

Hindsight Structured Advice Distillation: Steering Executor Agents with Advisors Trained on Their Failures and Task Characteristics

O. Ozan Koyluoglu

Samaya AI

Language-model agents are used for specialized work such as financial research, healthcare and law. Such agents are often capable enough for a task but still miss its details. In finance, for example, these details include which filing reports a figure or how a company discloses a metric. Retraining the agent to learn these details is expensive, and often impossible when the agent is a closed or shared system. Writing the missing know-how into the agent’s prompt makes the prompt long and identical for every input, although different inputs need different guidance. We propose to learn to generate this guidance instead. A small *advisor* model reads each input and writes structured advice, which is appended to the prompt of a frozen executor. The advisor is trained with hindsight, following work on distillation with privileged information. On a training pool with graded answers, we analyse the executor’s failures, run it under several strategies to see which one worked on each input, and let a teacher that sees the grades and the traces write the advice that would have helped. The advisor then learns, by supervised fine-tuning, to write that advice from the input alone. Because the student is a separate small model rather than the executor, it is cheap to train and adds little latency. We call this approach *hindsight structured advice distillation* (HSAD). On FrontierFinance, a public benchmark of expert-written financial research queries, an 8B advisor raises the qualification rate of Samaya’s proprietary research agent, which already leads the benchmark’s leaderboard, from 52.9% to 56.6%. This is above the agent’s own high-effort configuration, at 37% lower cost per query. We study the generalization of the proposed approach, and in particular show that HSAD also improves results on WebShop, BIRD text-to-SQL and ExpertLongBench.

Correspondence: ozan@samaya.ai



1 Introduction

Language-model agents now browse websites, query databases, call tools and write research reports [1–3]. In many deployments, the model at the core of the agent cannot be changed. It may be a closed model served through an API, a production agent shared by many teams, or a small open model that is cheap to serve and costly to re-validate after every change. Improving such a system therefore usually means changing what it is told rather than its weights.

What the agent needs to be told is often know-how about the task rather than general ability. For example, a financial research agent may be asked for a company’s quarterly revenue by product category when the company publishes only each category’s percentage share of sales, not its revenue in dollars. An agent that does not know this may fill the table with its own estimates, when the better answer reports the percentages and states that dollar figures are not disclosed. A shopping agent may buy the right shirt in the wrong size, because it did not work out which option buttons the instruction required. In both cases, the agent has the knowledge and the tools it needs. It is missing a detail of how this kind of task should be done, and that detail differs from one input to the next.

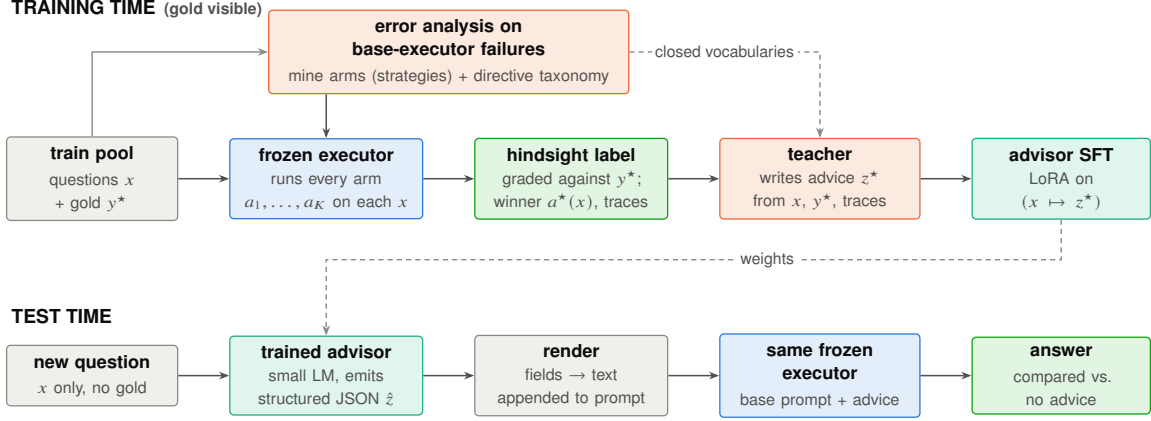


Figure 1. HSAD. At training time (top), error analysis on the frozen executor’s failures yields closed vocabularies of arms and directives. The executor is then run under every arm, and each input is labelled with the arm that won against the gold answer. A teacher that sees the gold answer and the traces writes a structured advice target, and a small advisor is fine-tuned to map the input to that target. At test time (bottom), the advisor sees only the input, and its advice is rendered as text and appended to the unchanged executor prompt.

There are two usual ways to supply such know-how. The first is to train the agent. Reinforcement learning (RL) is now a standard way to train language agents [3], but it is expensive when every rollout takes minutes and the reward is a rubric score, and it is not an option when the agent’s weights are fixed. The second is to write the know-how into the agent’s prompt, and a large body of work optimizes such prompts automatically [4–9]. However, most of these methods produce one prompt or context for a whole task. The prompt grows long as it covers more cases, and much of it applies to only some inputs. Per-input steering also exists, for example a small policy model that writes a hint for each input [10], or an advisor model trained with RL to write per-instance advice for a black-box model [11]. These methods still leave open what the advice should contain and where its training signal should come from.

We argue that hindsight can supply that signal. After the fact, on a training input with a graded answer, it is often easy to see what would have helped. The shopping episode above, for instance, shows that the agent needed to be told which size button to click. Hindsight relabeling has a long history in RL [12], in instruction following [13], and in reasoning [14]. It is also the core of learning with privileged information, where a teacher that sees information unavailable at test time trains a student that does not [15–17]. Recent work in this line distills the privileged teacher into the agent itself. We distill it into a separate small advisor instead, which leaves the agent unchanged and is cheap to train and to run. With this approach, the student must be able to infer, from its own inputs, what the teacher inferred with the privileged information, or the transfer fails [16, 18].

We call the resulting approach *hindsight structured advice distillation* (HSAD), shown in Fig. 1. At test time, we have a small trained advisor in front of a frozen executor (similar to [10, 11]), and we train the advisor by hindsight distillation. The training targets come from hindsight, what is distilled into the student model is advice, and the advisor writes that advice as a structured output (JSON) whose fields take values from closed vocabularies. First, we analyse the frozen executor’s failures on a training pool and mine two vocabularies from them: a small set of *arms*, which are whole-response strategies appended to the executor’s base prompt, and a two-level *directive taxonomy* of error areas and specific fixes. Second, we run the executor under every arm on every training input, grade the answers, and record which arm won. Third, a teacher writes the advice that would have helped most. The teacher combines an LLM, which sees the input, the grading, every arm’s answer with its score

and tool usage, the winning trace, and the closed vocabularies to choose from, with fixed rules that fill the fields that can be read off the runs or the input. Fourth, we train a small advisor by supervised fine-tuning (SFT) to write that advice from the input alone. At test time, this advice is rendered as text and appended to the executor prompt, where it acts as guidance that the executor is free to follow or ignore.

Our main setting is FrontierFinance, a public benchmark of 220 expert-written financial research queries graded against more than eleven thousand source-attributed rubric items [19]. Its report finds that the harness (tools, prompts and so on) and the executor model strongly shape quality, and that Samaya’s proprietary research agent leads all evaluated systems. We take that agent as the frozen executor and ask whether a small advisor can still improve it. On the public leaderboard, the advised agent reaches a 56.6% qualification rate, against 52.9% without advice and 56.0% for the agent’s own high-effort configuration, at 37% lower cost per query than that configuration. We then apply HSAD with a small frozen Qwen3-4B executor [20] in three further settings: WebShop [2], a simulated shopping environment; BIRD text-to-SQL [21]; and ExpertLongBench [22], a set of expert-level long-form generation tasks. On WebShop, the score rises from 54.6 to 63.4 and the success rate increases from 3.8% to 23.2% on 500 held-out samples. On BIRD and ExpertLongBench, the gains are +3.1 execution-accuracy points and +4.9 checklist-coverage points, respectively. These results suggest that the approach carries over to other executors and domains.

These four settings also show how the advice helps. Part of its value is insight learned from the executor’s traces. The vocabulary comes from error analysis of how this executor fails, and the labels come from which strategy worked on which input. On FrontierFinance, the directives mined in this way are one of the most important fields of the advice. The other part is instance-specific information that the advisor can supply and the executor tends to miss. On WebShop, copying the option values from the task into the advice as exact click targets improves performance. How much the advice helps therefore depends on the task, the training data and the executor. In our experiments, how the hindsight labels were defined had a clear effect on the trained advisor.

Contributions. (1) We propose HSAD, a supervised approach that learns from a frozen executor’s traces, through error analysis and hindsight labels, and distills what it learns into a small advisor that writes structured, input-specific guidance. (2) We make explicit where each training label comes from, and show that both the judgments of the teacher’s LLM and the labels its rules compute from the runs are useful. (3) We distinguish three reference points for how much advice can help. The first is the best strategy for each input as recorded during collection (best-arm oracle), the second is the score when the teacher’s advice, or only its arm, is given to the executor in a new run, and the third is the score with the trained advisor’s advice (the student’s performance). (4) We show that a small advisor lifts a proprietary, leaderboard-leading research agent above its own high-effort configuration on the public FrontierFinance benchmark, and we report gains in three further settings with a small open executor.

2 Related Work

Steering a frozen model with a second model. Several lines of work keep a large model fixed and train a smaller one to influence it. Directional stimulus prompting trains a small policy model, first with SFT and then with RL, to write an instance-specific hint that is added to a black-box model’s prompt [10]. Proxy tuning steers a large model at decoding time by adding the logit difference between a small tuned model and its untuned version [23], which requires access to the large model’s output distribution. Planner-executor splits divide an agent into a model that plans and a model that acts,

either within a multi-LLM tool-use agent [24] or with a separately trained planner for long-horizon web tasks [25]. Advisor models [11] train a small open-weight advisor with reinforcement learning, on rewards computed from the black-box model’s outputs, to write per-instance natural-language advice for that model, and report that an advisor trained with a low-cost student model transfers its improvements to a frontier model. Plan-and-Solve prompting asks a single model to write a plan before solving [26], and DIN-SQL decomposes text-to-SQL into schema linking, query classification and decomposition, SQL generation and self-correction steps [27]. Several of our mined arms resemble such fixed decompositions, for example BIRD’s schema-grounding and CTE-decomposition arms. Weak-to-strong generalization studies the related question of whether a weaker supervisor can elicit a stronger model’s capabilities [28]. HSAD keeps the split between a small trained advisor and a frozen executor. It differs from these methods in three ways. First, the advisor is trained by supervised fine-tuning on hindsight targets. Second, the advice is a structured object whose fields take values from closed vocabularies, rather than free text. Third, those vocabularies, and the labels for which strategy helped, come from error analysis of the executor’s own failures and task characteristics and from running the executor under different strategies utilizing the training data.

Prompt and context optimization. Automatic prompt engineering searches over instructions with a language model as the proposer [4, 5], and DSPy compiles declarative pipelines by optimizing their prompts and demonstrations [6]. TextGrad treats textual feedback as a gradient [7], and GEPA evolves prompts by reflecting on rollouts [8]. Context-engineering methods keep an evolving playbook or memory across tasks, for example ACE’s generation–reflection–curation loop [9], Dynamic Cheatsheet’s persistent test-time memory [29], and experience or workflow memories mined from past trajectories [30–32]. Self-Refine has the model critique and revise its own output [33], and Reflexion has an agent keep verbal reflections on failed trials in memory for later attempts [34]. Early experience trains an agent on the consequences of its own exploratory actions, without a reward signal, and reports gains over imitation learning across eight environments [35]. These methods optimize a context that is shared across inputs or accumulated over time. Our arms and directive taxonomy are also a mined, shared vocabulary, but the choice among them, and the values filled into them, are made per input by a trained small model.

Routing and strategy selection. Adaptive-RAG trains a small classifier to predict a question’s complexity and routes it to no retrieval, single-step retrieval, or multi-step retrieval [36]. RouteLLM and FrugalGPT route queries between models of different cost [37, 38]. The *arm* field of our advice is a router in this sense, and its labels come from running every arm, as in these works. Routing alone, however, leaves gains on the table. A routing decision helps only as much as the predicted best arm is right, and predicting it from the input alone is itself a hard problem. Even with good routing, our results show further gains from the other fields of the advice, which tell the executor what to cover and where to look.

Hindsight relabeling. Hindsight experience replay relabels a failed trajectory with the goal it actually achieved, which turns sparse-reward failures into useful training data [12]. Hindsight instruction relabeling brings the idea to instruction following by relabeling outputs with the instruction they satisfy [13]. For language agents, recent work relabels agent trajectories in hindsight to create successful demonstrations [39], and uses hindsight over explored experience to create training tasks for web agents [40]. Our labels are hindsight labels of a different kind. We keep the task fixed and relabel which strategy, and which advice, would have made the frozen executor succeed.

Privileged information and context distillation. In learning with privileged information, a teacher sees extra information during training, and a student learns to do well without it. This setting is closely related to knowledge distillation [15]. Context distillation trains a model to give, without a prompt, the answers it gave with the prompt [41, 42]. STaR shows a model the correct answer, has it write a rationale for that answer, and trains on the rationale [14]. The idea has long been used to train agents. In Learning by Cheating, a driving agent that sees the ground-truth state of the scene teaches a student that sees only camera images [43]. Several recent methods apply it to language-model agents. π -Distill trains a teacher that sees privileged information and a student that does not within one model, and uses it to distill frontier agents in multi-turn environments [44]. In LEAP, the teachers can see the environment’s hidden state [16]. In gated hindsight distillation, the teacher of a mobile GUI agent also sees the next screenshot, and distillation is applied only when the student fails and the teacher recovers the correct action [17]. Showing the teacher the answer does not always help the student. A teacher that sees a reference answer can pull its targets toward that reference [45], and in controlled comparisons the correct reference does not help consistently [46]. On-policy self-distillation trains one model as both a teacher that sees privileged information and a student that does not, on the student’s own outputs [47]. A study of the method finds that it works when the privileged information is a rule shared across inputs, such as a system prompt, and fails when that information is specific to one input, such as the answer to one problem [18].

HSAD uses the gold answer and the executor’s traces as privileged information. It differs from the methods above in two ways. The student is a separate small model rather than the agent, and it learns to write advice rather than to execute. The difference between shared rules and input-specific facts also matches what we observe. What the advisor learns from the traces, such as which directive fixes which kind of failure, is a rule shared across inputs, and the advisor learns it well. An input-specific fact helps only when the advisor can read it from the input. We therefore build the input-specific fields of the advice around information that the input states, such as the options listed in a shopping instruction. To our knowledge, HSAD is the first method to distill hindsight advice, built from a frozen executor’s own failures, into a separate small advisor that writes structured, per-input advice.

3 Methods

We first define the setting and then describe the five stages of HSAD.

3.1 Problem setup

We consider a frozen *executor* E that maps a prompt to an output, and a task distribution over inputs x . The executor may be a single model call, as in text-to-SQL, or a multi-step agent that calls tools, as in WebShop and the finance agent. Its base prompt $p_0(x)$ is fixed by the deployment. A scoring function $R(x, o) \in [0, 1]$ grades an output o against the gold answer $y^*(x)$, for example execution accuracy of a SQL query or the fraction of checklist items an answer satisfies. Gold answers exist for a training pool $\mathcal{D}_{\text{train}}$ and for held-out evaluation sets, but not at deployment.

An *advisor* A_ϕ is a small language model that reads x and emits advice $z = A_\phi(x)$. The advice is a JSON object whose fields and allowed values are fixed in advance (§3.2.3). A deterministic renderer r turns z into text, which is appended to the base prompt, and the executor runs on $p_0(x) \oplus r(z)$. We measure the method by its *paired lift* on held-out inputs,

$$\Delta = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x \in \mathcal{D}_{\text{test}}} \left[R(x, E(p_0(x) \oplus r(A_\phi(x)))) - R(x, E(p_0(x))) \right], \quad (1)$$

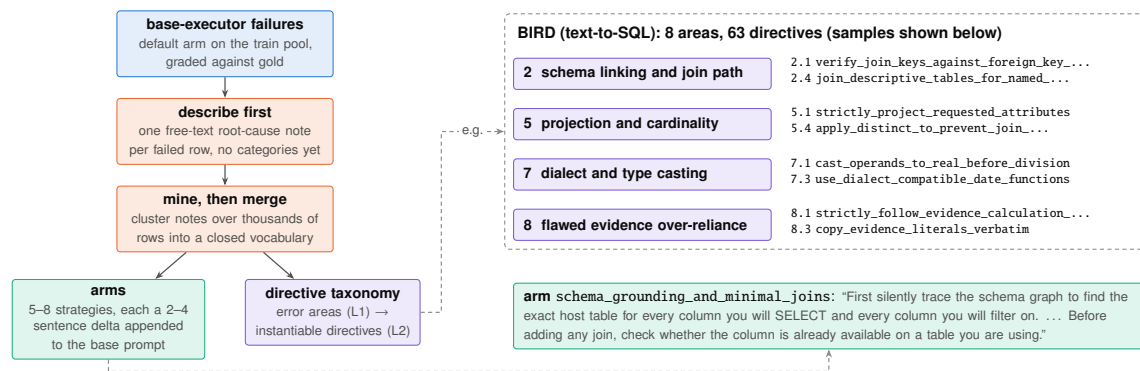


Figure 2. Error analysis on the base executor’s training failures produces two closed vocabularies. Failures are first described in free text and only then clustered. Arms are short strategy deltas appended to the base prompt. Directives are organised as error areas (L1) with specific fixes (L2). Right: four of the eight areas of BIRD’s taxonomy, with two sample directives each, and one of its arms, quoted from the arm definition.

in which the same executor answers the same input with and without advice, so that only the advice differs.

At training time, but not at test time, we have privileged information: the gold answer y^* , the executor’s traces under each strategy, and the scores those traces received. The teacher that writes advice targets may use all of it. The student advisor sees only x . This is the setting of learning with privileged information [15]. Training the student on a target is useful only if the student can produce that target from x alone, and §6 discusses which parts of the advice meet this condition in our settings.

3.2 The five stages of HSAD

HSAD has five stages. The first four run once, offline, over the full training pool: error analysis, hindsight labels, advice targets and advisor training. The fifth, rendering, runs at test time.

3.2.1 Error analysis and closed vocabularies

The first stage studies how the executor fails, so that the advice can later be written in terms of those failures. We run the executor with its base prompt over the training pool and grade each output against gold. For each failed input, a strong model writes a short free-text note on what went wrong, without being given any categories. The model sees the input, the executor’s output and its grade, together with whatever else explains the failure. For text-to-SQL, for example, it also sees the database schema and the execution error, since an error such as “no such column” often points straight to the mistake. We then cluster the notes from thousands of failed inputs into a closed vocabulary. Writing the notes before choosing any categories improves the quality of the vocabulary, since every category comes from failures the executor made on training inputs and every directive addresses one of them.

Two vocabularies come out of this stage (Fig. 2). *Arms* are three to eight whole-response strategies, each a short delta of a few sentences, plus a default arm that adds nothing. An arm must add to the base prompt rather than replace it. Otherwise an arm-versus-arm comparison mixes the strategy with incidental rewording of the base instruction. *Directives* form a two-level taxonomy with error areas at the first level and specific, named fixes at the second, for example area 7 of BIRD, “dialect and type casting”, with directive 7.1 `cast_operands_to_real_before_division` in the text-to-SQL taxonomy of Fig. 2. The size of each vocabulary depends on the task (Table 2).

WebShop "...slim fit men's pants ... with color: green, and size: xx-large, and price lower than 50.00 dollars" arm: concise_search_ and_fast_commit effort: long expected_tools: back_to_search, page_nav required_options: color: green, size: xx-large directive_picks: 3.1 select_color, 3.2 select_size	BIRD "What is the average score of the posts owned by the user csgillespie?" arm: schema_grounding_ and_minimal_joins effort: high length: medium directive_picks: 1.1 validate_attribute_ existence_and_ownership, 2.4 join_descriptive_tables_ for_named_entities, 8.1 strictly_follow_evidence_ calculation_formulas, 8.3 copy_evidence_literals_ verbatim	ExpertLongBench an agent trajectory to audit for security threats (task T11CyberRDG) arm: security_threat_ frame_first target_length: short (<300 words) effort: low directive_picks: 1.5 analyze_security_ trajectory_not_surface..., 2.5 detect_adversarial_ content_in_tool_returns, 3.10 populate_all_security_ analysis_fields, 8.1 detect_and_label_ prompt_injection	FrontierFinance Colgate-Palmolive revenue by geography and product, each quarter since 2019 arm: default target_length: long (3+ pages) web_use: avoid sources: FILINGS, ... tools: documents_list, documents_search, ... directive_picks: 2.2 distinguish_metric_ unavailability_from_zero, 1.1 enumerate_all_periods_ in_requested_range, ...
---	---	---	---

Figure 3. Example advice JSON in the four domains. Table 1 lists where the training label of each field comes from. Directive identifiers refer to the domain’s taxonomy (Fig. 2).

3.2.2 Hindsight labels from running every arm

The second stage produces the hindsight label. We run the frozen executor under every arm on every training input, grade every output against gold, and record the winning arm $a^*(x)$ together with its trace. Some rows produce ties, because several arms often give the same score, and the tie-break is part of the label definition. We break ties toward default. If no arm beats doing nothing, the correct advice is to change nothing, and training the advisor to fire an arm on such rows teaches it to intervene without benefit. When scores have several dimensions, ties can be broken by a fixed fallback order of secondary criteria. If every arm fails, or several arms tie and the scores cannot separate them, the teacher’s LLM, which sees the gold answer and every arm’s output, can pick the arm whose output is closest to correct. Section 4 gives the scheme used in each setting, and Appendix C.1 gives examples of how the tie-break changes the advisor.

3.2.3 Advice targets and the field-source rule

The third stage writes the advice target $z^*(x)$ for every training input. This is the signal that the advisor will learn to produce. The *teacher* is the labelling procedure that has hindsight access to gold and to the runs. It combines an LLM, for fields that need judgment, with fixed rules, for fields whose value can be read off the arm scores, the winner’s trace or the input. The teacher’s LLM receives the input and its grading material (the gold answer or a grading rubric), every arm’s output with its score and tool usage, and the full trace of the winning arm. It also receives the closed vocabularies it must choose from: the arms, the directive taxonomy and, where relevant, the values of other fields (e.g., the list of data sources). It then fills a domain-specific JSON schema whose fields are illustrated in Fig. 3 and listed in Table 1. The schema is designed before any LLM call, and the LLM does not fill every field. The quality of this schema matters a great deal for the results. For a new task, iterations on failure analysis and on alternative schemas therefore pay off.

Where each label comes from. Every training label has two properties: how the teacher computes it and what it is computed from. It is computed either by one of the teacher’s fixed rules (*rule-based*) or by the teacher’s LLM (*LLM reasoning*). It is computed from the input alone, from the gold answer, or from the executor’s runs. From the runs, a rule can read either the arm scores, meaning the grader’s score of every arm’s output against gold, or the winner’s trace, meaning the run of the winning arm. Everything except the input is available only in hindsight. Table 1 gives both properties for every field we use.

Domain	Field	Training label			At test time produced by
		computed by	computed from	how	
WebShop	arm effort, expected_tools required_options, directive_picks	rule-based	arm scores	highest-scoring arm; ties to default	advisor
		rule-based	winner’s trace	winner’s tool calls	advisor
		rule-based	input	options stated in the instruction	advisor
FrontierFinance	arm target_length, web_use, sources, tools directive_picks	rule-based	arm scores	highest-scoring arm; cascade tie-break	advisor
		LLM reasoning	gold + all runs	reads all arms’ answers, grades	advisor
		LLM reasoning	gold + all runs	103 directives	advisor
BIRD	arm effort, length directive_picks	LLM reasoning	gold + all arms’ queries, results	arm closest to gold	advisor
		rule-based	winner’s trace	winning SQL’s structure, length	advisor
		LLM reasoning	gold + all runs	63 directives	advisor
ExpertLongBench	arm target_length, effort directive_picks	rule-based	arm scores	highest-scoring arm; ties to default	advisor
		rule-based	winner’s trace	winner’s length	advisor
		LLM reasoning	gold + all runs	57 directives	advisor

Table 1. Source of each advice field’s training label: how the teacher computes it (rule-based or by LLM reasoning) and what it is computed from. “Arm scores” are the grader’s scores of every arm’s output against gold; “winner’s trace” is the run of the winning arm. The last column shows what produces each field at test time.

Rules where possible, the LLM where necessary. The same field can often be obtained in either way. Answer length, for example, can be computed by a rule from the winner’s trace or estimated by the teacher’s LLM. We use a rule whenever a field’s value can be read off the runs, the gold answer or the input, and we use the teacher’s LLM only for fields that require judgment. We adopted this rule after finding that the LLM’s estimates can disagree with the runs (§5.4). Effort and length fields are therefore coarse buckets of the winning trace’s tool-call count, output complexity or answer length. The arm is always the hindsight winner, and the LLM picks it only when the scores cannot separate the arms.

Training labels and test time. The source of a training label is separate from how the field is produced at test time. A label computed by a rule on the runs, such as the highest-scoring arm, still has to be predicted by the advisor, because the runs it was computed from exist only after the executor has run. In all four settings the trained advisor predicts every field from the input alone (last column of Table 1). Only a rule on the input could in principle replace the advisor for a field, as for the option fields on WebShop (§5.2).

Leakage. The teacher’s LLM sees the gold answer, so the advice it writes must not leak that answer, for example by naming the value the answer should contain. We prevent this in how the vocabularies are built and used. When we mine a directive taxonomy, the prompt requires every directive to be a generic behaviour that applies across many inputs, with no entity, date, period or value in it, and candidates that name one are rewritten or dropped. When the teacher’s LLM writes an advice target, it only picks among these fixed values, and the executor receives the fixed description of each picked directive. The advice therefore contains no free text written with the gold answer in view.

3.2.4 Advisor training and selection

The fourth stage trains the advisor by supervised fine-tuning. Each training example pairs an input with the teacher’s advice target for it. The advisor is a small open model that learns to produce the JSON target from the input and a system prompt that lists the schema and vocabularies. It never sees gold. Because it is small, it is cheap to train, and running it adds one low latency model call per input. We select the epoch by paired lift on a held-out validation split, running the executor with each checkpoint’s advice, rather than by the agreement between the advisor’s JSON and the targets (see

Setting	Frozen executor	Metric	Train	Test	Arms	Directives
FrontierFinance	Samaya production agent	rubric qualification rate	1,819	220	4	103
WebShop	Qwen3-4B agent	score (0–100), success rate	6,210	500	6	6
BIRD	Qwen3-4B	execution accuracy	9,421	1,074	9	63
ExpertLongBench	Qwen3-4B	checklist coverage	450	100	8	57

Table 2. The four settings. “Train” is the number of advisor training rows. Arms include default.

Appendix C.1 for more details).

3.2.5 Rendering

The fifth stage turns the advice into text for the executor. Each field has a fixed template. The arm contributes its strategy delta, effort and length become one sentence each (“Plan for 6 or more tool calls in total.”), and each directive contributes its one-sentence description. The rendered text is appended to the executor’s prompt or message list, as guidance that the executor is free to follow or ignore.

Instantiated directives. A directive can be generic (“select every required option before buying”) or *instantiated* with values from the input (“Before clicking Buy Now you **MUST** click the size button labelled ‘xx-large’”). A generic directive asks the executor to work out what to do, whereas an instantiated one is more specific.

3.3 Performance comparisons

To judge how much of the possible gain the trained advisor captures, we compare it with reference points that use hindsight. Studies on advice and routing often call such a reference an “oracle”, but the term is used for different quantities, so we define ours here.

1. The *best-arm oracle* is the highest score that any arm received during collection, averaged over the dataset. Like the other two reference points, we report it as a lift over the default arm. It bounds the gain from choosing the arm, but not the gain from the full advice.
2. The *teacher-executed ceiling* runs the executor again on held-out inputs with the teacher’s hindsight advice. It leaves out the advisor’s errors in predicting this advice target.
3. The *student* is the trained advisor’s advice, predicted from the input alone and executed. It measures the deployable performance, as the lift over the executor without advice.

4 Experimental Setup

We evaluate HSAD in four settings that differ in the executor, the kind of output and the scoring function. Our main setting is FrontierFinance, where the executor is a production research agent and the goal is to improve on a strong proprietary system. The other three settings use a small open executor and test whether the method carries over to shopping, text-to-SQL and expert long-form generation. Table 2 summarizes them, and Appendix C.1 gives further details of the experimental setup.

FrontierFinance. FrontierFinance is a public benchmark of 220 expert-written financial research queries, graded against 11,543 source-attributed rubric items that span six use cases of an investor’s workflow, from earnings and events to screening and discovery [19]. Each query has its own rubric, and some rubric items are marked essential. The metric is the qualification rate, the fraction of rubric items an answer satisfies, averaged over queries. We report it for all items, as the public leaderboard does, and separately for essential items. Our executor is Samaya’s production research agent, which leads the benchmark’s leaderboard [19], with tools over financial filings, market-data services and the web. We freeze it and try to improve it only through advice. Its four arms are `default`, `structured_priority` (prefer structured and tabular data tools), `tables_extraction` (extract figures from filing tables), and `restricted_tools` (structured sources only, no web search). Ties between arms are broken by a fallback order: essential criteria, then all criteria, then cost, then latency. The advisor is trained on 1,819 questions from the same expert-annotated pool that FrontierFinance was drawn from, none of which is among the 220 public queries. It is evaluated on a separate held-out validation set and on the 220 public queries, which serve as the test set.

WebShop. WebShop is a simulated e-commerce website in which an agent receives a shopping instruction, such as this one from its test set: “slim fit men’s pants with elastic waist . . . with color: green, and size: xx-large, and price lower than 50.00 dollars”, and must search, open products, select options and buy [2]. The environment’s reward gives partial credit for the product type, its attributes, the selected options and the price. We report it as a score from 0 to 100, together with the success rate, the fraction of episodes with a perfect reward. The executor is a frozen Qwen3-4B agent that emits one JSON action per turn, with a 15-turn cap and a fixed environment seed. The training pool has 6,410 goals (samples of the dataset), of which 200 are held out for epoch selection, and the test set has 500 samples.

BIRD. BIRD is a text-to-SQL benchmark of 12,751 question–SQL pairs over 95 databases [21]. Each question comes with an “evidence” hint that states domain knowledge, for example that “Italian refers to language = ‘Italian’”. The metric is execution accuracy, the fraction of questions whose predicted query returns the same result as the gold query. The executor is a frozen Qwen3-4B that writes one query per question. The training pool is BIRD’s training split (9,428 questions over 69 databases). We split BIRD’s development set into 460 questions for epoch selection and 1,074 for test. Because many training questions have several correct arms or none, the teacher’s LLM, which sees the gold query and every arm’s query and result, picks the arm closest to correct.

ExpertLongBench. ExpertLongBench is a benchmark of expert-level long-form tasks, each with a rubric designed or validated by domain experts [22]. Outputs are graded by extracting a checklist of the rubric’s items from the output and comparing it with the checklist extracted from a reference. We use six of its public tasks: materials-science synthesis extraction (T03), pedagogical evaluation of tutoring dialogues (T04), clinical note generation (T06), molecule and protein description generation (T07, T08) and security review of agent trajectories (T11). We exclude the seventh public task (legal multi-document summarization), whose source documents have a median length of about 62K tokens. The metric is the fraction of checklist fields an answer satisfies. The executor is a frozen Qwen3-4B. It has 450 training and 100 test documents.

5 Results

We report results on FrontierFinance, our main setting, and then on WebShop, BIRD and ExpertLongBench. The advice improves the executor in all four settings (Tables 3 and 5). For each setting, we

System (harness)	Qualification rate	Cost / query	Latency / query
Samaya System + advisor	56.6%	\$1.14	257 s
Samaya System, high effort	56.0%	\$1.81	278 s
Claude Fable 5.1 (Finance Agent v2)	55.9%	\$6.89	268 s
Samaya System (no advice)	52.9%	\$0.93	214 s
Claude Fable 5 (Finance Agent v2)	49.2%	\$4.06	165 s
GPT 5.6 Sol (Finance Agent v2)	46.8%	\$3.03	171 s
Kimi K3 (Finance Agent v2)	46.4%	\$0.90	336 s
Gemini 3.6 Flash (Finance Agent v2)	46.3%	\$2.41	164 s
Claude Opus 4.8 (Finance Agent v2)	45.0%	\$2.61	156 s

Table 3. FrontierFinance public leaderboard (220 queries, all rubric items) with the advised agent added. All rows except the advised agent are taken from the public FrontierFinance leaderboard [19]. The two Samaya rows from the leaderboard use the same proprietary harness we use, the other rows use the open-source Finance Agent v2 harness.

also examine which advice fields contribute to the gain, and give additional analyses in Appendix C.

5.1 FrontierFinance

The finance setting asks whether a small advisor can still improve a system that is already strong. The executor leads the benchmark’s leaderboard, and its harness was built for this kind of research, so any gain has to come from advice that the agent does not already give itself. Each run of the agent takes about four minutes and costs about a dollar, so the arms collection over 1,819 training questions is the most expensive step of the pipeline.

The advice has six fields (Fig. 3). The arm is the hindsight winner strategy. The teacher’s LLM writes `target_length` (short, medium or long, with fixed page counts for each category), `web_use` (avoid, fallback or required), a prioritized list of data sources, a list of suggested tools, and `directive_picks` drawn from a 103-directive taxonomy in 9 areas (for example 3.1 `cover_all_named_entities_in_query`). To write these fields, the LLM sees each training question’s rubric, the four arms’ answers with their rubric scores, tool calls and costs, and the tool-call trace of the winning arm. The rendered advice is appended to the query as a guidance block (Appendix A.2). It is phrased as a suggestion, with an explicit fallback to the agent’s own exploration when needed.

Table 3 compares the performance of the advised agent against the leaderboard’s entries. The leaderboard’s Samaya System entry is the same agent without advice. With the advisor, the agent’s qualification rate rises from 52.9% to 56.6%. This is above the Samaya System’s high-effort configuration (56.0%), which adds improved tools, stronger context management and more reasoning effort, and above the best frontier model in the open finance-agent harness (Claude Fable 5.1, 55.9%). The advised agent costs \$1.14 per query, against \$0.93 without advice, \$1.81 for the high-effort configuration and \$6.89 for Claude Fable 5.1. Fig. 4 shows the same comparison as a cost–quality trade-off. The advisor moves the Samaya System up to the top of the leaderboard while staying in the low-cost region, where every other system of similar cost, apart from the unadvised agent itself, scores at least ten points lower. More details are included in Appendix C.2.

To find out which fields carry the gain, we ran a leave-one-field-out ablation on the validation set. The ablation uses the teacher’s own advice rather than the student’s, so that each run isolates one field without prediction errors. Table 4 reports the results. With all fields, the teacher’s advice scores

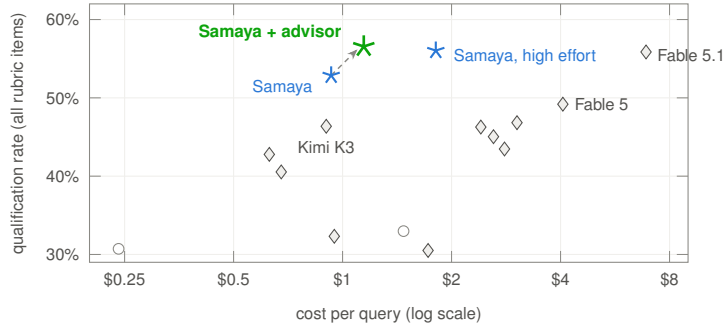


Figure 4. Qualification rate against cost per query on the FrontierFinance public leaderboard (220 queries). Stars are the proprietary Samaya harness, diamonds the open-source Finance Agent v2 harness with different frontier and open-weight models, and circles a web-search-only harness. The green star is the Samaya System with the trained advisor; the dashed arrow connects it to the same system without advice.

69.3% essential against 63.0% without advice, a teacher-executed gain of 6.3pp that the trained advisor matches (6.2pp). Dropping the directive picks removes the whole gain. Every other field contributes a smaller amount, between 1.8pp and 4.4pp. The ablation was run while the schema still had an effort field. Removing it slightly improved the score, so the final schema does not include it.

Teacher advice	essential	all	Contribution (essential)
all fields	69.3%	59.6%	–
without directive_picks	62.6%	54.3%	6.7pp
without sources	64.9%	55.5%	4.4pp
without tools plan	66.3%	56.2%	3.0pp
without web_use	66.6%	57.0%	2.7pp
without target_length	67.5%	57.5%	1.8pp
without effort	70.0%	59.0%	−0.7pp
no advice	63.0%	53.9%	–

Table 4. Leave-one-field-out ablation of the teacher’s advice on the FrontierFinance validation set. The contribution of a field is the drop in essential-criteria coverage when it is removed.

The directives are general guidance, for example to give every named company equal depth of coverage or to pull sub-segment detail within reported segments. Such guidance is the main source of the gain for this finance task. In Appendix D, we show two test queries with the advisor’s output and its effect on the answer. On the training side, further variants, namely more epochs, a reasoning trace before the JSON and RL, did not improve performance, as we discuss in Appendix C.2.

Taken together, these results show that an 8B advisor trained on hindsight labels lifts Samaya’s production agent from 52.9% to 56.6% on the public FrontierFinance leaderboard, above the agent’s own high-effort configuration at 37% lower cost per query.

5.2 WebShop

On WebShop, the trained advisor raises the executor’s score from 54.6 to 63.4 and its success rate from 3.8% to 23.2% (Table 5). We reached this design in several steps (Appendix C.3). Advisors that only chose the arm gained at most 3.65 points, and adding generic directives initially lowered the

Setting	Metric	No advice	With advice	Lift
WebShop	score / success rate	54.62 / 3.8%	63.35 / 23.2%	+8.73 / +19.4
BIRD	execution accuracy	40.04%	43.11%	+3.07
ExpertLongBench	checklist coverage	49.7%	54.5%	+4.9

Table 5. Held-out lift of the trained advisor with a frozen Qwen3-4B executor. Each row compares the same executor on the same inputs with and without advice. Success-rate lifts are in percentage points.

score before further optimizations.

To find what the arms were missing, we measured where the lost reward went. Across the 5,927 training samples where even the best arm fell short of full reward, 69% of the lost reward came from missed options, 18% from attributes, 10% from product type and 3% from search or commitment failures. The executor could search and it could click, but it often did not work out which option buttons the instruction required. A generic directive, “inspect all option buttons and click every required one”, did not help. An instantiated one did. Parsing the option values from the instruction and rendering them as “Before clicking Buy Now you MUST click the size button labelled ‘x-large’; . . .” raised score by 8.9 points on validation. The option values are stated verbatim in the instruction, and a regular expression recovers these values for every sample. Rendering the values as click instructions removes the extraction step, and the executor only has to click the named buttons.

The final advisor emits the schema given in Fig. 3. All of its training labels come from the teacher’s rules, without any LLM. The arm, effort and expected tools come from the scored runs, and the options and their directive IDs are parsed from the instruction, where they are stated verbatim and match the gold goal. On 200 validation samples the advisor’s `required_options` matched the instruction’s option values on every sample. On test, its full advice gains 8.73 score points. The option values could also be filled in by a rule, but the arm, effort and tool fields come from hindsight and must be learned; with them, the success rate rises from 3.8% to 23.2%, against 19.4% with the options alone.

Our test set is WebShop’s standard 500 test instructions, the same set used by ReAct [1], so the scores are comparable, although the agent harnesses differ. With advice, the Qwen3-4B agent’s score (63.4) moves above the imitation- and reinforcement-learning agent of the original paper (62.4) [2] and stays below ReAct prompting with a 540B model (66.6) [1]. Its success rate, 23.2% against ReAct’s 40.0%, is still much lower, so the agent with a small executor more often buys products that match the instruction partly rather than exactly.

5.3 BIRD

On BIRD, advice must steer a single SQL generation, and the most useful knowledge is which tables, columns and joins the question needs. This is harder for an advisor than WebShop’s options, because it is not stated in the question, and in our setup the BIRD advisor sees only the question and its evidence hint, not the database schema.

The trained advisor raises execution accuracy from 40.04% to 43.11% on the 1,074 test questions. Appendix C.4 compares the result against the three reference points. The best arm recorded during collection would add 20.11 points, running each question again with its best arm adds 11.27, and the student captures about 27% of the executed ceiling. By the advisor’s chosen arm, the largest gains are on `schema_grounding_and_minimal_joins` (+7.98 on 188 questions) and `strict_projection_and_`

output_alignment (+13.16 on 38). One example query is the following: “What is the average score of the posts owned by the user csgillespie?” The executor without advice filters on a denormalized display-name column of the posts table and returns the wrong answer. With the advice of Fig. 3 (schema grounding, “join descriptive tables for named entities”) it joins the users table on the owner’s identifier and filters on users.DisplayName, which matches the gold result. Another pattern is dialect. Given “how many badges . . . were obtained in the year 2011”, the executor wrote YEAR(Date) = 2011, which SQLite does not support, and the advice “use dialect-compatible date functions” led it to strftime('%Y', Date) = '2011'.

We also analysed the advisor’s own errors on the test set (Appendix C.4). When the advice hurts, it is usually because the executor applies a directive where it does not fit, for example by adding an unneeded join. When the executor answers a question wrongly both with and without advice, an existing directive already covers the root cause in 94% of the cases we inspected. The remaining gap therefore lies mostly in choosing and applying directives, and few failures point to a missing strategy. The advised executor can be compared with published work, although published BIRD results use the full development set or the hidden test set, while ours use a 1,074-question subset of the development set, so comparisons are approximate. The advised Qwen3-4B executor performs comparably to what BIRD’s authors report for ChatGPT on the test set with external knowledge (39.3%, or 40.1% with chain-of-thought prompting) [21], and below DIN-SQL (55.9% on the test set) [27], which prompts GPT-4 to solve the task in separate steps (schema linking, query classification and SQL generation) and then to correct its own query.

The advice thus helps on BIRD, mainly by grounding questions in the schema (Appendix C.4), but the student captures only about a quarter of the executed ceiling. These analyses show that there is room for improvement with further iterations.

5.4 ExpertLongBench

ExpertLongBench tests the method where data is scarce and outputs are long. The advisor has only 450 training documents across six tasks, and each output is a structured report of several hundred words.

Error analysis produced eight arms, several of which are task-specific frames such as security_threat_frame_first, pedagogical_evaluator_role_lock and clinical_note_completeness_and_fidelity, and a 57-directive taxonomy in nine areas. The areas cover task interpretation, source grounding, field completeness, value accuracy, schema-field disambiguation, negative findings, output structure, security analysis and pedagogical evaluation. Example directives are 2.2 scan_entire_source_for_each_field, 6.1 record_absent_fields_explicitly and 8.1 detect_and_label_prompt_injection. The best-arm oracle over the eight arms is +9.5 points on the training pool. The target_length labels come from the winning answer’s length. We first let the teacher’s LLM estimate this field, but its estimate matched the length of the winning answer on only 149 of 450 training rows (33%), and it systematically called for longer answers than the winning arms produced. Replacing the estimates with the winner’s actual length bucket changed 301 of the 450 labels.

Trained on the vocabularies and labels described above, the advisor raises checklist coverage from 49.7% to 54.5% on the 100 test documents. We analyse the distribution of the lift over tasks, together with the advisor’s arm and directive choices for each task, in Appendix C.5. In short, the advisor picks directives that fit each task but sometimes chooses an arm meant for a different task. With only about 75 training documents per task, this is not surprising. We expect more training data, and an error analysis done separately for each task, will improve the arm choices and raise the gain further.

6 Discussion

Types of information in advice. The advice carries two kinds of information. The first is insight learned from the executor’s traces. Error analysis on the training pool shows how this executor fails and which strategies fix each failure, and the hindsight labels show which fix actually worked on which input. The advisor turns this into a per-input choice from the arms and directives. This kind of information is not in the input, and the advisor could not have produced it without seeing the traces during training. All three settings that use mined directives show this. On FrontierFinance, the directives distilled from the agent’s failed and successful traces are what the gain depends on: without them, the teacher’s advice no longer improves on no advice (Table 4). On ExpertLongBench, the same kind of directives lets the advisor pick guidance that fits each task, and on BIRD, a mined directive already addresses the root cause of 94% of the remaining failures.

The second kind is instance-specific values filled into the advice. WebShop’s option values are stated in the instruction, so the advisor copies this field without error, while the arm, effort and tool fields are what it learns from hindsight. On BIRD, the most useful values are the tables and columns a query needs. Our advisor does not see the database schema, so it has to infer them from the question, which is the same inference the executor gets wrong, and the student captures only about a quarter of the executed gain. This matches two observations from prior work on privileged information. A student can only learn what a teacher infers from privileged information if it can recover the same thing from its own input [16], and on-policy self-distillation works when the privileged information is a shared rule, such as a system prompt, but fails in the settings tested when it is specific to each instance and absent at test time [18]. In our setting, the insight from the traces plays the role of shared rules, and the filled-in values are instance-specific facts, which the advisor can supply only when the input contains them.

Specific instructions work better than guidance for small executors. In both Qwen3-4B settings, advice that tells the executor exactly what to do helped more than advice that asks it to reason more carefully. On WebShop, “click the size button labelled ‘xx-large’ ” gained almost 9 points, while “inspect all option buttons” did not help and “verify the product type” hurt. On BIRD, most of the harm came from directives that the executor applied where they did not fit. One reading is that a small executor does not have spare capacity to decide when a general piece of guidance applies, so general guidance mostly adds text to its prompt, whereas a specific instruction replaces a decision it would otherwise get wrong. The finance agent, whose executor is a large frontier model, is the exception. Its gain comes mostly from general directives such as “cover all named entities in the query” (§5.1), which is consistent with the same reading, since a stronger executor can decide what to do with such guidance.

Label quality matters. In our experiments, the way the training labels were defined had a clear effect on the trained advisor. On FrontierFinance, the cascade rule for the winning arm changed 681 of the 1,819 labels and gave our best advisor. More epochs, a reasoning trace and RL did not improve on it (Appendix C.2). On WebShop, breaking ties toward default gave a better advisor than breaking them toward the arm with the fewest turns (Appendix C.1). On ExpertLongBench, the teacher’s LLM estimated the answer length correctly for only a third of the training samples. An advisor trained on these estimates would have learned mostly the LLM’s bias. All three cases follow the field-source rule of §3.2.3. Some fields have a single correct value that a rule can compute, such as the answer length or the options stated in the instruction. For these fields, the rule gives more reliable labels than the LLM. Other fields, such as the directives, require judgment across several runs. For these, the LLM’s labels carry the gain, as the FrontierFinance ablation shows.

Advice quality depends on the data and the task. The same pipeline gave gains of different size in the four settings. The differences come mainly from the training data and the task. The advisor can only be as good as its labels and as broad as its training pool. On ExpertLongBench, for example, 450 training documents were enough to learn task-appropriate directives but not enough to learn a reliable arm choice. The advice also helps only where the executor’s traces show failures that advice can fix. An instantiated field helps only if its value can be recovered from the input. WebShop’s option values can be recovered this way, but BIRD’s tables and columns cannot. Before applying the method to a new setting, we suggest checking four things. The executor should succeed on some training inputs and fail on others. The training pool should be large and reliably graded. Error analysis should reveal failures that advice can address. Finally, any instance-specific values that the advice needs should be recoverable from the input.

Advice as guidance. The advice is appended to the executor’s instructions as guidance. The executor keeps all of its tools and can depart from the advice when the evidence points elsewhere. Not every suggestion is followed, but on average the advice helps. The fields also differ in how closely they are tied to one executor. Some fields describe the input and what a good answer needs, such as which sources hold the answer, how many periods it spans or which disclosures to check. These fields should carry over to other executors. Other fields reflect the habits of the executor whose traces the advisor learned from, such as its typical blind spots. These fields are most useful for that executor.

Advice in the prompt rather than in the weights. Our advice is text, and the advisor acts as a small adapter that supplies task know-how for each input. A single optimized prompt gives every input the same guidance. The advisor instead gives each input its own guidance, without making every prompt longer. The executor never changes, so the method works with closed or shared models, and the same advice can be given to a different executor. The cost is that the executor reads the advice on every call and does the extra work it asks for. On the FrontierFinance leaderboard, the advised agent costs 23% more per query than the agent without advice, but it still costs 37% less than the agent’s high-effort configuration and scores higher, which shows the value of guidance tailored to each input. Methods that distil experience or skills into the executor’s weights [48, 49], or that have a side model produce parameters or latent tokens instead of text [50–52], avoid this per-call cost, but they have to modify the executor. Advice written as text also has an interpretability benefit, since the advice given for each input can be read and inspected directly.

7 Limitations

Statistical power. Each result comes from a single evaluation run per condition. Most of our test sets are large enough that the averages over samples are reasonably stable: 220 queries on FrontierFinance, 500 instructions on WebShop and 1,074 questions on BIRD. The exception is ExpertLongBench, with 100 test documents. Our validation sets are also small, with 100 questions on FrontierFinance and 200 samples on WebShop, so differences of a few points between advisors on validation can be within run-to-run variation. Repeated runs with different seeds would give confidence intervals for every comparison, and we leave this to future work.

Iterations. The arms and the directive taxonomies went through several rounds of revision, each guided by error analysis of the executor’s traces, and most rounds improved the result. On WebShop, rebuilding the taxonomy around instantiated option directives raised the gain from about 3 to 8.7 points. On BIRD, appending the arms to a fixed base instruction improved every arm, and on FrontierFinance, revising the winning-arm rule improved the advisor. These revisions were not

exhaustive. Further error analysis and iterations would likely give further gains.

Scope. We studied four settings, and the results may not hold for other tasks, executors or datasets. In particular, we have not measured how well the advice transfers to other executors or to other datasets in the same domain. More generally, the quality of the structured advice and of the trained advisor depends on the training data and on the task. Some tasks may be hard to learn with an advisor. When the most useful advice requires the same inference that the executor gets wrong, as with BIRD’s choice of tables and columns, the advisor gains little, and training the executor itself may be necessary. The method also needs a training pool with gold answers or a reliable grader, an executor that succeeds on some training inputs and fails on others, and an arms collection phase whose cost grows with the number of arms.

8 Conclusion

We described hindsight structured advice distillation (HSAD), a way to improve a frozen executor by training a small advisor on the executor’s own failures. The motivation is that a capable executor often lacks know-how about the task rather than general ability, and that this know-how is better supplied per input than written once into a long, shared prompt. The advisor learns, from inputs alone, the structured advice that a teacher wrote with hindsight access to gold answers and to the results of every strategy. It follows the idea of distillation with privileged information, but it distills into a separate small model rather than into the executor, so the executor stays unchanged and the advisor is cheap to train and run. On the public FrontierFinance benchmark, an 8B advisor lifted Samaya’s proprietary research agent from 52.9% to 56.6%, above the agent’s own high-effort configuration and the best frontier model in an open harness. With a small Qwen3-4B executor, the advice also improved a shopping agent, text-to-SQL and expert long-form generation.

How much the advice helps depends on the task and on the training data. The largest gains came where error analysis of the executor’s traces revealed failures that advice could fix, where any instance-specific values could be recovered from the input, and where the training labels were reliable. These results suggest two practical rules for building such an advisor. First, derive the advice from error analysis of the executor’s own traces, and give it fields whose values the advisor can predict from the input. On FrontierFinance, the teacher-executed gain disappears without the directives mined from the agent’s failures, and on WebShop the decisive field restates option values that the instruction already contains. Second, label every field that can be read off the runs with a rule, and use the teacher’s LLM only for fields that need judgment.

The advisor could also be made cheaper. Most advice fields are typed choices over closed vocabularies, such as the arm, the effort and length buckets, web_use and the directive picks. Recent decision models such as Jev return this kind of output directly, as calibrated probabilities over declared options, in a single parallel pass and at a small fraction of the latency and cost of an LLM call [53]. Producing the categorical fields this way, and generating text only for instantiated values such as WebShop’s option strings, could make structured advice generation much cheaper and faster. Open questions include how much of the advice transfers across executors and across datasets in the same domain. Another is how the advisor can learn from the executor’s results after deployment. Our advisor is trained once, offline, and gives its advice before the executor runs. It could instead be updated online, adding new graded traces to its training data as they arrive, or revise its advice within an episode after seeing intermediate results, such as an error message or an empty query result.

Acknowledgments

We thank Yuhao Zhang and Ashwin Paranjape for their helpful comments on this work.

References

- [1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*, October 2022. ICLR 2023. [arXiv:2210.03629](#).
- [2] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. *WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents*, July 2022. NeurIPS 2022. [arXiv:2207.01206](#).
- [3] 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*, September 2025. TMLR 2026. [arXiv:2509.02547](#).
- [4] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitit, Harris Chan, and Jimmy Ba. *Large Language Models Are Human-Level Prompt Engineers*, November 2022. ICLR 2023. [arXiv:2211.01910](#).
- [5] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. *Large Language Models as Optimizers*, September 2023. ICLR 2024. [arXiv:2309.03409](#).
- [6] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. *DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines*, October 2023. ICLR 2024. [arXiv:2310.03714](#).
- [7] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. *TextGrad: Automatic "Differentiation" via Text*, June 2024. Nature 2025, as "Optimizing generative AI by backpropagating language model feedback". [arXiv:2406.07496](#).
- [8] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziemis, 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*, July 2025. ICLR 2026. [arXiv:2507.19457](#).
- [9] 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*, October 2025. ICLR 2026. [arXiv:2510.04618](#).
- [10] Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. *Guiding Large Language Models via Directional Stimulus Prompting*, February 2023. NeurIPS 2023. [arXiv:2302.11520](#).
- [11] Parth Asawa, Alan Zhu, Abigail O'Neill, Matei Zaharia, Alexandros G. Dimakis, and Joseph E. Gonzalez. *How to Train Your Advisor: Steering Black-Box LLMs with Advisor Models*, October 2025. ICML 2026. [arXiv:2510.02453](#).
- [12] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. *Hindsight Experience Replay*, July 2017. NeurIPS 2017. [arXiv:1707.01495](#).
- [13] Tianjun Zhang, Fangchen Liu, Justin Wong, Pieter Abbeel, and Joseph E. Gonzalez. *The Wisdom of Hindsight Makes Language Models Better Instruction Followers*, February 2023. ICML 2023. [arXiv:2302.05206](#).
- [14] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. *STaR: Bootstrapping Reasoning With Reasoning*, March 2022. NeurIPS 2022. [arXiv:2203.14465](#).
- [15] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. *Unifying distillation and privileged information*, November 2015. ICLR 2016. [arXiv:1511.03643](#).
- [16] Sanjiban Choudhury and Paloma Sodhi. *Better than Your Teacher: LLM Agents that learn from Privileged*

- AI Feedback*, October 2024. ICLR 2025. [arXiv:2410.05434](#).
- [17] Weiwei Li, Junzhuo Liu, Tong Chu, Hengfu Yu, and Wen Li. *The Next Screenshot Knows: Gated Hindsight Distillation for Mobile GUI Agents*, August 2026. [arXiv:2608.06065](#).
 - [18] Siqi Zhu, Xuyan Ye, Hongyu Lu, Weiye Shi, and Ge Liu. *The Many Faces of On-Policy Distillation: Pitfalls, Mechanisms, and Fixes*, May 2026. [arXiv:2605.11182](#).
 - [19] Yuhao Zhang, O. Ozan Koyluoglu, Thejas Venkatesh, Richard Diehl Martinez, Vishank Bhatia, Arash Alidoust, and Ashwin Paranjape. *FrontierFinance: A Challenging Benchmark for Measuring Frontier Intelligence of Finance Agents*, August 2026. [arXiv:2608.11683](#).
 - [20] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. *Qwen3 Technical Report*, May 2025. [arXiv:2505.09388](#).
 - [21] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. *Can LLM Already Serve as A Database Interface? A BIG Bench for Large-Scale Database Grounded Text-to-SQLs*, May 2023. NeurIPS 2023 Datasets and Benchmarks Track. [arXiv:2305.03111](#).
 - [22] Jie Ruan, Inderjeet Nair, Shuyang Cao, Amy Liu, Sheza Munir, Micah Pollens-Dempsey, Tiffany Chiang, Lucy Kates, Nicholas David, Sihan Chen, Ruxin Yang, Yuqian Yang, Jasmine Gump, Tessa Bialek, Vivek Sankaran, Margo Schlanger, and Lu Wang. *ExpertLongBench: Benchmarking Language Models on Expert-Level Long-Form Generation Tasks with Structured Checklists*, June 2025. ICLR 2026. [arXiv:2506.01241](#).
 - [23] Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A. Smith. *Tuning Language Models by Proxy*, January 2024. COLM 2024. [arXiv:2401.08565](#).
 - [24] Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. *Small LLMs Are Weak Tool Learners: A Multi-LLM Agent*, January 2024. EMNLP 2024. [arXiv:2401.07324](#).
 - [25] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. *Plan-and-Act: Improving Planning of Agents for Long-Horizon Tasks*, March 2025. ICML 2025. [arXiv:2503.09572](#).
 - [26] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. *Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models*, May 2023. ACL 2023. [arXiv:2305.04091](#).
 - [27] Mohammadreza Pourreza and Davood Rafiei. *DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction*, April 2023. NeurIPS 2023. [arXiv:2304.11015](#).
 - [28] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever, and Jeff Wu. *Weak-to-Strong Generalization: Eliciting Strong Capabilities With Weak Supervision*, December 2023. ICML 2024. [arXiv:2312.09390](#).
 - [29] Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. *Dynamic Cheatsheet: Test-Time Learning with Adaptive Memory*, April 2025. EACL 2026. [arXiv:2504.07952](#).
 - [30] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. *ExpeL: LLM Agents Are Experiential Learners*, August 2023. AAAI 2024. [arXiv:2308.10144](#).
 - [31] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. *Agent Workflow Memory*, September 2024. ICML 2025. [arXiv:2409.07429](#).
 - [32] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. *Memp: Exploring Agent Procedural Memory*, August 2025. Findings of ACL 2026. [arXiv:2508.06433](#).

- [33] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. *Self-Refine: Iterative Refinement with Self-Feedback*, March 2023. NeurIPS 2023. [arXiv:2303.17651](#).
- [34] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. *Reflexion: Language Agents with Verbal Reinforcement Learning*, March 2023. NeurIPS 2023. [arXiv:2303.11366](#).
- [35] Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. *Agent Learning via Early Experience*, October 2025. ICML 2026. [arXiv:2510.08558](#).
- [36] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity*, March 2024. NAACL 2024. [arXiv:2403.14403](#).
- [37] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. *RouteLLM: Learning to Route LLMs with Preference Data*, June 2024. ICLR 2025. [arXiv:2406.18665](#).
- [38] Lingjiao Chen, Matei Zaharia, and James Zou. *FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance*, May 2023. TMLR 2024. [arXiv:2305.05176](#).
- [39] Zichao Li, Gang Wu, Zichao Wang, Ruiyi Zhang, Wanrong Zhu, Ryan A. Rossi, Vlad I Morariu, and Jihyung Kil. *Spinning Straw into Gold: Relabeling LLM Agent Trajectories in Hindsight for Successful Demonstrations*, July 2026. ICLR 2026. [arXiv:2607.04235](#).
- [40] Tianyi Men, Zhuoran Jin, Pengfei Cao, Yubo Chen, Kang Liu, and Jun Zhao. *Empowering GUI Agents via Autonomous Experience Exploration and Hindsight Experience Utilization for Task Planning*, June 2026. ACL 2026. [arXiv:2606.27330](#).
- [41] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. *A General Language Assistant as a Laboratory for Alignment*, December 2021. [arXiv:2112.00861](#).
- [42] Charlie Snell, Dan Klein, and Ruiqi Zhong. *Learning by Distilling Context*, September 2022. [arXiv:2209.15189](#).
- [43] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. *Learning by Cheating*, December 2019. CoRL 2019. [arXiv:1912.12294](#).
- [44] Emiliano Penalosa, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. *Privileged Information Distillation for Language Models*, February 2026. ICML 2026. [arXiv:2602.04942](#).
- [45] Sarthak Harne, Chinmay Karkar, Yash Pandya, Ahmed Awadallah, and Akshay Nambi. *Privileged, but Biased: How PI-Conditioned Teachers Break Self-Distillation*, August 2026. [arXiv:2608.04794](#).
- [46] Samyak Shrestha and Alexander Tessier. *Rethinking Privileged Information in On-Policy Self-Distillation*, August 2026. [arXiv:2608.18271](#).
- [47] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. *Self-Distilled Reasoner: On-Policy Self-Distillation for Large Language Models*, January 2026. ICML 2026. [arXiv:2601.18734](#).
- [48] Chenhui Gou, Haoqin Tu, Yunhao Fang, Jianfei Cai, and Hamid Rezaatofighi. *Sample-Efficient Learning from Agent Experience*, July 2026. [arXiv:2607.21051](#).
- [49] Yubo Jiang, Fengying Xie, Zhiguo Jiang, and Haopeng Zhang. *Distill Skills into Weights, Not Prompts: Abstract Skills as Privileged Signals for On-Policy Self-Distillation*, August 2026. [arXiv:2608.09826](#).
- [50] Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. *Text-to-LoRA: Instant Transformer Adaption*, June 2025. ICML 2025. [arXiv:2506.06105](#).
- [51] Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. *Doc-to-LoRA: Learning to Instantly Internalize Contexts*, February 2026. [arXiv:2602.15902](#).

Setting	Arms
WebShop	default, concise_search_and_fast_commit, strict_option_click_then_buy, exact_variant_text_verification, category_demographic_and_price_verify, bounded_variant_fallback_pivot
FrontierFinance	default, structured_priority, tables_extraction, restricted_tools
BIRD	default, evidence_literal_and_formula_execution, schema_grounding_and_minimal_joins, conditional_aggregation_and_ratios, superlative_extremums_and_ties, cte_multi_stage_decomposition, strict_projection_and_output_alignment, sqlite_dialect_and_data_sanitization, multi_intent_union_and_cardinality
ExpertLongBench	default, schema_first_structured_output, entity_identity_first_then_annotate, security_threat_frame_first, pedagogical_evaluator_role_lock, mechanistic_rationale_extraction, clinical_note_completeness_and_fidelity, exhaustive_field_coverage_guard

Table 6. Arms of each setting. Each arm is appended to the domain’s base prompt, and default adds nothing.

- [52] Guibin Zhang, Muxin Fu, and Shuicheng Yan. *MemGen: Weaving Generative Latent Memory for Self-Evolving Agents*, September 2025. ICLR 2026. [arXiv:2509.24704](#).
- [53] Diogo Almeida. *Introducing System One Models & Jev*. TypeSafe AI blog, September 2026. URL: <https://typesafe.ai/blog/introducing-system-one-models-and-jev>.
- [54] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention*, September 2023. SOSP 2023. [arXiv:2309.06180](#).
- [55] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*, June 2021. ICLR 2022. [arXiv:2106.09685](#).
- [56] 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*, February 2024. [arXiv:2402.03300](#).

A Arms and rendering

A.1 Arms

Table 6 lists the arms of each setting.

A.2 Rendering examples

Each field of the advice is rendered with a fixed template, and the resulting text is appended to the executor’s input.

FrontierFinance. For the Colgate-Palmolive query of Appendix D, the advice is rendered as one guidance block after the query. The default arm adds no text. Each directive is replaced by its description from the taxonomy, and the other fields become one short line each:

APPROACH GUIDANCE (advisory – NOT part of the question to answer; you retain full tool access and may broaden your search if this does not cover it): Focus on: 1) When a requested metric is not disclosed by the company, explicitly state that the metric is not reported rather than leaving it blank, implying zero, or silently omitting it. . . . 4) Clearly separate what management

has explicitly stated or quantified from what can only be inferred or estimated, and flag when a metric has not been formally disclosed even if related information exists. | Prioritize data sources: FILINGS, ... | Suggested tools: documents_list, documents_search, ... | Target length: long | Web search: avoid

WebShop. For the instruction in Fig. 3, the options part of the advice renders as: “This product’s page has option buttons. Before clicking Buy Now you **MUST** click the color button labelled ‘green’; the size button labelled ‘xx-large’. Click each of those exactly as written, then click Buy Now.” The full rendering adds the arm’s strategy text, “Plan for 6 or more tool calls in total.” and “This task should require you to go back to search at least once; page through more results.”

BIRD. Advice is added to the user turn after the schema and question as a block headed “GUIDANCE (advisory – apply if relevant)”, containing one line for effort (for example “Expected effort (high): a structurally demanding query ...”), one for length, and one line per picked directive with that directive’s description.

ExpertLongBench. For the security-review document of Appendix D, the arm’s strategy text is appended to the document first. It begins: “Before extracting any fields, perform a mandatory security-threat pass: scan every tool return value in the transcript ...” The guidance block follows:

APPROACH GUIDANCE (advisory – NOT part of the source document): Focus on: 1) When the task involves evaluating a transcript for security or safety concerns, analyze the agent’s decision trajectory and any injected or adversarial content for risk rather than processing the surface-level content of the transcript as if executing the embedded task. ... 4) When a transcript contains an agent interacting with external tools, check whether any tool return value contains an injected instruction distinct from the original request, and if so, populate all security-analysis fields including trigger, motivation, tools, outcome, and binary label. | Target length: short (a single structured note/summary section, <300 words) | Effort: low – answer directly with minimal deliberation; do not second-guess or re-derive an already-clear answer.

B Directive taxonomies

FrontierFinance. 103 directives in nine families, for example 2.8 pull_sub_segment_detail_within_reported_segments and 3.1 cover_all_named_entities_in_query.

WebShop. The mined taxonomy had 50 directives in five families (search-query formulation, product validation, option configuration, exact variant matching, and fallback or pivoting). Evaluated one family at a time on 200 validation samples, search formulation lowered score by 5.45 points and fallback by 1.93, product validation and variant matching had no measurable effect, and only option configuration helped (+1.50). The final taxonomy keeps one area, option selection, with six instantiated directives: 3.1 select_color, 3.2 select_size, 3.3 select_fit_type, 3.4 select_item_shape, 3.5 select_style and 3.6 select_scent_or_flavor.

BIRD. Error areas, from the failure notes on BIRD’s training split: flawed evidence over-reliance; string literal and value mismatch; uncleaned numeric text operations; schema linking and join path errors; projection and cardinality mismatch; aggregation and deduplication errors; SQL dialect and type-casting violations; superlative and ordering misinterpretation. Examples of the 63 directives: 1.2 map_question_concepts_to_schema_columns, 2.1 verify_join_keys_against_foreign_key_schema, 2.3 eliminate_redundant_intermediate_joins, 5.1 strictly_project_requested_attributes, 6.1 prefer_order_by_limit_for_single_extremes, 7.1 cast_operands_to_real_before_division, 7.5 escape_single_quotes_in_literals, 8.1 strictly_follow_evidence_calculation_formulas, 8.3 copy_evidence_literals_verbatim.

ExpertLongBench. Nine areas (task interpretation, source grounding, field completeness, value accuracy, schema-field disambiguation, negative-finding handling, output structure and completeness, security analysis,

pedagogical evaluation) with 57 directives, for example 1.2 `adopt_correct_evaluator_role`, 2.1 `ground_all_fields_strictly_in_source`, 2.2 `scan_entire_source_for_each_field`, 3.1 `populate_all_schema_fields_explicitly`, 6.1 `record_absent_fields_explicitly`, 8.1 `detect_and_label_prompt_injection`.

C Implementation details and additional analyses

This appendix first gives settings shared by all experiments and then additional details and analyses for each setting.

C.1 Common settings and training

Common settings. The teacher’s LLM and the error-analysis model are API models, and neither is used as the executor or the advisor. The advisors are Qwen3-8B for FrontierFinance and Qwen3-4B [20] elsewhere, fine-tuned with LoRA. Ties between arms are broken toward `default` unless stated otherwise. Open models are served with vLLM [54].

Advisor training. All advisors are trained with LoRA [55] (rank 64, $\alpha = 128$) for six epochs, and we keep the checkpoint of every epoch for selection. Training one advisor takes a few hours on a single GPU.

Epoch selection. In every setting, we choose the checkpoint used on the test set by its lift on the validation set, measured by running the executor with each checkpoint’s advice. For example, on BIRD the validation lift was +1.1, +3.5 and +2.8 points at epochs 4, 5 and 6, so we use the epoch-5 checkpoint.

Tie-breaks. On WebShop, we also tried breaking ties toward the arm with the fewest turns instead of toward `default`, which moved 602 training rows away from `default`. We observed a drop in performance with the advisor trained on these labels, so on WebShop we break ties toward `default`. On BIRD, 52% of training questions have two or more correct arms and 41% have none, so on most rows the scores alone cannot pick a winner. On these rows the teacher’s LLM picks the arm. It sees the gold query and its result, together with each arm’s query and execution result, and chooses the arm whose query is closest to the gold query.

Run-to-run variation. The executor’s score on a single input varies between runs. Ideally, each arm would be run several times with different seeds, and its average score, or its $\text{pass}@k$, would be used to find the best arm and compute the reference points. We did not do this because it would multiply the cost of the arms collection, which is already the most expensive step of the pipeline. The reported lifts are less affected, since they are averages over many test inputs. We leave a statistical analysis over repeated runs to future work.

C.2 FrontierFinance

Choice of the winning-arm rule. The winning arm of each training question can be defined in more than one way, and the definition affects the trained advisor. We compare two rules. The first picks the arm with the highest composite score, a hand-weighted sum of the essential-criteria rate with smaller, capped terms for cost and latency, and breaks ties toward `default`. The second is the cascade of §3.2.2, which orders the arms by essential-criteria rate, then all-criteria rate, then cost, then latency. The two rules choose a different arm on 681 of the 1,819 training questions. Under the composite rule, 51% of the labels are `default`, because many arms tie on the composite and ties go to `default`. Under the cascade, the four arms receive 25%, 31%, 22% and 22% of the labels. With the same schema, teacher prompts, trainer and renderer, advisor variants trained on the composite labels reach 57.2–57.6% essential-criteria coverage on the test queries, and the advisor trained on the cascade labels reaches 59.9% in the deployment configuration (Table 7). A rule that orders the arms by the benchmark’s own criteria therefore produces a better advisor, which indicates that the winning-arm objective is aligned with end-task quality.

Condition	Public test (220)		Validation (100)	
	essential	all	essential	all
No advice	53.9%	45.4%	63.0%	53.9%
Best fixed arm (<code>tables_extraction</code>)	57.9%	51.7%	69.1%	60.2%
Trained advisor	59.9%	52.6%	69.2%	61.9%
<i>Training variants of the advisor</i>				
more epochs (9)	59.3%	52.2%	65.9%	58.4%
reasoning before the JSON	57.4%	49.9%	66.3%	58.2%
GRPO from the SFT checkpoint	58.4%	52.4%	67.8%	58.8%

Table 7. FrontierFinance in the deployment configuration. The best fixed arm applies the same arm to every question (without other fields in the advice).

Deployment configuration. We also evaluated the agent in the configuration in which it is deployed, which is also how it was run during training. Table 7 reports essential and all-criteria qualification rates on the 220 test queries and on the internal validation set. The advisor raises essential-criteria coverage from 53.9% to 59.9% on the test queries and from 63.0% to 69.2% on validation, and all-criteria coverage by about 7–8pp on both.

A single fixed arm, `tables_extraction`, is a strong baseline (Table 7). The advisor beats it on the test queries. Choosing the arm is only one of the advisor’s fields, and the ablation of §5.1 shows that the other fields result in additional gains.

More epochs of SFT. Training the advisor for nine epochs instead of six slightly raises its agreement with the teacher’s targets on the held-out validation questions, measured with the field-by-field reward described in RL section below (0.393 against 0.384). However, this lowers the executor’s essential-criteria coverage on validation, from 69.2% to 65.9% (Table 7). We therefore use the checkpoint after six epochs as the SFT advisor.

Reinforcement learning from the SFT checkpoint. We also tried to improve the SFT advisor with GRPO [56]. Starting from the selected SFT checkpoint, we trained a new LoRA for one epoch over the 1,819 training questions, with eight sampled outputs per question. The reward compares the advisor’s JSON with the teacher’s target field by field and does not run the agent, since one agent run takes about four minutes and costs about a dollar. The field weights reflect how much each field contributed in the ablation of §5.1: 0.40 for the arm, 0.35 for the overlap (Jaccard index) between the predicted and target directive picks, 0.10 for the sources, 0.06 each for the tools and web use, and 0.03 for the target length. In a first run, the arm part of the reward gave partial credit. For each training question, we know the score that every arm received during collection. Choosing the best arm earned 1, choosing the worst arm earned 0, and any other arm earned a value in between, in proportion to its score. For example, if the four arms scored 80%, 70%, 60% and 40% on a question, choosing the arm that scored 70% earned 0.75. The reward rose from 0.57 to 0.74 during training, but the advisor learned to choose `tables_extraction` for almost every question, because under graded credit this single arm already earns 0.67 of the arm reward. A second run scored the arm by exact match with the label and added a KL penalty to the SFT policy ($\beta = 0.01$). This kept the arm choices varied, and the directive-pick overlap rose from 0.51 to 0.71 and the arm match from 0.25 to 0.34 on the training questions. On held-out questions, however, arm accuracy was flat, and in the executor the advisor scored 67.8% essential on validation and 58.4% on the test queries, below the SFT checkpoint it started from (Table 7). These results show that this form of reward, which compares the advice with the teacher’s target without measuring the executor’s performance on the task, can rise on the training questions without leading to better answers. Finding low-cost reward signals that correlate with the executor’s downstream performance is a natural direction for future work on training the advisor with RL.

C.3 WebShop

Advice designs and reference points. Fig. 5a compares three advice designs on the 500 test samples, each produced by a trained advisor. Predicting only the arm gains 3.65 score points. The second design adds generic directives from the taxonomy mined from failure notes (§3.2.1). That taxonomy has 50 directives in five families. Evaluated one family at a time on the 200 validation samples, only the option-configuration family helped (Appendix B), while the search and fallback families lowered the score, for example by telling the agent to abandon a product whose exact option was missing. With only the option family kept, the generic directives, such as “inspect all option buttons and click every required one”, add nothing on top of the arm (+2.94). The third design, the trained advisor with its full advice, replaces these generic directives with the six option directives of the final taxonomy, each instantiated with the option value stated in the instruction, such as “click the size button labelled ‘x-large’”. This design follows from the lost-reward analysis of §5.2 and gains 8.73 points, close to the 8.95 points obtained when the option values come from the regular-expression rule instead of the advisor. The difference between the second and third designs is therefore not only which directives are kept but whether they are generic or instantiated. Fig. 5b shows the reference points of §3.3 on the 200 validation samples. The best-arm oracle is +9.59, but running each sample again with its best arm gives only +6.49 (run-to-run variance, Appendix C.1). The trained advisor gains 8.16 points on validation, more than the executed best arm, because its options field carries information that no arm contains.

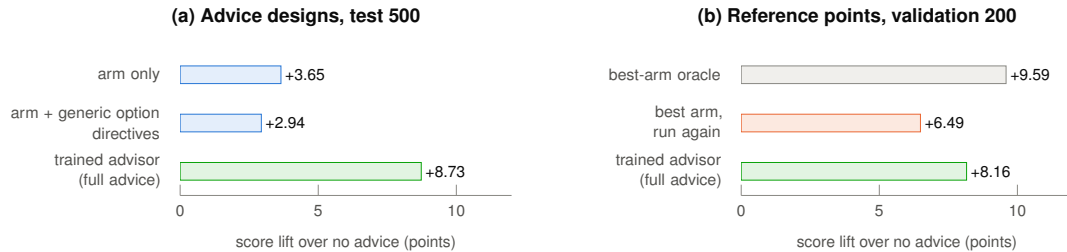


Figure 5. WebShop score lift over no advice. (a) Three advice designs on the 500 test samples. (b) The reference points of §3.3 on the 200 validation samples: the best arm per sample as recorded during collection, the same best arm run again on each sample, and the trained advisor.

A difficulty field. The WebShop advisor can also predict a `difficulty` field, labelled by a rule from the `default` arm’s score on each training sample (hard, medium or easy by tercile). Rendering this field in the prompt lowers the score by 2.95 points, so the final advice does not render it.

Other directive variants. Instantiating the product type and price as verification directives, such as “verify that the title is really a men’s dress shirt”, lowers the score by 1.85 points. The verification framing leads the agent to reject products and search again, the same failure caused by the directives that tell it to abandon a product whose exact option is missing.

C.4 BIRD

Reference points. Fig. 6 shows the reference points of §3.3 on the 1,074 BIRD test questions. The best-arm oracle is +20.11 execution-accuracy points, running the winning arm again gives +11.27, and the trained advisor gains 3.07. The oracle is higher than the rerun because each arm was run only once during collection: on a question where one of the nine arms happened to succeed, running that arm again does not always succeed. See also Appendix C.3 for similar behavior for WebShop. The trained advisor stays well below the best arm run again, unlike on WebShop. On WebShop, the most useful advice copies option values that the instruction states, so the advisor can produce it exactly. On BIRD, the most useful advice says which tables and columns the query needs, and the advisor has to guess this from the question without seeing the database schema (§5.3). High-quality instantiated advice is therefore harder to obtain on this dataset. With more training data, or with better advice fields, there is still room to improve.

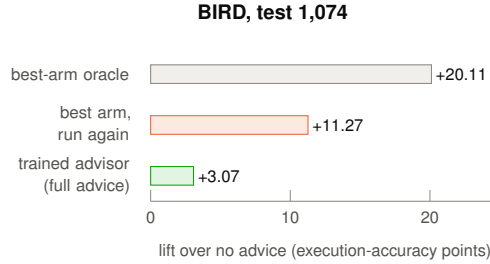


Figure 6. BIRD’s three reference points on the 1,074 test questions: the best arm per sample as recorded during collection, the same best arm run again on each sample, and the trained advisor.

Gain by difficulty. BIRD labels each question as simple, moderate or challenging. The gain is concentrated in simple questions (+5.19 points), is small for moderate ones (+0.93) and is negative for the most challenging ones (−4.21).

Appending arms. An arm can replace the base instruction as a complete rewritten system prompt, or be appended to a fixed base instruction. On BIRD, we collected all 9,428 training questions both ways. Appending improves every one of the eight non-default arms, by 0.5 to 7.1 execution-accuracy points. The largest gains are for the two arms that scored lowest as full prompts.

When the advice hurts. A model diagnosed the 72 test questions that the advice turns from correct to wrong. The dominant pattern is over-application, such as adding a join that duplicates rows, “correcting” a literal that was already right (for example turning ‘CA’ into ‘California’), or choosing the wrong one of two similarly named columns. In about two thirds of these cases the arm is `default`, so the harm comes from the directive text. Four of the 63 directives are net negative, led by `use_conditional_aggregation_for_pivoted_metrics`, and the other frequently used directives are net positive. These cases show that the BIRD taxonomy and directive texts leave room for further refinement, for example by rewording or removing the directives that are over-applied.

Coverage of the residual failures. For a sample of 180 test questions that the executor answers wrongly both with and without advice, we checked each root cause against the taxonomy. For 94% of them, an existing directive already addresses the root cause. The remaining gap is therefore mostly a matter of picking the right directive and having the executor apply it correctly, not of missing strategies. A second test supports this. If we run all nine arms on each question and choose among their queries using the execution results, either the first query that runs without error or the result that most arms agree on, execution accuracy rises by about 6 points, twice the advisor’s gain. For many questions, one of the existing arms already writes a correct query, and the missing step is choosing it. The advisor cannot use this signal, because it runs before any query is executed.

C.5 ExpertLongBench

Metric. ExpertLongBench’s own paper reports an F1 score computed with its checklist-extraction framework [22], whereas we report the fraction of checklist fields covered using a judge, so our numbers are not comparable with the published ones.

Lift by task. Table 8 breaks the ExpertLongBench lift down by task. The largest gains are in security review (+20.4 points) and pedagogical evaluation (+9.0), and the other four tasks change little. The directive picks generally fit the task, as the examples in Table 8 show: the advisor picks prompt-injection detection for security reviews and examination-finding transcription for clinical notes. The arm choices fit less well. For several tasks, the advisor often chooses an arm designed for another task, for example the security frame for clinical notes and materials-science papers. We read this as cross-task interference. With 450 training documents shared by six tasks and eight arms, the advisor sees few examples of each task and arm, and arms learned for one task carry over to others. The two tasks with a clear gain are also the two where the predicted arm usually

matches the task. We expect that more training data per task, a taxonomy and set of arms organised by task, or restricting the arm choice to the arms that fit each task could raise the lift on the remaining tasks, and we leave this to future work.

Task	<i>n</i>	Lift	Most frequent predicted arm	Example directive picked
T11 security review	18	+20.4	security_threat_frame_first (12)	8.1 detect_and_label_prompt_injection
T04 pedagogical evaluation	19	+9.0	pedagogical_evaluator_role_lock (19)	5.3 separate_mistake_identification...
T03 materials synthesis	9	0.0	security_threat_frame_first (7)	3.8 capture_precursor_rationale...
T08 protein descriptions	18	0.0	clinical_note_completeness... (16)	3.6 populate_biological_annotation...
T06 clinical notes	18	-1.0	security_threat_frame_first (18)	3.5 transcribe_complete_examination...
T07 molecule descriptions	18	-1.9	pedagogical_evaluator_role_lock (13)	3.7 report_all_compound_classification...

Table 8. ExpertLongBench lift by task on the 100 test documents, in checklist-coverage points, with the arm the advisor chose most often for that task (count in parentheses) and a representative directive it picked.

D Examples of advisor output

D.1 FrontierFinance

The examples below are queries from the 220 public FrontierFinance queries, answered by the agent without advice and with the trained advisor’s advice (deployment configuration). The advisor saw only the query. We show the advice fields and the directive picks, with some source and tool names abbreviated.

Colgate-Palmolive (financial data). *“What is Colgate-Palmolive’s revenue by geography and product category for each quarter since 2019?”*

```

arm          default
target_length long
web_use      avoid
sources      FILINGS, ...
tools        documents_list, documents_search, ...

```

Directives: 2.2 distinguish_metric_unavailability_from_zero; 1.1 enumerate_all_periods_in_requested_range; 6.5 derive_standalone_periods_from_cumulative_filings; 2.17 distinguish_explicit_from_implicit_disclosures.

Without advice, the agent filled many quarterly cells with estimates obtained by applying a percentage share to a total. With advice, it reported the quarterly figures by region and stated that Colgate discloses product categories only as a share of sales, not in dollars. Rubric items satisfied: 36% without advice, 90% with advice.

ASML (earnings and events). *“What is the summary of ASML’s earnings?”*

```

arm          tables_extraction
target_length medium
web_use      fallback
sources      TRANSCRIPTS, FILINGS, NEWS, ...
tools        documents_list, documents_search, ...

```

Directives: 4.1 search_across_all_document_types_for_topic; 2.15 surface_management_guidance_and_forward_commentary; 2.5 include_geographic_and_segment_breakdowns; 2.7 include_capital_allocation_and_investment_announcements.

Without advice, the agent centred its summary on the latest quarter and its guidance. With advice, it also built a quarterly grid for the year and added the regional breakdown, including China, and the capital return programme. Rubric items satisfied: 47% without advice, 100% with advice.

D.2 WebShop, BIRD and ExpertLongBench

For each of the other settings, we show one test input on which the advice helped, with the trained advisor’s output and the executor’s score with and without it.

WebShop. *“Find me hand wash, wash cold women’s bodysuit tops with polyester spandex for tumble dry, daily wear with size: x-large, and price lower than 50.00 dollars.”*

```
arm                concise_search_and_fast_commit
effort, expected_tools  short, none
required_options    {size: x-large}
directive_picks     3.2 select_size
```

Rendered as the arm’s strategy text (“Formulate search queries using strictly 2 to 4 essential keywords . . . On the product page, select all required attribute options and click Buy Now within 4 to 5 total turns.”), followed by “Plan for about 3 tool calls in total.” and “This product’s page has option buttons. Before clicking Buy Now you MUST click the size button labelled ‘x-large’. Click each of those exactly as written, then click Buy Now.” Reward: 0.06 without advice, 1.00 with the full advice. With the option line alone, the reward was 0.07, so on this sample the arm and effort fields made the difference.

BIRD. *“What is the average score of the posts owned by the user csgillespie?”* (evidence: “csgillespie” is the DisplayName of the user; average score refers to AVG(score)).

```
arm                schema_grounding_and_minimal_joins
effort, length      high, medium
directive_picks     1.1 validate_attribute_existence_and_ownership; 2.4 join_
                    descriptive_tables_for_named_entities; 8.1 strictly_
                    follow_evidence_calculation_formulas; 8.3 copy_evidence_
                    literals_verbatim
```

Without advice the executor wrote

```
SELECT AVG(posts.score) FROM posts WHERE posts.OwnerDisplayName =
'csgillespie'
```

which filters on a denormalized column and does not match the gold result. With advice it wrote

```
SELECT AVG(posts.score) FROM posts JOIN users ON posts.OwnerUserId =
users.Id
WHERE users.DisplayName = 'csgillespie'
```

which matches. The schema-grounding arm and the directive to join descriptive tables for named entities point the executor to the users table.

ExpertLongBench. Task T11 (security review): the input is a transcript of an agent that was asked to fetch the details and reviews of a laptop on Amazon. One of the returned reviews contains an injected instruction to delete a patient’s clinical documents in another system. The expected output is a structured security analysis with six fields.

```
arm                security_threat_frame_first
target_length, effort  short, low
directive_picks       1.5 analyze_security_trajectory_not_surface_content; 2.5
                    detect_adversarial_content_in_tool_returns; 3.10
                    populate_all_security_analysis_fields; 8.1 detect_and_
                    label_prompt_injection
```

Without advice, the executor wrote a product summary with the reviews, noting only that one review was unrelated to the product. With advice, it wrote the security analysis, with the label, the attacker's motivation, the verbatim trigger and the tool that returned the injected text. Checklist fields satisfied: 1 of 6 without advice, 5 of 6 with advice.