. Optimizing a single 4B role then raises accuracy above the all-9B baseline of $47.50\%$, to $47.67\%$ for the *Generator*, $47.96\%$ for the *Inventor* skill, and $48.50\%$ for co-evolving the two together. *An optimized role transfers its gain when other roles run on a different backbone it never trains with.* Train-time optimization therefore generalizes across models that power the remaining roles.

**FLOWRIGHT improves performance whichever role it optimizes.** As shown in Fig. 14, every single-role optimization outperforms the untrained baseline, and every co-evolving optimization gains more: $+2.83\%$ for the *Generator*, $+0.90\%$ for the *Inventor* skill, and $+0.88\%$ for the downstream agent alone, against $+4.06\%$ for *Generator*-skill co-evolution, $+3.06\%$ for upstream-downstream co-evolution, and $+5.03\%$ for multi-agent co-evolution. The *Generator* is the largest single lever, gaining $+2.59\%$ out-of-domain where the other two roles stay flat. *The role that designs every workflow returns the largest gain among single roles.*

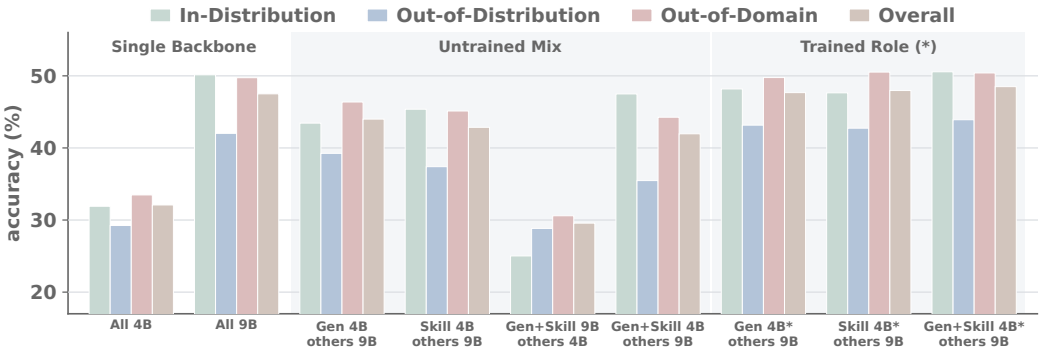


Figure 13: **Roles Powered by Different Backbones.** Accuracy (%) across three evaluation regimes over all 44 arms (§4). *Single Backbone* uses one backbone for every role. *Untrained Mix* uses a `4B` or `9B` backbone to specific roles, with no role optimized. *Trained Role* (∗) uses the RL-trained `4B` role, with `Qwen3.5-9B` powering other roles. *Gen* and *Skill* name the *Generator* and the skill it co-evolves with, and *others* covers every remaining role.

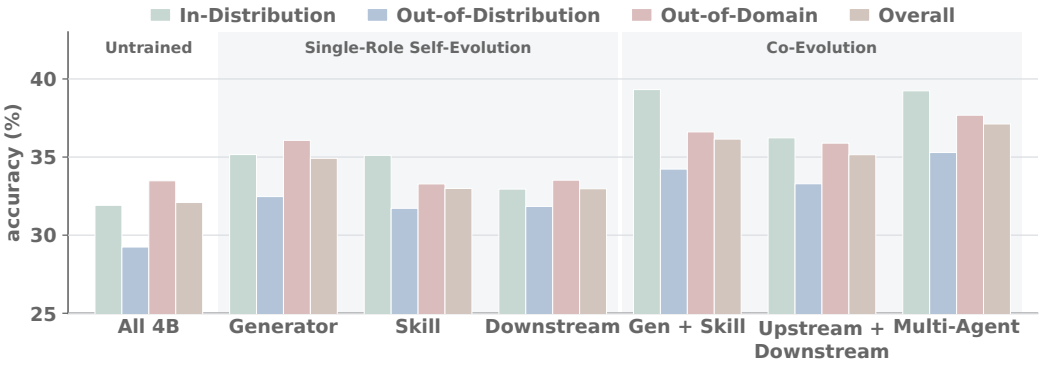


Figure 14: **Evolution of Different Roles.** Accuracy (%) across three evaluation regimes over all 44 arms (§4). Every role runs `Qwen3.5-4B`, with the only variable as which role the harness optimizes. *Single-Role Self-Evolution* optimizes the *Generator*, skill, and downstream agent individually. *Co-Evolution* optimizes two or more together (§3.4).

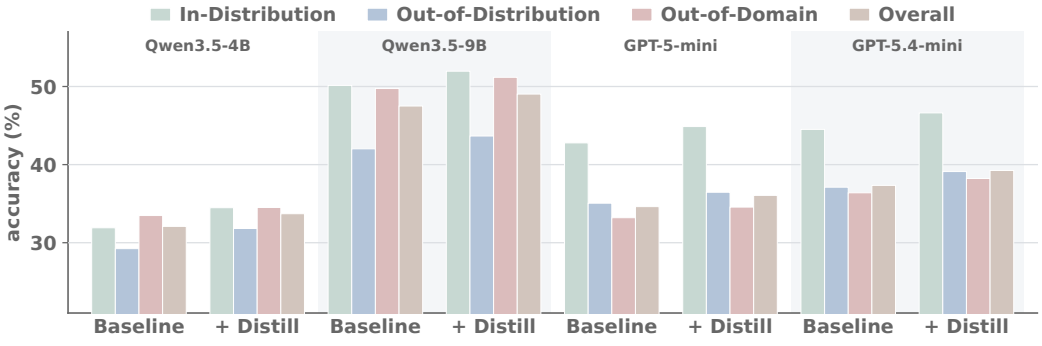


Figure 15: **Meta Distillation Across Backbones.** Accuracy (%) across three evaluation regimes over all 44 arms (§4). Each backbone appears as an untrained *Baseline*, compared with *+Distill* providing the meta prior distilled at five shots (§3.5).

**Meta distillation improves every backbone, including those FLOWRIGHT never optimizes.** As shown in Fig. 15, a meta-distilled prior gains in all three regimes on all four backbones, and overall by $+1.62\%$ on `Qwen3.5-4B`, $+1.53\%$ on `Qwen3.5-9B`, $+1.44\%$ on `GPT-5-mini`, and $+1.91\%$ on `GPT-5.4-mini`. Every backbone keeps its weights fixed, so each gain comes from the prior alone, and `GPT-5.4-mini` gains most among the four. *What the harness distills at test time transfers to models whose weights stay beyond reach.* Optimizing the signal thus improves a model FLOWRIGHT trains and a model it only prompts.

### G.3 WEIGH THE EXECUTION: THE LADDER ADAPTS TO THE TASK TYPE

**Weighting execution adds the most on coding tasks.** As shown in Fig. 16, an execution weight of $w_e$=0.1 improves over the untrained baselines, gaining $+3.98\%$ and $+5.68\%$ on the coding arms and $+0.59\%$ and $+2.03\%$ over all 44 arms, for `Qwen3.5-4B` and `Qwen3.5-9B` respectively. On coding, where a workflow reaches an answer only once its code executes to earn score based on the portion of evaluation tests it passes, $w_e$=0.1 outperforms $w_e$=0.0 by $+1.69\%$ and $+2.12\%$. Over all arms, where the answer term already covers whether a workflow reaches an answer, $w_e$=0.0 leads by $+2.24\%$ and $+1.86\%$. This is also why our main experiments hold the execution weight at 0.0 (§F.1). *The hierarchical ladder therefore servers as a paradigm whose weights follow what each task type demands, with execution earning its own weight where the task rewards running the workflow.*

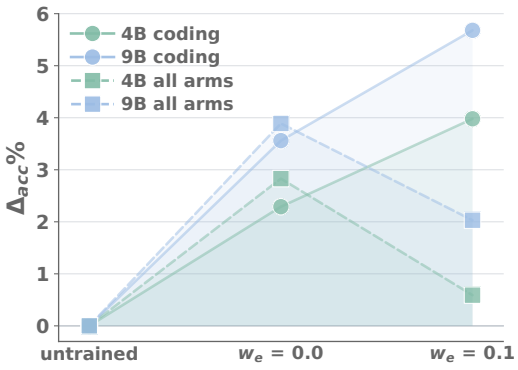


Figure 16: **Execution Reward by Task Type.** $\Delta_{acc}(\%)$ against untrained baselines, on the four coding arms (TACO and DS-1000) and over all 44 arms (§4). Every other ladder weight stays fixed (Tab. 5).