ARBIGRAPH: COMPOSING ARBITRARY VERIFIABLE TASKS FOR AGENT EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language model agents must often solve a sequence of problems and retain earlier answers as the workflow grows. Evaluating problems one at a time does not show how well agents retain and use earlier answers. To evaluate this ability, we introduce ARBIGRAPH, a framework for generating benchmark instances that combine problems with verifiable answers into graphs of dependent tasks. Each task is a problem, and each connection specifies an earlier answer that the agent must report to obtain the next problem's input. An incorrect report returns an alternative input whose verified answer differs from the intended answer. This mechanism supports tasks from different domains without requiring conversions between their input and output types. We test the framework with math problems and tasks that ask agents to predict the output of Python code. We evaluate three models on the same problems both in isolation and in three topologies, each containing 18 tasks. We construct an independent-error baseline by multiplying the isolated accuracies of the final task and all tasks it depends on. Averaged across the three topologies and the math and Python categories, measured accuracy falls below this baseline by 45.0 percentage points for Gemma-4-31B-it, 35.6 for Qwen3.8-27B, and 11.0 for GPT-OSS-120B. These additional losses indicate that solving problems within a larger workflow adds difficulty beyond the accumulation of errors on isolated problems. We also study how graph structure affects accuracy on the final task while keeping the number of tasks fixed.

1 INTRODUCTION

Language model agents use tools to solve problems over many steps. As a workflow grows, agents must keep track of earlier answers, update them when needed, and ignore information that is no longer useful. Solving each problem correctly in isolation is not enough to ensure success across the whole workflow. Existing evaluations show that models struggle with distracting information, changes in the position of relevant information, and long interaction histories (Liu et al., 2024; Hsieh et al., 2024; Bai et al., 2024b;a; Maharana et al., 2024). These evaluations examine how models use context through documents, conversations, and memory tasks. Other work combines multiple tasks: REST presents several problems at once (Pan et al., 2026), SIFo tests sequences of instructions (Chen et al., 2024), and Composition-RL combines verifiable problems into new questions (Xu et al., 2026). RACES connects tasks with compatible input and output types using a set of composition operators (Xiang et al., 2026). We study how agents perform when the same problems are connected in different ways. This lets us vary the dependencies while keeping the difficulty of the individual problems fixed.

Connecting existing problems is difficult when one problem's answer must become another's input. Their input and output types may differ, requiring a conversion and instructions that explain it to the agent. A task that depends on several earlier answers also needs a rule for combining them. Some conversions can hide errors: rounding, for example, can turn both a correct and an incorrect answer into the same input for the next task.

We introduce ARBIGRAPH, a framework for generating benchmark instances from problems with verifiable answers. Each instance is a graph: every node contains a concrete problem, and each edge

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.

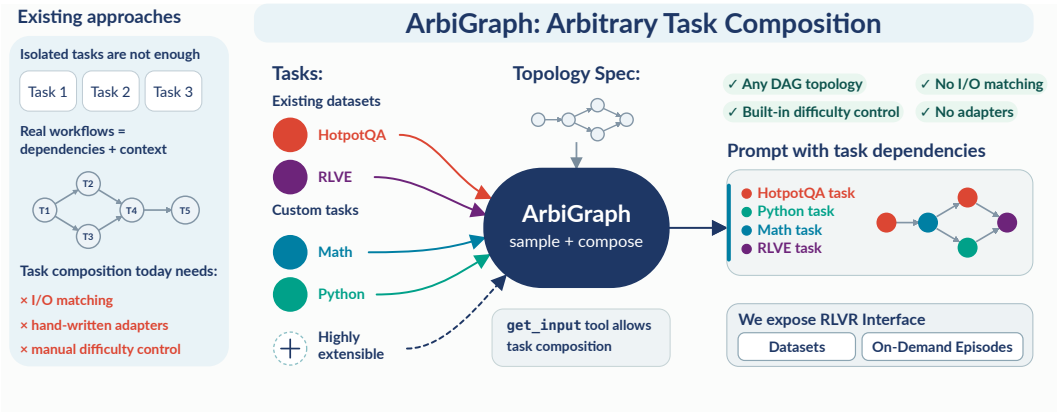

Figure 1: Overview of ARBIGRAPH. Custom tasks and existing datasets provide native, independently verifiable sources for user-specified task graphs, without pairwise adapters or input/output type matching. Answer-gated input access supports agent evaluation and RLVR training.

identifies an earlier answer needed by a later task. Each problem keeps its original input format and answer format. To request a task's input, the agent reports the answers to the tasks it depends on. A tool, `get_input`, checks these reports and returns the intended input if they are correct. Otherwise, it returns an alternative input with a different verified answer. A second tool, `submit_answers`, accepts the agent's answer to the designated final task for grading. Because earlier answers are checked before an input is supplied, tasks from different domains can be connected without converting their answers into one another's input types.

The framework supports different kinds of tasks with verifiable answers. Our implementation includes math problems, predicting the output of Python code, algorithmic problems from Reinforcement Learning with Adaptive Verifiable Environments (RLVE) (Zeng et al., 2026b), and multi-hop questions from HotpotQA's distractor setting (Yang et al., 2018). Adding a task category requires examples or a generator, along with a way to check answers and provide alternative inputs with different answers. The workflows can also support reinforcement learning with verifiable rewards (RLVR), with a reward determined by the final answer. We provide this interface but do not train models in this work. The framework requires agents that can call tools.

We evaluate three models on individual problems and on instances containing 18 math or 18 Python tasks. We construct an independent-error baseline by multiplying the isolated accuracies of the final task and all tasks it depends on. Accuracy on the final task falls below this baseline, with the additional loss varying across models. We also study changes in topology, task order, and task categories. For the last comparison, we replace half the tasks in each math-only instance with tasks from other categories. After accounting for differences in the difficulty of the individual problems, GPT-OSS-120B loses accuracy. For Qwen3.8-27B and Gemma-4-31B-it, the uncertainty intervals include zero. A branch-pruning experiment further shows that agents sometimes correctly solve tasks on a branch disconnected from the requested target.

Our main contributions are:

- A method for composing problems with verifiable answers. It checks earlier answers to determine the next task's input, so incorrect reports lead to inputs with different answers. No conversion between task input and output types is needed.
- ARBIGRAPH, a framework that generates benchmark instances with chosen topologies and task pools, with tools for evaluation and training with verifiable rewards.
- An evaluation of problems in isolation and in workflows, including workflows that mix task categories or contain branches unnecessary for the target. We measure how much accuracy falls below the independent-error baseline and whether agents skip unnecessary work.

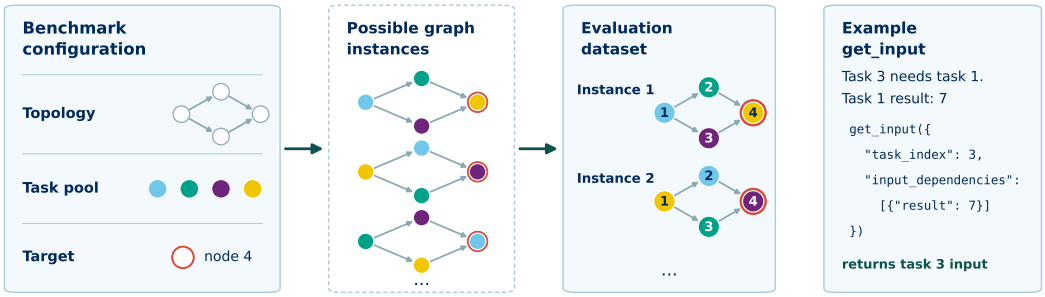

Figure 2: Framework terms and an example tool call. On the left, a benchmark configuration determines the possible graph instances. Sampling from it produces an evaluation dataset. Four colors distinguish example tasks, and the red outline marks the target. On the right, task 3 depends on task 1. The agent passes task 1's reported answer in `input_dependencies`. The tool checks it before returning an input for task 3.

2 The ArbiGraph Framework

ArbiGraph connects tasks through tool calls. Each generated *instance* is a graph with one task at each node. When the framework creates an instance, it fixes one intended input and its verified answer for each task. To request a node's input, the agent reports answers to its prerequisites via a tool call. The tool returns the intended input if all reports are correct; otherwise, it returns another valid input with a different verified answer. This lets us check intermediate answers while connected tasks keep their own input and answer formats.

The instance's *topology* specifies its nodes and dependency edges. An edge points from a prerequisite task to a task that needs its answer. One node is the *target*. The agent's answer to the target determines the score for the instance. The framework also keeps reported answers at other nodes so that earlier errors can be studied.

A *task pool* contains the tasks available for assignment to nodes. A *benchmark configuration* specifies the topology, task pool, target, and how tasks are sampled. Together, these determine the *possible graph instances*. An *evaluation dataset* contains concrete instances generated from that configuration. Figure 2 shows how these terms fit together.

2.1 Task Node

We represent the task at node i as

$$T_i = (p_i, x_i, y_i, \mathcal{B}_i, c_i),$$

where p_i is the problem statement, x_i is the *intended input*, and y_i is its verified answer. The *alternative-input bank* $\mathcal{B}_i = \{(\tilde{x}_i^{(k)}, \tilde{y}_i^{(k)})\}_k$ contains valid input–answer pairs whose verified answers differ from y_i and from one another under c_i. The *answer comparator* c_i determines whether answers match; it can require exact equality or allow a numerical tolerance. Answers can be verified by an executable solver or labels. Correctly executing the task on any bank input $\tilde{x}_i^{(k)}$ produces its verified answer $\tilde{y}_i^{(k)}$, which the comparator rejects against the intended answer: $c_i(\tilde{y}_i^{(k)}, y_i) = 0$. The agent sees p_i and the input returned by the tool, but not the stored verified answers.

2.2 Task Composition

The *parents* of node i are its direct prerequisites. We denote them by $\mathrm{pa}(i)$. To get the input for node i, the agent calls `get_input`. The tool call arguments specify the task index and a list with answers to the parent tasks. The prompt specifies the order of the parents in this list. Figure 2 shows a call for a node with one parent. We write the reported values as $\hat{\mathbf{y}}_{\mathrm{pa}(i)}$. The tool checks each value with the corresponding parent's answer comparator. It returns

$$\text{input for node } i = \begin{cases} x_i, & \text{if } c_j(\hat{y}_j, y_j) = 1 \text{ for every } j \in \mathrm{pa}(i), \\ \tilde{x}_i^{(k)}, & \text{otherwise, with } (\tilde{x}_i^{(k)}, \tilde{y}_i^{(k)}) \in \mathcal{B}_i. \end{cases}$$

Root nodes have no parents, but they still require a tool call with an empty list of answers to receive their intended input. This is also true for our single-task evaluations, which makes them directly comparable with larger topologies. The agent submits its target answer through a `submit_answers` tool.

The user provides a JSON file that lists the nodes, edges, problem types, and target. The framework uses NetworkX (Hagberg et al., 2008) to check that the dependencies contain no cycles, then assigns problems to nodes from prerequisites to dependents. It also supports visualizations using Graphviz (Ellson et al., 2004). The topology can vary the number and arrangement of dependent or irrelevant problems without rewriting the problems. Any task in the topology can be selected as the target.

2.3 Why Direct Adapters Are Insufficient

Composing problems by passing outputs as inputs requires compatible formats or an adapter to convert between them. A node with several parents also needs a rule for combining their answers. The prompt must explain each conversion to the agent, and that explanation must match the code. These requirements limit which problems can be connected and add instructions unrelated to the problems. Direct composition also makes a downstream task's input depend on upstream outputs, which can lead to difficulty accumulation. Controlling difficulty may therefore require additional constraints on these outputs or their conversion. For example, in math, Python, and other technical tasks, numeric magnitudes or sequence lengths can grow unchecked across successive tasks, requiring explicit bounds or truncation to keep downstream inputs manageable. Additionally, we observed that conversions could hide errors. For example, rounding may map a wrong parent answer to the same child input as the correct answer. This problem was especially prevalent when the child task had few possible inputs. Even without an explicit conversion, two different inputs may have the same answer. Checking the reported answers and selecting inputs that result in distinct answers avoids both cases.

2.4 Task Categories

ArbiGraph currently supports math, Python tracing, RLVE algorithmic tasks, and multi-hop reading comprehension in HotpotQA's distractor setting. Adding a category requires examples or a generator, an answer comparator, and alternative inputs with distinct answers.

Math. The category consists of 40 operators selected by hand. They cover four input-output formats: 10 list-to-list tasks, 10 list-to-scalar tasks, 10 scalar-to-scalar tasks, and 10 scalar-to-list tasks. The tasks are drawn from areas that are algorithmically concrete and exactly computable: linear algebra, polynomial arithmetic, discrete transforms, combinatorics, geometry, and number theory. Python solvers verify the answer for each sampled input.

Python tracing. The category contains 80 function/input variants drawn from the LeetCode Dataset (Xia et al., 2025). The agent traces a concrete function on concrete inputs and reports the returned value without using tools to run the code. These functions include array and list processing, sorting and searching, stack and greedy algorithms, dynamic programming, simulation, and arithmetic routines.

RLVE. The category adapts 40 algorithmic objectives from RLVE-Gym (Zeng et al., 2026b). Their underlying problems are retained, but the implementations are integrated with ArbiGraph's input sampling, executable solvers, and alternative-input banks.

HotpotQA (distractor setting). The category draws 200 distinct labeled questions from the HotpotQA training data in distractor format (Yang et al., 2018). Each input supplies the source question and all ten context paragraphs, including distractors, with paragraph titles and sentence boundaries preserved. The agent returns a short answer and supporting-fact locations in a structured result.

2.5 RLVR Interface

The same instances can be used for reinforcement learning with verifiable rewards (RLVR). An instance is sampled from a benchmark configuration or loaded from a saved dataset. For each agent attempt, the agent receives the instance's prompt and tool definitions. An accepted target

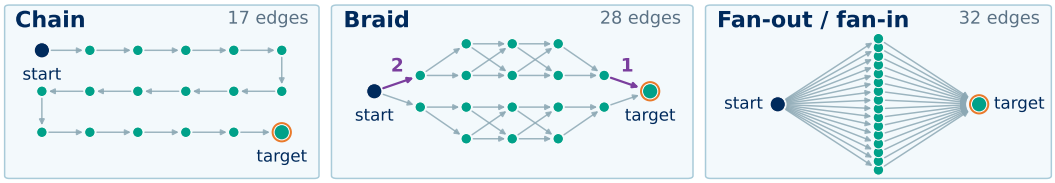

Figure 3: The three topologies with 18 nodes used in Section 4. Every node other than the target is an ancestor of the target. The *shuffled braid* (not shown) keeps the braid's problems and dependencies but reassigns task numbers. Edges 1 and 2 in the braid panel mark the cuts used for branch pruning.

answer earns a reward of one if it matches the target's verified answer, and zero otherwise. We record correctness at every node for diagnostics; future work could also use these records for reward shaping or advantage attribution.

3 Methodology

We compare an agent's accuracy on individual problems with its accuracy on the same problems inside instances with 18 nodes. Each problem keeps its statement, parameters, intended input, verified answer, and alternative-input bank in every instance that contains it, including instances where it is the only node. All instances are generated with the framework in Section 2 before inference. We call one agent attempt on an instance a *run*.

Models and Runs. We evaluate Qwen3.8-27B, GPT-OSS-120B, and Gemma-4-31B-it with temperature 1, top-$p = 0.95$, and 16,384 output tokens per turn (32,768 for continuation turns). Section A lists context limits and reasoning settings. The prompt names the target and lists all task statements and dependencies. Besides the tools from Section 2, the agent has a calculator tool for arithmetic. In our evaluation, the agent can make at most five `get_input` requests per node to limit guessing. It can correct a report on a later request, but earlier requests still count toward this limit. The harness rejects malformed submissions and lets the agent retry. It also rejects submissions before the input for the target task is obtained, which GPT-OSS-120B often attempted. After malformed tool calls, missing submissions, or truncated responses, it sends a limited number of continuation prompts. These prompts never correct a wrong value reported to `get_input`. A run that ends without an accepted target submission, times out, or exceeds the context limit is incorrect.

Answer Collection. Only an accepted `submit_answers` call supplies the graded target answer. To find where a run first goes wrong, we extract answers for every node. We take it from the `get_input` calls of the node's children, since these contain the value the agent passed on. If no child call contains it, we parse it from the agent's reasoning and response text.

Topologies. Figure 3 shows the chain, braid, and fan-out/fan-in topologies, each with 18 nodes. In each, node 1 is the only root, node 18 is the target, and every other node is an ancestor of the target. The prompt lists tasks in numerical order, and every parent has a smaller number than its children. The *shuffled braid* keeps the braid's problems and edges but renumbers the tasks, so some prerequisites are listed after the tasks that depend on them. For branch pruning (Section 4.4), we keep all 18 tasks and their inputs fixed and remove edge 1 in the braid panel of Figure 3, leaving eight tasks unnecessary for the target but sharing a root with the required branch. Removing edge 2 as well disconnects the unnecessary branch completely.

Metrics. We report target accuracy with Wilson 95% intervals. Integers must match exactly, lists are compared element by element, and floating point values must match within 10^{-3}. HotpotQA uses the official joint exact match, which requires both the answer and the supporting facts to match. If errors at different nodes were independent, an instance would succeed only when the target and all its ancestors succeed. For an instance G with target t and ancestors $\mathrm{Anc}(t)$, this gives the *independent-error baseline* $\hat{q}(G,t) = \prod_{i \in \mathrm{Anc}(t) \cup \{t\}} \hat{p}_i^{\,\mathrm{single}}$, where $\hat{p}_i^{\,\mathrm{single}}$ is the accuracy on the problem at node i over repeated runs in which it is the only node. We average $\hat{q}$ over the instances used to measure target accuracy $\hat{a}$ and report the *additional loss* $\hat{q} - \hat{a}$. We bootstrap the single-node

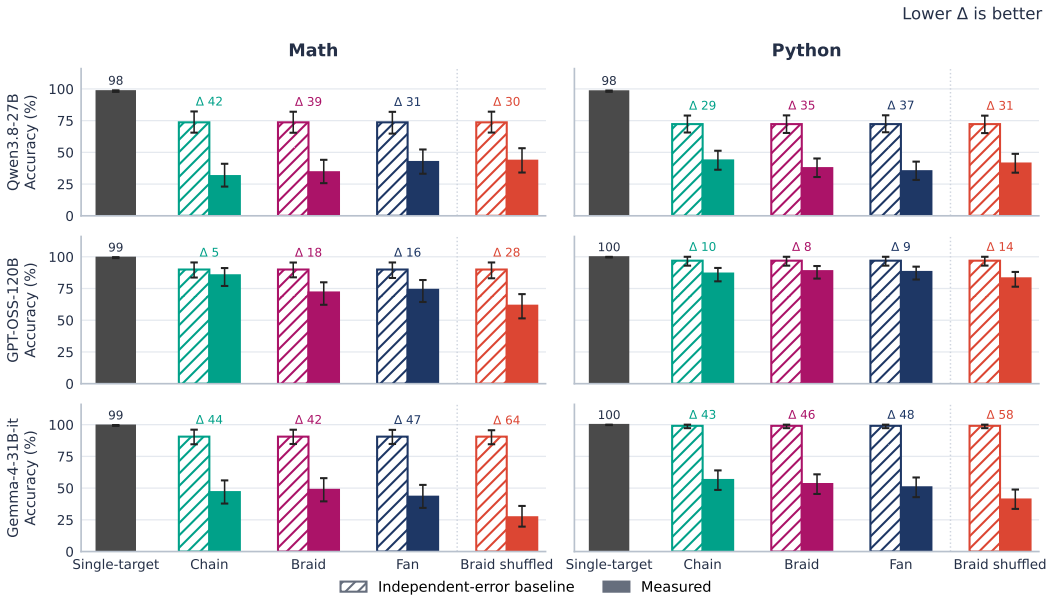

Figure 4: Accuracy on individual tasks and on targets in instances with 18 nodes. Hatched bars show the independent-error baseline, and adjacent solid bars show measured target accuracy. Δ labels give the additional loss in percentage points. Whiskers are 95% bootstrap intervals for baselines and Wilson intervals for measurements. Chain, braid, and fan differ in their edges at a fixed number of tasks. The shuffled braid reassigns the braid's task numbers without changing its problems or dependencies.

runs to estimate uncertainty in $\hat{q}$. For branch pruning, we measure the fraction of runs with correct answers for all seven non-leaf tasks on the unnecessary branch. We average the math and Python rates. To compute 95% confidence intervals, we resample target tasks within each category, treating each target's three graph instances as one group.

4 Experiments

We first run every task alone and use the results to choose tasks for the larger instances. We then measure how far accuracy on 18-node math-only and Python-only instances falls below the independent-error baseline, how topology and task order change it, and whether agents skip branches unnecessary for the target. Finally, we replace half of the tasks in the math-only instances with tasks from other categories to study the performance on mixed domains.

4.1 Individual Tasks

With seed 0, we generate eight inputs for each of the 40 math operators, 80 Python tracing variants, and 40 RLVE objectives, and add 200 questions from HotpotQA's distractor setting. Each of the 1,480 instances has one node and is run once per model. All models solve more than 91% of math, Python, and RLVE tasks (Table 1). HotpotQA accuracy is 30% to 40% under joint exact match, which also requires the supporting facts. We use these results only to choose tasks. The baselines in later sections come from separate repeated runs.

4.2 Accuracy Loss in Workflows

For each model, we keep the math operators and Python variants that it solved on all eight inputs. For each, we fix the first input and require that its alternative-input bank has distinct answers. We make this selection before any run on a larger instance. For GPT-OSS-120B, we also drop tasks whose mean response exceeds 12,000 tokens, so that instances fit its context. This leaves 33 math

Table 1: Accuracy on instances with one node (%).

Model	Math	Python	RLVE	HotpotQA
Qwen3.8-27B	97.2	93.3	91.9	40.0
GPT-OSS-120B	97.5	98.4	94.1	30.0
Gemma-4-31B-it	98.4	96.9	93.4	32.5

and 55 Python tasks for Qwen, 32 and 53 for GPT-OSS, and 37 and 52 for Gemma. We run each math task alone 50 times, and each Python task 45 times for Qwen and 30 times for the other models. For each target task and topology, we generate three instances whose other 17 nodes are different tasks of the same category. This gives 99 math and 165 Python instances per topology for Qwen, 96 and 159 for GPT-OSS, and 111 and 156 for Gemma. All topologies use the same problems, so they share one baseline. The task sets differ between models; we compare each model only with its own baseline.

Accuracy is below the baseline in every condition (Figure 4, Table 3). On Qwen's math chains, the baseline is 73.7% but accuracy is 31.3%, a loss of 42.4 points. Among chains and braids, Gemma's Python braid loses the most, 45.9 points (99.1% versus 53.2%). Without shuffling, GPT-OSS loses 4.5 to 18.1 points in math and 8.2 to 10.1 points in Python. Only the GPT-OSS math chain has an interval that includes zero. The chain and braid results match a constant drop per node of 2.8 to 4.6 points for Qwen, 3.1 to 3.6 for Gemma, and 0.3 to 1.3 for GPT-OSS.

Figure 5 shows additional loss at every node of the chain. Qwen is below its baseline at the root, while Gemma falls behind at later nodes. Outside the shuffled braid, 78% to 100% of failed instances first go wrong at a node that received its intended input. On Qwen chains, accuracy at the root, which has no parents, drops from 98.4% alone to 92.9% in math and from 97.9% to 92.1% in Python. Most failures therefore begin with a wrong solution rather than a wrong report of a parent answer. The truncation is negligible: only 4 of Qwen's 1,056 runs exceeded the context limit.

4.3 Topology and Task Order

We compute differences between topologies separately for math and Python and combine them with inverse variance weights, reporting estimates $\pm$ one standard error. Compared with the chain, the braid changes target accuracy by -2.5 ± 4.2 points for Qwen, -2.5 ± 3.1 for GPT-OSS, and -1.1 ± 4.3 for Gemma. The fan-out/fan-in topology changes it by -1.0 ± 4.2, -2.5 ± 3.1, and -4.9 ± 4.3. We find no consistent effect of the edges on target accuracy, but small effects remain possible. Compared with the braid, the shuffled braid changes target accuracy by $+5.7 \pm 4.2$ points for Qwen, -6.8 ± 3.4 for GPT-OSS, and -16.3 ± 4.2 for Gemma. Gemma loses accuracy in both categories. In math, 36% of its failures in the shuffled braid begin at a node for which it never called `get_input`, compared with 7% in the braid.

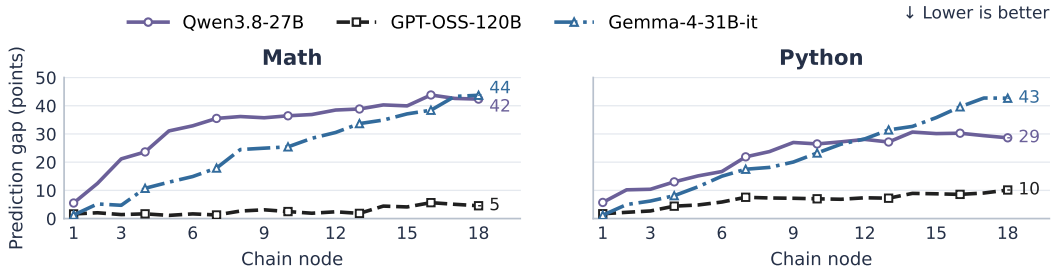

Figure 5: Additional loss along chains with 18 nodes. Each line shows the independent-error baseline minus measured correctness of the answer collected at that node, in percentage points. Math and Python share the same vertical scale; line-end labels give the target-node losses.

4.4 Pruning Unnecessary Branches

We test whether agents skip unnecessary tasks using the two braid variants in Section 3. All three models sometimes solve the disconnected branch (Table 2). GPT-OSS produces correct answers for all seven checked tasks in 83.2% of runs, compared with 87.2% when the branch is required. Qwen and Gemma do so in 20.4% and 18.3% of disconnected-branch runs, respectively. Although the branch is unnecessary for the target, GPT-OSS requests inputs for at least one of its tasks in 98.4% of runs, Qwen in 71.9%, and Gemma in 45.5%.

4.5 Instances with Mixed Categories

For each math-only instance in the chain, braid, and fan-out/fan-in topologies, we build a mixed instance with the same edges, target, and eight other math nodes. The remaining nine nodes are replaced by five Python, three RLVE, and one HotpotQA task. The RLVE and HotpotQA tasks come from a set of five RLVE tasks and three HotpotQA questions (Section A), each run alone 30 times. This gives 99, 96, and 111 pairs per topology for Qwen, GPT-OSS, and Gemma. We report the difference in target accuracy between mixed and math-only instances, both directly and after subtracting the difference in their baselines. The second value removes the effect of the replacement tasks' difficulty. We compute 95% intervals by resampling target tasks. For the adjusted difference, each bootstrap sample also draws the accuracy of each task from its Jeffreys Beta$(x + \frac{1}{2}, n - x + \frac{1}{2})$ posterior, where x of n single-node runs are correct.

The effect of mixing categories depends on the model (Figure 6, Table 4). GPT-OSS loses 21.9 points directly ($[-30.6, -13.2]$) and 16.0 points after adjustment ($[-25.3, -0.5]$). Qwen gains 7.1 points directly ($[+0.7, +13.5]$) and 6.0 after adjustment ($[-1.5, +18.3]$). Gemma changes by $+2.1$ ($[-5.7, +10.5]$) and -2.3 points ($[-7.3, +13.5]$). The RLVE and HotpotQA tasks were almost always solved alone but were solved less often inside mixed instances. No run on a mixed instance exceeded the context limit.

5 Related Work

Context and long-horizon degradation. Long-context benchmarks show that usable context is limited by distractors, lexical mismatch, and evidence position (Hsieh et al., 2024; Bai et al., 2024b; Kuratov et al., 2024; Modarressi et al., 2025; Liu et al., 2024). Multi-turn, memory, and agent benchmarks test state across interactions and growing contexts (Bai et al., 2024a; Deshpande et al., 2025; Maharana et al., 2024; Wu et al., 2025; Lee et al., 2026; Zeng et al., 2026a). Accuracy also drops when instructions are split across turns (Laban et al., 2025) and when simple steps are executed over long horizons, in part because models condition on their own earlier errors (Sinha et al., 2026). These works vary context or interaction length for one task type. ArbiGraph connects heterogeneous verifiable problems and runs each one alone and inside workflows, separating the loss caused by the workflow from the difficulty of each problem.

Composing problems for evaluation. Compositional GSM substitutes the answer of one GSM8K problem into a second and finds accuracy below the product of the two isolated accuracies (Hosseini et al., 2024); we extend this baseline to graphs of 18 problems. Relay-Bench joins subproblems from several domains into one prompt (Swayne, 2026), MuSiQue composes single-hop questions into multi-hop ones (Trivedi et al., 2022), REST and SIFo present several problems or sequential instructions at once (Pan et al., 2026; Chen et al., 2024), and Robust Reasoning Benchmark finds decay over independent AIME problems solved in sequence (Golikov et al., 2026). These bench-

Table 2: Fraction of runs with correct answers for all seven verifiable branch tasks (%). The branch is required in the full braid and unnecessary in the two pruned variants. Brackets give 95% confidence intervals.

Model	Full braid	Shared root	Disconnected [95% CI]
Gemma-4-31B-it	74.7	27.4	18.3 [13.6, 23.3]
GPT-OSS-120B	87.2	82.7	83.2 [78.5, 87.8]
Qwen3.8-27B	59.3	37.6	20.4 [15.5, 25.8]

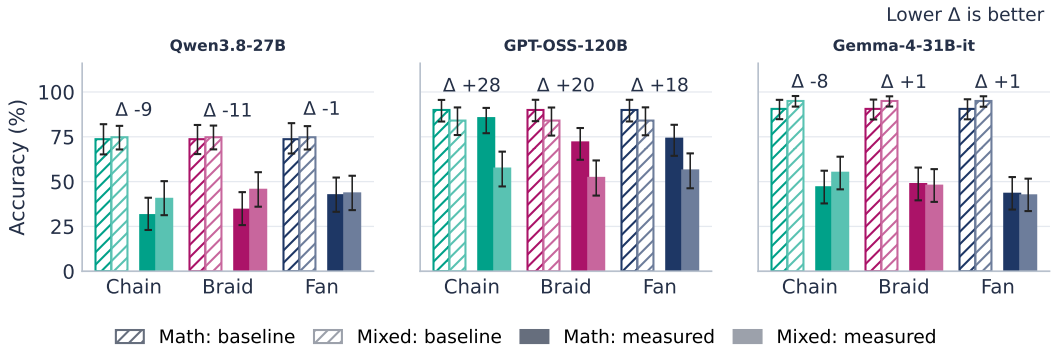

Figure 6: Instances with only math tasks and their paired instances with mixed categories. Colors identify topologies as in Figure 4; lighter shades denote mixed instances. Within each topology, the two hatched bars show the independent-error baselines, followed by the two solid bars showing measured accuracy. Δ is measured math-only accuracy minus mixed accuracy, in percentage points; lower is better. Whiskers are 95% intervals.

marks write dependencies into the problem text, so each combination is authored by hand or requires compatible formats, and the structure is fixed. ARBIGRAPH leaves each problem unchanged, checks each reported answer when the agent requests the next input, and varies topology and task order while the problems stay fixed. GSM-∞, DyVal, and Faith and Fate instead build a single problem from a graph of quantities, premises, or computation steps (Zhou et al., 2025; Zhu et al., 2024; Dziri et al., 2023), whereas our edges connect separate problems. In agent benchmarks such as NESTFUL, GAIA, and τ-bench, one tool output can become the argument of a later call (Basu et al., 2025; Mialon et al., 2024; Yao et al., 2025), but steps are not measured as standalone problems.

Composing environments for training. RACES fuses verifiable environments whose output and input types match, using composition operators (Xiang et al., 2026). Composition-RL writes an upstream answer into a downstream prompt (Xu et al., 2026), and h1 chains GSM8K problems for curriculum reinforcement learning (Ivanova et al., 2026). These methods require compatible formats or conversions between them, and a conversion can map a wrong upstream answer to the correct downstream input (Section 2.3). ARBIGRAPH instead checks parent answers to select inputs with distinct verified answers.

6 LIMITATIONS AND FUTURE WORK

We intentionally selected tasks that each model solves reliably in isolation, so that the independent-error baseline stays high and any additional loss is attributable to the workflow. Future work could vary what the context contains, such as the number of listed tasks or the retained reasoning from earlier tasks, to identify which part causes the additional loss. Context overflow was rare in our experiments. Instances that exceed the context limit, or agents with memory and summarization tools, would test whether topology matters once the agent must choose what to keep. The same verified intermediate answers can also provide rewards for each node when training with the RLVR interface.

7 CONCLUSION

ARBIGRAPH composes problems with verifiable answers into workflows without matching types or writing adapters, so tasks from different categories can be combined freely. In instances with 18 nodes, all three models lose accuracy beyond compounded single-task errors, by 36 to 45 points on average for Qwen and Gemma and 11 for GPT-OSS. With all nodes required, topology has no consistent effect on target accuracy, while shuffled task order and mixed categories hurt some models. Agents also perform unnecessary work on disconnected branches. Verified intermediate answers show where each failure begins. We hope ARBIGRAPH helps evaluate and train agents that stay reliable as their tasks become more compositional.

Reproducibility Statement

We will release the ArbiGraph code and all generated datasets upon acceptance. Section 2 describes how instances are built from a topology specification, and explains how problems are assigned to nodes and how inputs are chosen so that each child input has a distinct correct answer. All instances are generated with fixed seeds before inference. The task categories and their sources are listed in Section 2. Section 3 describes the models, sampling parameters, harness behavior (retries, continuation prompts, and failure handling), answer collection, the topologies, and the metrics, including the independent-error baseline and its bootstrap. All three models have open weights. Section A gives the remaining inference settings (context budgets, reasoning effort, top-k, and serving setup). It also lists the task-selection procedure, the IDs of the selected tasks for the mixed instances, and the held-out calibration counts. Section B reports detailed benchmark results.

AI use statement

In this work, we used generative AI tools for proofreading and polishing the writing, including rewriting full paragraphs, writing and reviewing code, rescheduling failing experiments, verifying that we did not miss prior works, and reviewing statistical practices and the correctness of conclusions. We have not used generative AI tools for writing full sections from scratch, writing the critical code paths, research ideation, or generating datasets. Math proofs are not applicable to this work. We have reviewed all AI-assisted work. We checked references and edited the writing, and all LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. MT-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7421–7454, August 2024a. doi: 10.18653/v1/2024.acl-long.401. URL https://aclanthology.org/2024.acl-long.401/.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, August 2024b. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 33538–33547, 2025. URL https://aclanthology.org/2025.emnlp-main.1702/.

Xinyi Chen, Baohao Liao, Jirui Qi, Panagiotis Eustratiadis, Christof Monz, Arianna Bisazza, and Maarten de Rijke. The SIFo benchmark: Investigating the sequential instruction following ability of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1691–1706, November 2024. doi: 10.18653/v1/2024.findings-emnlp.92. URL https://aclanthology.org/2024.findings-emnlp.92/.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E. Primack, Summer Yue, and Chen Xing. MultiChallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, July 2025. doi: 10.18653/v1/2025.findings-acl.958. URL https://aclanthology.org/2025.findings-acl.958/.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Proc. of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.18654.

John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *Graph Drawing Software*, Mathematics and Visualization, pp. 127–148. Springer, 2004. doi: 10.1007/978-3-642-18638-7_6. URL https://graphviz.org/.

Pavel Golikov, Evgenii Opryshko, Gennady Pekhimenko, and Mark C. Jeffrey. Robust reasoning benchmark, 2026. URL https://arxiv.org/abs/2604.08571. arXiv preprint arXiv:2604.08571.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proc. of the Python in Science Conference*, pp. 11–15, August 2008. doi: 10.25080/TCWV9851. URL https://networkx.org/en/.

Arian Hosseini, Alessandro Sordoni, Daniel Toyama, Aaron Courville, and Rishabh Agarwal. Not all LLM reasoners are created equal, 2024. URL https://arxiv.org/abs/2410.01748. arXiv preprint arXiv:2410.01748.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Proc. of the 1st Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2404.06654.

Alesia Ivanova, Sumeet Ramesh Motwani, Ziyang Cai, Philip Torr, Riashat Islam, Shital Shah, Christian Schroeder de Witt, and Charles London. h1: Bootstrapping LLMs to reason over longer horizons via reinforcement learning. In *Proc. of the International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=3BW15kSPfN.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Proc. International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 106519–106554, 2024. doi: 10.52202/079017-3381.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs get lost in multi-turn conversation, 2025. URL https://arxiv.org/abs/2505.06120. arXiv preprint arXiv:2505.06120.

Hyunji Lee, Justin Chih-Yao Chen, Joykirat Singh, Zaid Khan, Elias Stengel-Eskin, and Mohit Bansal. LongMINT: Evaluating memory under multi-target interference in long-horizon agent systems, 2026. URL https://arxiv.org/abs/2605.18565. arXiv preprint arXiv:2605.18565.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://doi.org/10.1162/tacl_a_00638.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, August 2024. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general ai assistants. In *Proc. of the International Conference on Learning Representations (ICLR)*, pp. 9025–9049, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/25ae35b5b1738d80f1f03a8713e405ec-Paper-Conference.pdf.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *Proc. of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Zhuoshi Pan, Qizhi Pei, Yu Li, Zinan Tang, QiYao Sun, H. Vicky Zhao, Conghui He, and Lijun Wu. REST: Stress testing large reasoning models by asking multiple problems at once. In *Proc. of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28110–28140, July 2026. doi: 10.18653/v1/2026.acl-long.1296. URL https://aclanthology.org/2026.acl-long.1296/.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in LLMs. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2509.09677.

Liam Swayne. Relay-Bench: Evaluating LLMs on multi-domain reasoning chains, 2026. URL https://arxiv.org/abs/2607.18438. arXiv preprint arXiv:2607.18438.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. URL https://aclanthology.org/2022.tacl-1.31/.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2025.

Yunhui Xia, Wei Shen, Yan Wang, Jason Klein Liu, Huifeng Sun, Siyue Wu, Jian Hu, and Xiaolong Xu. Leetcodedataset: A temporal dataset for robust evaluation and efficient training of code llms, 2025. URL https://arxiv.org/abs/2504.14655.

Hao Xiang, Qiaoyu Tang, Le Yu, Yaojie Lu, Xianpei Han, Ben He, Le Sun, Bowen Yu, Peng Wang, Hongyu Lin, and Dayiheng Liu. Verifiable environments are LEGO bricks: Recursive composition for reasoning generalization, 2026. URL https://arxiv.org/abs/2606.12373. arXiv preprint arXiv:2606.12373.

Xin Xu, Clive Bai, Kai Yang, Tianhao Chen, Yangkun Chen, Weijie Liu, Hao Chen, Yang Wang, Saiyong Yang, and Can Yang. Composition-RL: Compose your verifiable prompts for reinforcement learning of large language models, 2026. URL https://arxiv.org/abs/2602.12036. arXiv preprint arXiv:2602.12036.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Weihao Zeng, Yuzhen Huang, and Junxian He. LOCA-bench: Benchmarking language agents under controllable and extreme context growth. In *Proc. of the International Conference on Machine Learning (ICML)*, 2026a. URL https://openreview.net/forum?id=N4HIJq8v95.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. RLVE: Scaling up reinforcement learning for language models with adaptive verifiable environments. In *International Conference on Machine Learning (ICML)*, 2026b. URL https://arxiv.org/abs/2511.07317.

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. GSM-infinite: How do your LLMs behave over infinitely increasing reasoning complexity and context length? In *Proc. of the ICML Workshop on Long-Context Foundation Models*, 2025. URL https://openreview.net/forum?id=FGoZOZzotG.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. DyVal: Dynamic evaluation of large language models for reasoning tasks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=gjfOL9z5Xr.

A Experimental Setup Details

All models were served with vLLM on H100 GPUs. The workflow-study context budgets were 262k tokens for Qwen3.8-27B and Gemma-4-31B-it, and 131k for GPT-OSS-120B. Qwen used reasoning effort xhigh and top-$k = 20$; GPT-OSS used effort high, top-$k = 20$, and locally rendered Harmony prompts; Gemma used thinking mode and top-$k = 64$. Each model's repeated single-node runs used the same inference settings as its corresponding runs on instances. Results for all instances count context overflows, malformed tool calls, and missing accepted target submissions as incorrect. Problems in the instances with 18 nodes use sample index zero from the broad one-node datasets, chosen before any results from the instances were observed. The candidate task pool for mixed instances contained 24 questions from HotpotQA's distractor setting and 12 RLVE tasks. Among candidates that passed eight of eight single-target prescreen runs on all three models, we chose the three questions with the longest generated responses and five RLVE tasks nearest the math/Python output-length median after a 1,800-token minimum. The resulting IDs were HotpotQA 68, 25, and 184 and RLVE 13, 31, 38, 21, and 11. Held-out calibration then used 30 fresh repeats per task and model: 90 HotpotQA and 150 RLVE runs per model. The resulting HotpotQA/RLVE correct counts were 88/90 and 148/150 for Qwen, 87/90 and 146/150 for GPT-OSS, and 90/90 and 150/150 for Gemma.

B Detailed Benchmark Results

Table 3 reports all four conditions with one task category. The "Baseline" column gives the independent-error baseline, computed from repeated single-node accuracy of the same problems, not from the one-node results in Table 1. Each model's three structural topologies and shuffled braid use the same target problems and single-node runs.

Table 3: Single-node accuracy and target accuracy in instances with 18 nodes (%). "Fan" is fan-out/fan-in; "Shuffle" reassigns braid task numbers while keeping the problems and dependencies fixed. Except in the shuffled braid, tasks are listed in numerical order. Each condition has 99/165 Qwen, 96/159 GPT-OSS, and 111/156 Gemma math/Python instances.

Model	Category	Single-target	Baseline	Chain	Braid	Fan	Shuffle
Qwen3.8-27B	Math	98.4	73.7	31.3	34.3	42.4	43.4
	Python	98.3	72.3	43.6	37.6	35.2	41.2
GPT-OSS-120B	Math	99.4	90.0	85.4	71.9	74.0	61.5
	Python	99.8	96.9	86.8	88.7	88.1	83.0
Gemma-4-31B-it	Math	99.5	90.6	46.8	48.6	43.2	27.0
	Python	99.9	99.1	56.4	53.2	50.6	41.0

Table 4 gives the per-topology paired comparison. Its adjusted difference is the measured mixed-minus-pure difference less the change in the independent-error baseline. The paired instances retain the same math target and eight other math nodes; only the remaining nine nodes change.

Table 4: Paired target accuracy for instances with only math tasks versus mixed categories (%). Differences are mixed minus pure in percentage points and are computed before rounding. The change in the independent-error baseline from pure to mixed is $+1.1$ points for Qwen, -5.9 for GPT-OSS, and $+4.4$ for Gemma.

Model	Topology	Pure	Mixed	Direct	Adjusted
Qwen3.8-27B	Chain	31.3	40.4	$+9.1$	$+8.0$
	Braid	34.3	45.5	$+11.1$	$+10.0$
	Fan	42.4	43.4	$+1.0$	-0.1
GPT-OSS-120B	Chain	85.4	57.3	-28.1	-22.2
	Braid	71.9	52.1	-19.8	-13.9
	Fan	74.0	56.2	-17.7	-11.8
Gemma-4-31B-it	Chain	46.8	55.0	$+8.1$	$+3.7$
	Braid	48.6	47.7	-0.9	-5.3
	Fan	43.2	42.3	-0.9	-5.3