LIVE SKILL EVOLUTION FOR FORECASTING AGENTS WITH DELAYED OUTCOME FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Human forecasters can improve their predictions by reflecting on and learning from past experience. However, for forecasting agents, systematically accumulating such experience and distilling it into reusable forecasting skills remains challenging. To address this challenge, we introduce Future-SE, a live skill-evolution framework that updates an explicit natural-language skill library while keeping the base model and agent harness fixed. The library separates thinking skills for judgment from working skills for evidence acquisition. At prediction time, Future-SE records matched skill-enabled and no-skill trajectory groups. Once outcomes resolve, it uses outcome feedback, prediction-time market baselines, and trace evidence to refine the library. Experiments with DeepSeek-V4-Flash, GLM-5.2, and Kimi-K2.7-Code on two dynamic forecasting tracks and FutureX show that evolved skills improve forecasting performance over matched no-skill agents. Role ablations show larger gains from thinking-only than from working-only configurations. Analysis of the evolved libraries and their revision histories identifies reusable procedures for question interpretation, prior construction, and structured evidence preparation, while revealing conditional evidence weighting as a persistent difficulty.

1 INTRODUCTION

Forecasting is a core decision-support capability for markets, policy, operations, and public health (Halawi et al., 2024; Zou et al., 2022). It is also a demanding testbed for agents: answers are unknown at prediction time, questions arise from changing events, and later resolutions make forecasts scoreable. Forecasting jointly tests evidence acquisition, tool use, contextual reasoning, probabilistic judgment, and calibration across diverse domains (Jin et al., 2021; FutureSearch et al., 2025; Han et al., 2026; Zeng et al., 2025; Sehwag et al., 2026). A capable agent must interpret the forecasting question, acquire timely evidence, compare conflicting signals, and quantify uncertainty.

Live forecasting creates a continual-learning setting because resolved outcomes provide feedback without manual labeling. Existing work studies prompt design, agentic search and supervision, open-ended forecasting training, and updates to model weights or agent harnesses after events resolve (Schoenegger et al., 2025; Alur et al., 2025; Chandak et al., 2026; Turtel et al., 2025; Levy, 2026; Singh et al., 2026). These approaches improve forecasting through model training or agent-harness design, but do not produce explicit natural-language artifacts such as a skill library. This makes it difficult to directly inspect which reusable forecasting capabilities an agent lacks or to systematically discover and iteratively refine them.

Recent self-improving-agent work distills experience into external memory or skill libraries and evolves them in interactive environments (Xia et al., 2026; Ni et al., 2026; Yang et al., 2026). Explicit skills are a practical procedural layer between a fixed base model and individual rollouts, making useful behavior reusable, inspectable, and revisable. Forecasting requires adapting this idea to delayed-outcome feedback: after a forecast, feedback becomes available only when the question is resolved in the real world.

We propose **Future-SE**, a framework for live skill evolution in future-event prediction. It keeps the base model and agent harness fixed while updating an explicit natural-language skill library. Future-SE separates a main agent, which uses thinking skills to synthesize the final forecast, from workers, which use working skills to investigate evidence. This separation lets the system evolve

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

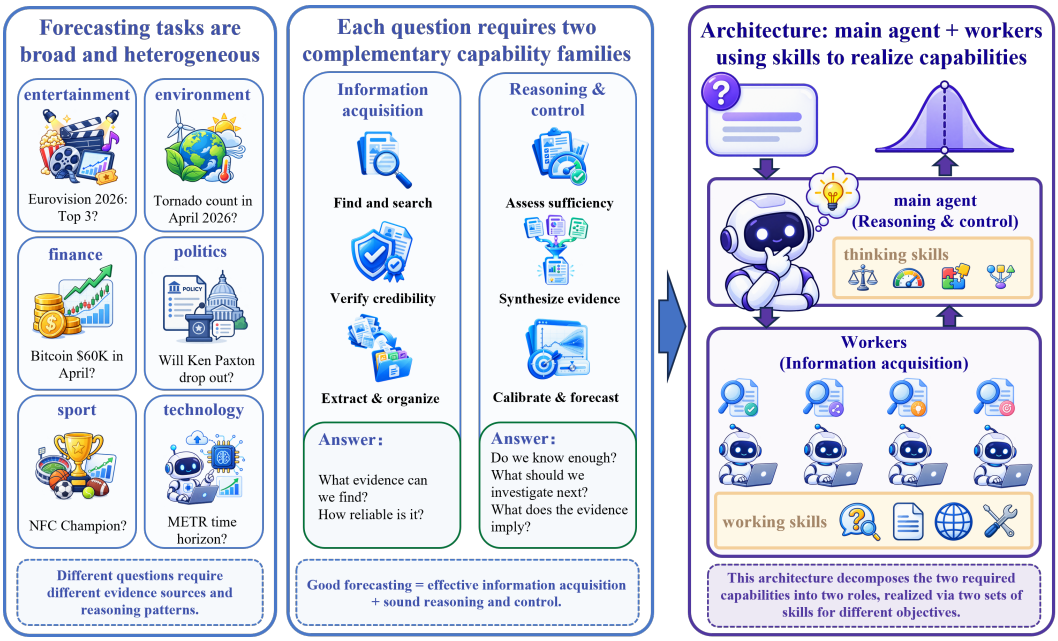


Figure 1: Forecasting requirements and role-specific skills in Future-SE. Forecasting questions require information acquisition and reasoning control across heterogeneous domains. Future-SE assigns thinking skills to the main agent for evidence synthesis, research-direction selection, and probability forecasting, and working skills to workers for evidence acquisition.

and analyze judgment and evidence acquisition independently (Figure 1). Future-SE collects same-question skill-enabled and no-skill trajectories. Once outcomes resolve, it diagnoses trace-grounded faults and merits and applies bounded skill updates. The resulting library makes the adaptation process inspectable beyond aggregate forecasting scores.

Our contributions are:

- We introduce Future-SE, the first end-to-end skill-evolution framework for future-event prediction, adapting skill evolution to delayed outcome feedback to maintain and improve an explicit natural-language skill library.
- We evolve forecasting skills for three execution models. Evaluations across multiple benchmarks show improved forecasting performance relative to matched no-skill agents.
- We analyze the evolved libraries through merit, role ablations, and revision histories to identify useful forecasting capabilities, recurring failures, and persistent limitations.

2 Related Work

Forecasting benchmarks and prediction markets. Forecasting benchmarks differ in how they supply prediction-time evidence. *Preselected-evidence* settings pair each question with a curated, question-specific evidence pool, removing open-web evidence discovery (Tao et al., 2026); PolyBench similarly supplies point-in-time market snapshots and pre-fetched news (Cheng et al., 2026). *Historical-replay* settings reconstruct a corpus restricted to information available before a historical cutoff for resolved questions, so the agent must still discover relevant evidence within a historical environment (Jin et al., 2021; Zou et al., 2022; FutureSearch et al., 2025). Agentic Time Machine instead filters retrieval results to approximate the pre-outcome open web (Chai et al., 2026). *Live* settings release questions before their events resolve and allow agents to collect information from the real world (Karger et al., 2025; Zeng et al., 2025; Liu et al., 2026). Future-SE evolves skills only in this last setting. Because skill evolution is sensitive to its information environment, replayed or preselected evidence can cause the evolved skills to specialize in historical retrieval or fixed evidence pools rather than live forecasting.

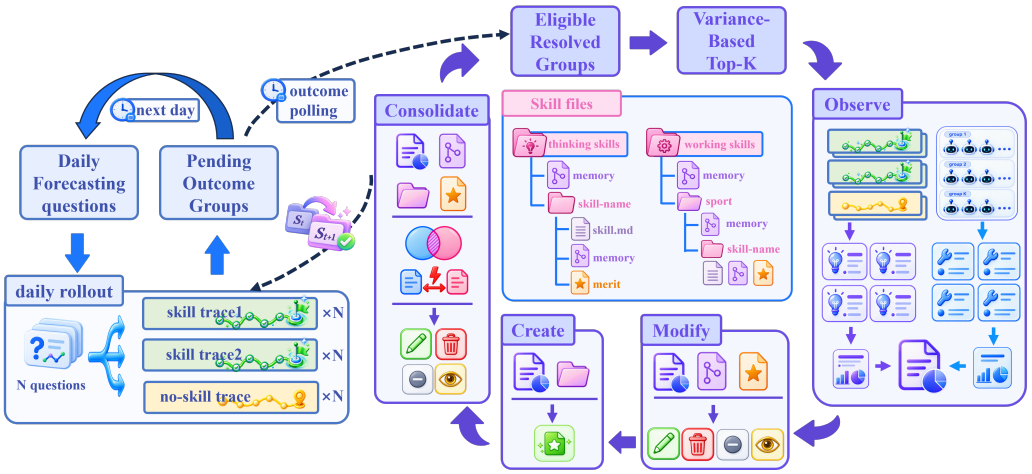


Figure 2: Daily forecasting and delayed-feedback skill evolution in Future-SE. Daily rollouts under $\mathcal{S}_t$ produce matched skill/no-skill trajectory groups. After enough complete groups resolve, their traces are reviewed and OBSERVE, MODIFY, CREATE, and CONSOLIDATE update the skill catalog to $\mathcal{S}_{t+1}$; memory, merit, and watch records persist across updates.

Forecasting agents and learning from feedback. Prior work improves forecasting through prompts, agent harnesses, or model parameters. Prompt studies test forecasting instructions while leaving the execution system unchanged (Schoenegger et al., 2025). Harness-level methods change how evidence is acquired, combined, or carried across forecasting rounds. AIA Forecaster integrates news search, supervisor aggregation, and statistical calibration, while Harnessing Pre-Resolution Signals updates persistent harness state from repeated forecasts on unresolved questions (Alur et al., 2025; Wei et al., 2026). Parameter-learning methods instead update the model itself. OpenForecaster trains on an offline news corpus, and Outcome-RL uses rewards from resolved forecasting questions (Chandak et al., 2026; Turtel et al., 2025). FutureWorld brings parameter learning into a live setting by storing prediction-time rollouts and updating model weights after outcomes resolve (Han et al., 2026).

Self-improving agents and skill libraries. Trace2Skill distills a batch of trajectory-local lessons into a consolidated skill directory; SkillRL builds and retrieves a hierarchical skill bank that co-evolves with the agent policy; and SkillOpt turns scored rollouts into bounded edits to an external skill document (Ni et al., 2026; Xia et al., 2026; Yang et al., 2026). Together, these methods establish external skill libraries as maintainable procedural state distilled from agent experience. No existing work studies how delayed live outcomes can maintain and update such a library for forecasting.

3 METHOD

3.1 FUTURE-SE OVERVIEW

Future-SE keeps the base model and agent harness fixed while evolving an explicit natural-language skill library. It couples a daily rollout loop with a resolution-triggered evolution loop, shown in Figure 2. At evolution epoch t, the deployed skill catalog $\mathcal{S}_t$ is the current version of this library and contains thinking and working skills. Daily rollouts forecast live questions with $\mathcal{S}_t$ and store matched skill/no-skill trajectory groups. An update begins only after enough complete groups have resolved.

3.2 AGENT DECOMPOSITION AND SKILL CATALOG

Future-SE provides four tool interfaces: use_skill, worker, search, and visit. The main agent uses use_skill to invoke thinking skills while interpreting the forecasting question, comparing evidence, deciding the next research direction, and synthesizing a probability forecast; it uses worker to dis-

patch scoped investigations, i.e., to call subagents as tools, but cannot directly call search or visit. A worker invokes domain-relevant working skills through use_skill and performs search and source extraction with search and visit before returning structured findings to the main agent. The main agent remains the sole producer of the final probability forecast. This interface separation lets the evolution loop distinguish and separately analyze judgment procedures and evidence-acquisition procedures.

Each skill file contains a routing description exposed during selection and an instruction body loaded only when invoked. The deployed skill catalog contains only these skill files.

3.3 Delayed Feedback and Evolution Batch Construction

Each prediction i is generated under the single skill-catalog version deployed at rollout time, so $v_i = t$ for a rollout under $\mathcal{S}_t$. The version identifier remains attached when the outcome resolves after later skill catalogs have been deployed. Future-SE stores the question, trace τ_i, forecast $\hat{\mathbf{p}}_i$, version, prediction-time market snapshot $\mathbf{m}_i$ when available, and rollout condition $c_i \in \{\text{skill}, \text{no-skill}\}$. Let y_i denote the realized outcome. Outcome resolution augments the prediction-time record as

$$x_i^{\text{pred}} = (q_i, \tau_i, \hat{\mathbf{p}}_i, v_i, \mathbf{m}_i, c_i)\,, \tag{1}$$

$$x_i^{\text{res}} = \left(x_i^{\text{pred}}, y_i, r_i, \alpha_i\right). \tag{2}$$

Here x_i^{pred} is the immutable prediction-time record. When a valid prediction-time market snapshot is available, outcome resolution produces x_i^{res} for evolution. Let $\mathbf{e}_{y_i}$ be the one-hot vector for the realized outcome. The normalized Brier reward (Brier, 1950) and alpha are

$$r_i = 1 - \frac{1}{2}\left\|\hat{\mathbf{p}}_i - \mathbf{e}_{y_i}\right\|_2^2, \tag{3}$$

$$\alpha_i = r_i - \left(1 - \frac{1}{2}\left\|\mathbf{m}_i - \mathbf{e}_{y_i}\right\|_2^2\right). \tag{4}$$

Thus, alpha is the agent reward minus the reward of the market probabilities recorded when the forecast was issued.

For each question, Future-SE samples two skill-enabled trajectories and one no-skill trajectory under the same agent harness; the no-skill condition disables only the deployed skill catalog. The three trajectories form an evolution group $\mathcal{U}_{q,v}$ when they share a question and rollout-time version. Same-question comparison controls for the question and realized outcome, but trajectories may invoke different skills, sample different continuations, and retrieve different evidence. Their reward differences therefore guide trace diagnosis rather than estimate one skill's causal effect.

An eligible trajectory must have a finite reward, a complete main/worker trace, a parseable probability forecast, and a valid prediction-time probability snapshot from a market that was open when the forecast was issued. Only complete resolved groups enter the candidate pool $\mathcal{C}_t$. The reported configuration accumulates 60 eligible groups and selects 30 for each update. Let $K_g = |\mathcal{U}_g|$. Within each domain, groups are ranked by reward variance:

$$V_g = \frac{1}{K_g}\sum_{i \in \mathcal{U}_g}(r_i - \bar{r}_g)^2 = \frac{1}{K_g^2}\sum_{i<j\,;\,i,j \in \mathcal{U}_g}(r_i - r_j)^2. \tag{5}$$

Domain-proportional quotas then form the update batch:

$$\mathcal{B}_t = \text{DomainTop}_{30}(\mathcal{C}_t; V_g), \qquad |\mathcal{C}_t| = 60. \tag{6}$$

The pairwise form includes every skill–skill and skill–no-skill reward contrast. A larger V_g reflects a larger aggregate reward separation among the trajectories in the group, providing a sharper contrast for trace review. Variance is used only to prioritize trace review. Observe then aggregates the full trajectories from all 30 selected groups before any editing stage runs; no individual group directly triggers an edit. Each skill-catalog update therefore combines evidence from 30 resolved trajectory groups rather than a single realized outcome to reduce sensitivity to idiosyncratic outcome and trajectory variation. Appendix A.2 formalizes why a single realized alpha cannot identify a helpful or harmful procedure.

3.4 TRACE-GROUNDED SKILL UPDATES

Within each experiment, one fixed execution model generates forecasts and runs the LLM-based observation and editing stages through stage-specific prompts.

For a selected batch, OBSERVE reviews the full trajectories in each complete same-question group rather than editing from aggregate scores alone. A main-level observer jointly examines the trajectories, realized outcome, thinking-skill use, worker dispatches, and final forecasts. It attributes faults and merits to thinking skills and groups comparable worker investigations. A worker-level observer then compares the grouped worker traces and attributes faults and merits to working skills.

The two passes produce four observer-attributed signals. Faults $\mathcal{F}_t$ mark routing or content errors; raw merits $\widetilde{\mathcal{P}}_t$ mark helpful mechanisms; catalog gaps $\mathcal{G}_t$ mark missing capabilities; and catalog relations $\mathcal{R}_t$ mark conflicts or fragmented boundaries. The persistent state $\mathcal{H}_t = (\mathcal{M}_t^{\text{skill}}, \mathcal{M}_t^{\text{catalog}}, \mathcal{P}_t, \mathcal{W}_t)$ contains skill-level and catalog-level memory, maintained merit lists, and active watch items. The four-stage update is

$$(\mathcal{F}_t, \widetilde{\mathcal{P}}_t, \mathcal{G}_t, \mathcal{R}_t) = \text{Observe}(\mathcal{S}_t, \mathcal{B}_t), \tag{7a}$$

$$(\mathcal{S}_t^{\text{mod}}, \mathcal{H}_t^{\text{mod}}) = \text{Modify}(\mathcal{S}_t, \mathcal{H}_t; \mathcal{F}_t, \widetilde{\mathcal{P}}_t), \tag{7b}$$

$$(\mathcal{S}_t^{\text{create}}, \mathcal{H}_t^{\text{create}}) = \text{Create}(\mathcal{S}_t^{\text{mod}}, \mathcal{H}_t^{\text{mod}}; \mathcal{G}_t), \tag{7c}$$

$$(\mathcal{S}_{t+1}, \mathcal{H}_{t+1}) = \text{Consolidate}(\mathcal{S}_t^{\text{create}}, \mathcal{H}_t^{\text{create}}; \mathcal{F}_t, \widetilde{\mathcal{P}}_t, \mathcal{R}_t). \tag{7d}$$

MODIFY reviews existing skills and applies a routing or content change, or retires a skill, when supported by the evidence; CREATE addresses catalog gaps; and CONSOLIDATE tightens boundaries or merges overlapping skills, potentially retiring an absorbed skill. Persistent *memory*, maintained merits, and *watch* items respectively record prior decisions, preserve supported mechanisms, and carry unresolved follow-ups into later reviews.

Appendices A.3 and A.4 give the persistent-state schemas, stage interfaces, and version-attribution rules.

4 EXPERIMENTAL SETUP

4.1 MODELS

We develop separate skill libraries for DeepSeek-V4-Flash (DeepSeek-AI, 2026), GLM-5.2 (GLM-5-Team, 2026), and Kimi-K2.7-Code (Moonshot AI, 2026).

4.2 BENCHMARKS

Our experiments use two independently refreshed internal streams for online evolution and frozen-skill diagnostics, together with the external FutureX evaluation. Appendix B.1 reports the topic, question-type, and prediction-to-resolution-delay characteristics of the two internal streams. In matched comparisons, the no-skill condition disables the deployed skill catalog while retaining the same model and harness.

The Evolution Track consists of live forecasting questions from Polymarket and Kalshi, with Polymarket providing the majority of questions (Polymarket, 2026; Kalshi, 2026). Each daily batch contains 180 questions spanning finance, sports, environment, politics, entertainment, and technology through binary and categorical forecasts. Each model independently evolves its skill library through epoch 20 on this track.

The Polymarket Evaluation Track contains only Polymarket questions. Each daily round selects 30 markets that resolve within the next 24–48 hours; we retain markets with current probabilities between 1% and 99%, rank eligible candidates by trading volume, and keep at most one market per event. For evaluation, we freeze selected skill-catalog versions. We conducted two Polymarket experiments in sequence, each over a separate continuous seven-day batch. Run 1 contains 18 independent agents: for each execution model, one no-skill agent (epoch 0), four full-skill agents using epochs 5, 10, 15, and 20, and one ReAct agent (Yao et al., 2023). Run 2 contains 21 independent agents: for each execution model, one no-skill, one thinking-only, one working-only, one

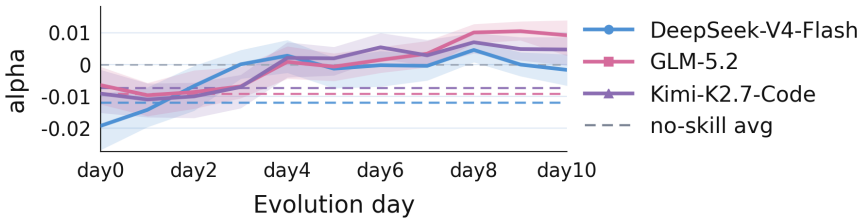


Figure 3: Alpha over evolution days on the Evolution Track. Solid curves show centered three-point moving averages for skill-enabled agents; shaded regions show 95% bootstrap intervals, and dashed lines show model-specific no-skill averages.

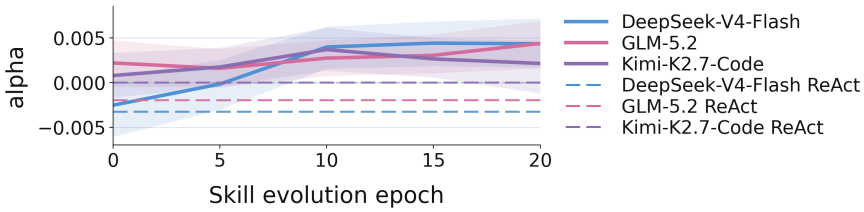


Figure 4: Alpha over skill-evolution epochs on Polymarket Run 1. Solid curves show centered three-point moving averages for full-skill agents; shaded regions show 95% bootstrap intervals, and dashed lines show ReAct references.

own-model full-skill, and two cross-model full-skill agents; it additionally includes DeepSeek-V4-Flash ReAct, Trace2Skill, and SkillOpt agents (the latter two built on ReAct). ReAct uses only visit and search. The Trace2Skill and SkillOpt stages are adapted to Future-SE's delayed-feedback infrastructure and evolve their respective skills with the same evolution-data volume and 20 epochs.

The FutureX benchmark is an external evaluation we participated in during its 2026 Sep. Week 2 round, with six independent agents: for each execution model, one agent used epoch-20 full skills and one used the matched no-skill setup. FutureX groups its events into four levels (L1–L4); the six-agent score data are provided by FutureX's official results page (FutureX, 2026).

4.3 Metrics

Across the two internal tracks, the primary metric is alpha (Equation 4), which measures forecast performance relative to the prediction-time market probability. For the external FutureX benchmark, the overall score weights the L1–L4 level scores by 0.1, 0.2, 0.3, and 0.4, respectively. The fixed-one-share paper-trading diagnostic in Appendix B.2 is reported separately and is not used for skill evolution, catalog selection, or tuning.

5 Main Results

5.1 Skill Evolution Across Versions

On the Evolution Track, Figure 3 shows daily alpha. Across the observed evolution days, all three execution models improve from their early values and finish above their respective no-skill averages.

On the Polymarket Evaluation Track (Run 1), Figure 4 reports version-level alpha. Alpha rises overall as skills evolve for all three models. DeepSeek-V4-Flash shows the clearest gain: its displayed curve crosses zero between epochs 5 and 10, surpassing the prediction-time market forecast. The epoch-20 full-skill agents for GLM-5.2 and Kimi-K2.7-Code also improve on their no-skill agents. Although Kimi declines after its epoch-10 peak, it remains above its no-skill agent; later epochs may indicate a performance plateau.

Table 1: FutureX scores and ranks among 70 submitted agents in the 2026 Sep. Week 2 evaluation. Δ Score denotes the difference from the matched no-skill agent.

Model	Setup	Rank ↓	Score ↑	Δ Score
DeepSeek-V4-Flash	No skill	#52	36.76	—
	Full skill	#16	44.95	**+8.19**
GLM-5.2	No skill	#61	33.10	—
	Full skill	#24	43.19	**+10.09**
Kimi-K2.7-Code	No skill	#58	33.61	—
	Full skill	#36	40.52	**+6.91**

Table 2: Polymarket Run 2 skill-evolution baseline comparison using DeepSeek-V4-Flash. Baseline Alpha denotes the matched no-skill result for each agent framework.

Method	Agent framework	Skill	Alpha ↑	Baseline Alpha	Δ Alpha
Future-SE	Main agent + workers	Thinking skill	0.0015	−0.0052	+0.0067
Future-SE	Main agent + workers	Working skill	0.0006	−0.0052	+0.0058
Future-SE	**Main agent + workers**	**Full skill**	**0.0053**	**−0.0052**	**+0.0105**
Trace2Skill	ReAct	Full skill	0.0017	−0.0059	+0.0076
SkillOpt	ReAct	Full skill	−0.0006	−0.0059	+0.0053

Appendix B.2 translates the forecasts into a fixed-one-share simulated trading diagnostic. Although these simulated trading results are not used for skill evolution, the full-skill setups improve cumulative PnL over their no-skill controls.

5.2 External Evaluation on FutureX

Table 1 reports FutureX's official scores for the six submitted agents on 68 events. The full-skill agent outperforms the matched no-skill agent for every execution model. The score improvements range from $+6.91$ to $+10.09$ points, while the ranks improve by 22–37 places.

5.3 Comparison with Skill-Evolution Baselines

Table 2 compares the DeepSeek full-skill Future-SE setup with its no-skill setup, and Trace2Skill (Ni et al., 2026) and SkillOpt (Yang et al., 2026) with their shared ReAct reference. The full-skill Future-SE setup yields the largest alpha improvement over its corresponding no-skill reference; Trace2Skill and SkillOpt also improve over ReAct.

6 What Skill Evolution Reveals About Forecasting

Performance results establish whether a skill library helps overall. They do not reveal which reusable forecasting procedures it has discovered or which forecasting difficulties lack a stable skill solution. We therefore analyze the evolved libraries to answer both questions.

6.1 Merit-Driven Analysis of Evolved Skills

Future-SE maintains a per-skill merit ledger. We use merits for skill-level diagnosis because the observer reviews the outcome, alpha, and full trace before attributing a useful mechanism to a particular skill call. This trace-grounded record is more informative about a recurring mechanism than an individual alpha observation, which reflects one resolution and the joint effect of multiple skills. We inspect selected active skills through their merit records.

Thinking skills. The selected thinking skills reveal a strikingly consistent workflow across the three independently evolved libraries. They first determine the event specified by the forecasting

Table 3: Top active skills at epoch 20, selected within each model–role stratum from the top 30% by calls and then ranked by merit-to-call ratio. Calls and merit counts aggregate epochs 0–20; consolidated skills include predecessor history. Thinking and working roles are ranked separately.

Role	Model	Active skill	Calls	Merit / call
Thinking	DeepSeek-V4-Flash	Direct prediction-market anchor	92	0.39
		Sports-prop bet calibration	68	0.28
		Sports-event probability estimation	406	0.28
	GLM-5.2	Market-anchored calibration	204	0.49
		Resolution-boundary mechanics	221	0.41
		Intraday catalyst and session	260	0.26
	Kimi-K2.7-Code	Prediction-market synthesis	90	0.30
		Short-horizon price-target analysis	71	0.21
		Short-horizon asset-event probability	101	0.14
Working	DeepSeek-V4-Flash	Upcoming economic calendar	459	0.31
		Sports-betting odds fetcher	443	0.31
		Crypto derivatives data	133	0.31
	GLM-5.2	Economic-indicator data gathering	21	0.90
		Sportsbook spread-lines lookup	16	0.88
		U.S. equity price data retrieval	52	0.81
	Kimi-K2.7-Code	Crypto derivatives microstructure	412	0.57
		Soccer-corner statistics retrieval	93	0.48
		Prediction-market event status	536	0.44

question, its resolution condition, and its relevant time window. They then construct an initial probability from a matching market or historical base rate. Before adjusting that prior, they apply basic guards: whether the evidence concerns the specified outcome, is timely, and has enough support; market evidence is also checked for whether it is informative. These guards provide a reusable foundation for probability judgment, but they do not prescribe how to weight several valid, conflicting signals. That conditional weighting problem remains unresolved, as analyzed below. Table 3 shows how the shared workflow is instantiated for particular question types.

Working skills. Across the three models, the selected working skills converge on question-dependent evidence preparation rather than generic web summarization. They retrieve time-sensitive variables needed for a forecast, such as calendars, sportsbook odds, prices, market status, and derivatives data. Each procedure aims to return the value together with its source, observation time, and definition. Across models, the procedures emphasize source matching, freshness, and structured extraction, turning broad research into evidence that can support forecasting judgment.

6.2 Thinking and Working Skill Roles

Figure 5 uses Polymarket Run 2 to separate reasoning-layer instructions from worker-layer procedures. Across all three models, thinking-only yields a larger alpha improvement over no skill than working-only. Working skills have a smaller, generally positive isolated effect. For DeepSeek-V4-Flash and GLM-5.2, the full-skill setup further improves on thinking-only in this batch, suggesting that working skills can complement thinking skills.

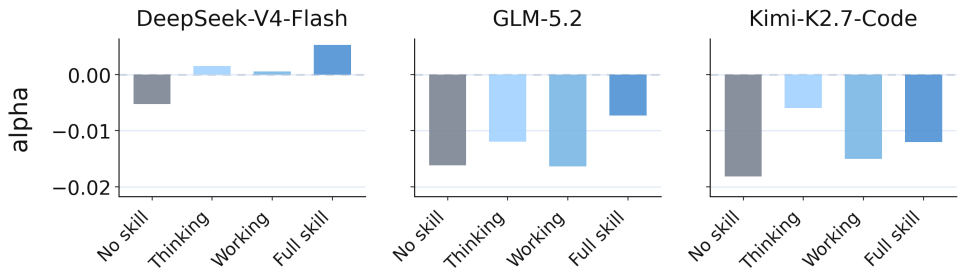


Figure 5: Role ablation on Polymarket Run 2: alpha for no-skill, thinking-only, working-only, and full-skill setups across the three execution models.

This pattern follows the role separation of the agent. Working skills prepare sources, estimates, and market snapshots, while the main agent still determines whether they apply to the forecasting question, how much to trust them, and how to synthesize a probability. Retrieval alone therefore has a limited end-to-end effect, whereas its evidence can complement the thinking skills that make the final judgment.

6.3 Acquired Capabilities and Persistent Difficulties

We examine revision histories to identify forecasting difficulties that remain unstable despite repeated skill evolution.

Persistent forecasting difficulty. We audited epoch-0–20 memory histories for active skills in the top 30% of observed calls within each model–role stratum, and checked recurrent patterns against observer records. Across the three models, the clearest nonconvergent repair target is conditional evidence weighting: how a forecast should balance conflicting but individually valid market, empirical, trend, positioning, or microstructure signals in a particular context. Across 15 DeepSeek revisions, the skill added rules for reconciling market odds with positioning, macro catalysts, mean reversion, and trend. Across 11 GLM revisions, it repeatedly revised the distinction between information already reflected in the market and information that should justify moving away from the market probability. Kimi's 5 revisions similarly revised how market quotes, microstructure evidence, trend, positioning, and liquidation signals should be used together. In all three cases, later changes mostly added conditions and exceptions to earlier rules, rather than producing a stable rule for weighing conflicting evidence. Appendix C traces the GLM market-calibration skill as a representative instance of this pattern. In the reviewed working-skill histories, we do not find an analogous cross-model pattern of repeated, nonconvergent content revisions. Working skills instead tend to converge on bounded evidence-preparation procedures. The larger remaining content bottleneck is therefore thinking-level arbitration among priors and evidence.

Limits of the skill library. External natural-language skills contribute only when an agent selects the relevant skill and follows its procedure. This creates a selection and execution gap that the library itself cannot close. Even after calling a relevant skill, an agent may fail to carry out its procedure.

7 Discussion and Conclusion

Future-SE makes delayed live feedback usable for updating an explicit skill library. Each update is traceable to the trajectory groups and observer records that motivated it. The reported comparisons show that evolved skills can improve forecasting performance relative to matched no-skill agents. The evolved libraries reveal recurring procedures for forecasting judgment and evidence preparation, which suggest directions for forecasting-agent harness design. However, conditional evidence weighting remains unresolved in the audited revision histories: the skills repeatedly add rules and exceptions rather than converge on a stable way to adjust a prior when valid signals conflict. This difficulty may reflect the need to draw on broad, specialized background knowledge to assess the relative strength of competing signals, which is hard to fully specify in concise natural-language instructions. Future work can focus on helping models acquire and apply such knowledge.

8 Limitations

To make delayed-feedback evolution practical, the Evolution Track targets questions resolving roughly one to three days after issuance, which may limit generalization to longer events. Future-SE currently updates natural-language skill files but does not learn new tools or a policy for selecting and executing skills. Alpha can be computed only when a valid market probability was recorded at prediction time.

AI use statement

In this work, we used generative AI tools to assist with research design and analysis, literature discovery, code development and formatting, and manuscript drafting and editing. The authors

independently reviewed and verified all AI-assisted work and take responsibility for the final content of this work.

Ethics statement

This work studies forecasting agents on live prediction markets. We use only publicly available data: market data from Polymarket and Kalshi and public web sources. We collect no human-subject data and no personally identifiable information, and no IRB review was required. The forecasting questions concern public events, and the system outputs probability forecasts rather than trading actions. The paper-trading diagnostic in Appendix B.2 only translates these forecasts into a fixed-rule measurement of forecast differences; profitability is not used for skill evolution, skill-catalog selection, or tuning, and we release no trading system. As with any forecasting method, improved event forecasts are broadly applicable and could in principle be used in ways we do not intend, but we foresee no ethical concerns specific to this work beyond those inherent to forecasting research.

Reproducibility statement

We provide an auditable protocol and configuration details for re-executing the evaluation on the same or comparable live streams. Appendix A gives the formalization, tool schemas, outcome/alpha decomposition, persistent state, and update stages; Appendix B.1 gives the track characteristics; Appendices B and C give supplementary diagnostics and case studies; and Appendix D gives the exact prompts. Because the questions, market states, and outcomes are live and externally resolved, exact deterministic reproduction is not possible; we report model versions, filtering rules, batch construction, catalog epochs, and metrics to support audit and comparison.

References

Rohan Alur, Bradly C. Stadie, Daniel Kang, Ryan Chen, Matt McManus, Michael Rickert, Tyler Lee, Michael Federici, Richard Zhu, Dennis Fogerty, Hayley Williamson, Nina Lozinski, Aaron Linsky, and Jasjeet S. Sekhon. Aia forecaster: Technical report, 2025. URL https://arxiv.org/abs/2511.07678.

Glenn W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, 1950. doi: 10.1175/1520-0493(1950)078⟨0001:VOFEIT⟩2.0.CO;2.

Jingyi Chai, Bingyang Zheng, Xiangrui Liu, Hao Lu, Zihang Zhou, Tianchen Wang, Kemeng Zhang, and Siheng Chen. Agentic time machine as an infrastructure for future-event forecasting, 2026. URL https://arxiv.org/abs/2606.21013.

Nikhil Chandak, Shashwat Goel, Ameya Prabhu, Moritz Hardt, and Jonas Geiping. Scaling open-ended reasoning to predict the future, 2026. URL https://arxiv.org/abs/2512.25070.

Pu Cheng, Juncheng Liu, and Yunshen Long. Polybench: Benchmarking llm forecasting and trading capabilities on live prediction market data, 2026. URL https://arxiv.org/abs/2604.14199.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

FutureSearch, Jack Wildman, Nikos I. Bosse, Daniel Hnyk, Peter Mühlbacher, Finn Hambly, Jon Evans, Dan Schwarz, and Lawrence Phillips. Bench to the future: A pastcasting benchmark for forecasting agents, 2025. URL https://arxiv.org/abs/2506.21558.

FutureX. Futurex live leaderboard, 2026. URL https://futurex.live/.

GLM-5-Team. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Danny Halawi, Fred Zhang, Chen Yueh-Han, and Jacob Steinhardt. Approaching human-level forecasting with language models, 2024. URL https://arxiv.org/abs/2402.18563.

Zhixin Han, Yanzhi Zhang, Chuyang Wei, Maohang Gao, Xiawei Yue, Kefei Chen, Yu Zhuang, Haoxiang Guan, Jiyan He, Jian Li, Yitong Duan, Yu Shi, Mengting Hu, and Shuxin Zheng. Futureworld: A live reinforcement learning environment for predictive agents with real-world outcome rewards, 2026. URL https://arxiv.org/abs/2604.26733.

Woojeong Jin, Rahul Khanna, Suji Kim, Dong-Ho Lee, Fred Morstatter, Aram Galstyan, and Xiang Ren. Forecastqa: A question answering challenge for event forecasting with temporal text data, 2021. URL https://arxiv.org/abs/2005.00792.

Kalshi. Kalshi, 2026. URL https://kalshi.com/.

Ezra Karger, Houtan Bastani, Chen Yueh-Han, Zachary Jacobs, Danny Halawi, Fred Zhang, and Philip E. Tetlock. Forecastbench: A dynamic benchmark of ai forecasting capabilities, 2025. URL https://arxiv.org/abs/2409.19839.

Amit Arnold Levy. Reinforcement learning for llm-based event forecasting, 2026. URL https://arxiv.org/abs/2606.15917.

Jiashuo Liu, Siyuan Chen, Zaiyuan Wang, Zhiyuan Zeng, Jiacheng Guo, Liang Hu, Lingyue Yin, Suozhi Huang, Wenxin Hao, Yang Yang, Zerui Cheng, Zixin Yao, Lingyue Yin, Haoxin Liu, Jiayi Cheng, Yuzhen Li, Zezhong Ma, Bingjie Wang, Bingsen Qiu, Xiao Liu, Zeyang Zhang, Zijian Liu, Jinpeng Wang, Mingren Yin, Tianci He, Yali Liao, Yixiao Tian, Zhenwei Zhu, Anqi Dai, Ge Zhang, Jingkai Liu, Kaiyuan Zhang, Wenlong Wu, Xiang Gao, Xinjie Chen, Zhixin Yao, Zhoufutu Wen, B. Aditya Prakash, Jose Blanchet, Mengdi Wang, Nian Si, and Wenhao Huang. Futurex-pro: Extending future prediction to high-value vertical domains, 2026. URL https://arxiv.org/abs/2601.12259.

Moonshot AI. Kimi k2.7 code. Hugging Face model card, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.7-Code.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

Polymarket. Polymarket, 2026. URL https://polymarket.com/.

Philipp Schoenegger, Cameron R. Jones, Philip E. Tetlock, and Barbara Mellers. Prompt engineering large language models' forecasting capabilities, 2025. URL https://arxiv.org/abs/2506.01578.

Udari Madhushani Sehwag, Elaine Lau, Haniyeh Ehsani Oskouie, Shayan Shabihi, Erich Liang, Andrea Toledo, Guillermo Mangialardi, Sergio Fonrouge, Ed-Yeremai Hernandez Cardona, Paula Vergara, Utkarsh Tyagi, Chen Bo Calvin Zhang, Pavi Bhatter, Nicholas Johnson, Furong Huang, Ernesto Gabriel Hernandez Montoya, and Bing Liu. Scipredict: Can llms predict the outcomes of scientific experiments in natural sciences?, 2026. URL https://arxiv.org/abs/2604.10718.

Sadanand Singh, Allam Reddy, and Manan Chopra. Verifiable rewards for calibrated probabilistic forecasting, 2026. URL https://arxiv.org/abs/2607.00164.

Zhengwei Tao, Pu Wu, Zhi Jin, Xiaoying Bai, Haiyan Zhao, Chengfeng Dou, Xiancai Chen, Jia Li, Linyu Li, Chongyang Tao, and Wentao Zhang. Prophet: An inferable future forecasting benchmark with causal intervened likelihood estimation, 2026. URL https://arxiv.org/abs/2504.01509.

Benjamin Turtel, Danny Franklin, Kris Skotheim, Luke Hewitt, and Philipp Schoenegger. Outcome-based reinforcement learning to predict the future, 2025. URL https://arxiv.org/abs/2505.17989.

Chuyang Wei, Maohang Gao, Zhixin Han, Kefei Chen, Yu Zhuang, Haoxiang Guan, Yanzhi Zhang, Yilin Cheng, Xiren Zhou, Huanhuan Chen, Jian Li, Jiyan He, Yu Shi, Yitong Duan, and Shuxin Zheng. Harnessing pre-resolution signals for future prediction agents, 2026. URL https://arxiv.org/abs/2604.15719.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026. URL https://arxiv.org/abs/2605.23904.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Zhiyuan Zeng, Jiashuo Liu, Siyuan Chen, Tianci He, Yali Liao, Yixiao Tian, Jinpeng Wang, Zaiyuan Wang, Yang Yang, Lingyue Yin, Mingren Yin, Zhenwei Zhu, Tianle Cai, Zehui Chen, Jiecao Chen, Yantao Du, Xiang Gao, Jiacheng Guo, Liang Hu, Jianpeng Jiao, Xiangsheng Li, Jingkai Liu, Shuang Ni, Zhoufutu Wen, Ge Zhang, Kaiyuan Zhang, Xin Zhou, Jose Blanchet, Xipeng Qiu, Mengdi Wang, and Wenhao Huang. Futurex: An advanced live benchmark for llm agents in future prediction, 2025. URL https://arxiv.org/abs/2508.11987.

Andy Zou, Tristan Xiao, Ryan Jia, Joe Kwon, Mantas Mazeika, Richard Li, Dawn Song, Jacob Steinhardt, Owain Evans, and Dan Hendrycks. Forecasting future world events with neural networks, 2022. URL https://arxiv.org/abs/2206.15474.

A Extended Method and Implementation Details

This appendix gives the full formalization and live-protocol details underlying the compact method description in Section 3.

A.1 Agent Tool Interfaces

In a full-skill rollout, the main agent accesses use_skill and worker; a worker accesses use_skill, search, and visit. The main agent uses use_skill with the thinking-skill directory and delegates a self-contained investigation through worker. A worker uses the working-skill directory, obtains web evidence through search and visit, and returns a structured report to the main agent. The abbreviated function schemas exposed to the agents are:

Tool schemas

use_skill

```
{
  "name": "use_skill",
  "description":
    "Load the full SKILL.md for a named local skill.",
  "parameters": {
    "type": "object",
    "properties": {
      "skill_name": {
        "type": "string",
        "description": "Selected skill name"
      },
      "action": {
        "type": "string",
        "enum": ["get_info"],
        "description": "Load the instruction body"
      }
    },
    "required": ["skill_name", "action"]
  }
}
```

worker

```
{
  "name": "worker",
  "description":
    "Dispatch an isolated worker for a scoped investigation.",
  "parameters": {
    "type": "object",
    "properties": {
      "task": {
        "type": "string",
        "description": "Self-contained investigation brief"
      },
      "domain": {
        "type": ["string", "array", "null"],
        "description": "Optional routing domain"
      }
    },
    "required": ["task"]
  }
}
```

search

```
{
  "name": "search",
  "description":
    "Run batched web searches and return result snippets.",
  "parameters": {
    "type": "object",
    "properties": {
      "query": {
        "type": "array",
        "items": {"type": "string"},
        "description": "Array of query strings"
      }
    },
    "required": ["query"]
  }
}
```

visit

```
{
  "name": "visit",
  "description":
    "Visit webpage(s) and return a goal-focused summary.",
  "parameters": {
    "type": "object",
    "properties": {
      "url": {
        "type": ["string", "array"],
        "items": {"type": "string"},
        "description": "One URL or an array of URLs"
      },
      "goal": {
        "type": "string",
        "description": "Extraction goal"
      }
    },
    "required": ["url", "goal"]
  }
}
```

A.2 Outcome-Realization Noise

Fix one rollout and its prediction-time market snapshot. Let $\mathcal{I}$ denote the information available to this rollout when it forecasts, and let Y denote the still-unresolved outcome. Its unobserved conditional outcome distribution is $\boldsymbol{\eta} = P(Y = \cdot \mid \mathcal{I})$, which is neither the agent nor market forecast. Let $\mathbf{e}_Y$ be the random one-hot vector for Y; the agent and market vectors $\mathbf{p}, \mathbf{m}$ are fixed by the rollout and snapshot. The realized alpha decomposes as

$$\alpha = \underbrace{\frac{1}{2}\left(\|\mathbf{m} - \boldsymbol{\eta}\|_2^2 - \|\mathbf{p} - \boldsymbol{\eta}\|_2^2\right)}_{\mu_\alpha\text{: conditional expected advantage}} + \underbrace{(\mathbf{p} - \mathbf{m})^\top(\mathbf{e}_Y - \boldsymbol{\eta})}_{\varepsilon_Y\text{: outcome realization noise}}. \tag{8}$$

Since $\mathbb{E}[\mathbf{e}_Y \mid \mathcal{I}] = \boldsymbol{\eta}$, the residual ε_Y has zero conditional mean and $\mu_\alpha = \mathbb{E}[\alpha \mid \mathcal{I}]$. However, $\boldsymbol{\eta}$ cannot be observed. The realized alpha contains outcome-realization noise, so a single realized score cannot be attributed to a helpful or harmful procedure.

A.3 Persistent State and Update Stages

Future-SE persists state alongside the deployed skill catalog: per-skill and catalog-level memory, maintained merits, watch items, and stage audit records. The following subsections document these state artifacts and the update stages.

A.3.1 Persistent State Artifacts

Filesystem layout. Future-SE separates deployable skill entries, persistent state retained across epochs, and epoch-local evidence produced by one update cycle. The resulting layout is:

Persistent filesystem layout

```
thinking_skills/
|-- MEMORY.jsonl
`-- <skill>/
    |-- SKILL.md
    |-- MEMORY.jsonl
    `-- MERIT.json
working_skills/
|-- MEMORY.jsonl
`-- <domain>/
    |-- MEMORY.jsonl
    `-- <skill>/
        |-- SKILL.md
        |-- MEMORY.jsonl
        `-- MERIT.json
runs/
`-- live_epoch<N>/
    |-- train/
    |   |-- predictions.jsonl
    |   `-- traces/<prediction-id>.json
    |-- observe.json
    |-- evolve_trace/<stage>_<timestamp>.jsonl
    |-- batch_metadata.json
    `-- evolve_state.json
```

The catalog directories persist across epochs. Each runs/live_epoch<N>/ directory records the resolved batch, its observation output, and the resulting update state for one evolution cycle.

Table 4 distinguishes per-skill and catalog-level decisions, maintained merits, and current-batch observations. The following boxes give abbreviated examples of the two MEMORY.jsonl forms, MERIT.json, and observe.json; each MEMORY.jsonl example is displayed as formatted JSON for readability, although each object occupies one line in storage. When reviewing a selected trajectory group, the observer emits the fault and merit labels defined below. Faults identify a problem in a deployed skill or the skill catalog, while merits record mechanisms judged to have helped.

Table 4: Evolution artifacts and their roles. Fields are abbreviated; optional fields depend on the action and its actual outcome.

Artifact	Contents	Role
SKILL.md	Name, description, instruction body.	Description supports routing; body provides the procedure used by the forecasting agent.
Skill MEMORY.jsonl	Epoch (bench), timestamp, stage, operation, status, rationale, watch items, signal summaries, actual changes/diffs, rejection information when relevant.	Append-only decision history, including skips and unsuccessful actions. Recent decisions and actual diffs help avoid repeating or reversing earlier repairs.
MERIT.json	{items: [{summary, score}, ...]}.	Compact, LLM-maintained list of useful mechanisms to preserve.
watch field	[{summary, advice}, ...] nested inside a memory record.	Follow-up concerns associated with a memory record.
Catalog MEMORY.jsonl	Epoch, stage, kind, domain/group, operation, status, rationale, actions with actual changes, watch items.	Creation/consolidation history for catalog coverage and boundaries. Working consolidation retrieves only its domain's records.
observe.json	Skill statistics, fault and merit occurrences, summaries, catalog gaps, relations, rollout metadata.	Observer outputs for the current batch, used as evidence by the update stages.

MEMORY.jsonl

Per-skill record.

```
{
  "bench": "live_epoch10",
  "skill_name": "direct-market-anchor",
  "stage": "create", "op": "create",
  "status": "accepted", "rationale": "...",
  "diff": {"edits": []}, "watch": []
}
```

Catalog-level record.

```
{
  "bench": "live_epoch10",
  "stage": "consolidate", "kind": "working",
  "domain": "sports", "op": "merge",
  "status": "accepted", "rationale": "...",
  "actions": [], "watch": []
}
```

MERIT.json

```
{
  "items": [
    {"summary": "Anchor to an exact matching prediction market",
     "score": 0.80}
  ]
}
```

observe.json

```
{
  "epoch": "live_epoch20",
  "counts": {"n_items_labeled": 30, "n_traces_missing": 0},
```

```
  "thinking_skills": {
    "<skill>": {
      "calls": 3,
      "fault_counts": {"routing_error.underselected": 3},
      "merit_counts": {}
    }
  },
  "catalog_gaps": [{
    "type": "catalog_gap", "subtype": "missing_capability",
    "kind": "working", "domain": "sports", "importance": "high"
  }],
  "catalog_relations": [{
    "type": "routing_error", "subtype": "peer_conflict",
    "kind": "thinking", "skills": ["<skill-a>", "<skill-b>"],
    "importance": "medium"
  }]
}
```

Observer Labels

Individual routing faults (Modify).

- routing_error.underselected: an applicable existing skill was not selected.
- routing_error.overselected: a selected skill was unnecessary for the trajectory.

Instruction faults (Modify).

- content_error.incomplete: the instruction omitted a needed procedure, check, or coverage condition.
- content_error.misleading: the instruction directed the rollout toward an inappropriate inference or evidence use.
- content_error.non_executable: the rollout did not follow or carry out the instruction as intended.

Catalog-level signals.

- catalog_gap.missing_capability (Create): no existing skill covered a needed capability.
- routing_error.peer_conflict (Consolidate): two skills overlapped or gave incompatible guidance.
- routing_error.fragmented_boundary (Consolidate): a procedure was split across skills with an unclear division of responsibility.

Merit labels.

- merit.pivotal: a mechanism was decisive to the useful forecasting behavior.
- merit.contributing: a mechanism materially supported the behavior without being decisive.

A.3.2 Stage Interfaces and Execution

Table 5 summarizes the default evolution path. The three editing stages draw on the observation artifact generated from the selected batch and inspect the skill catalog after earlier stages have acted.

The observer first labels main-agent reasoning and groups comparable worker investigations, then labels the corresponding worker traces; the resulting records form observe.json. Modify, Create, and Consolidate execute in that order, and later stages read the state left by earlier stages. The skill catalog is capped at 15 thinking skills and 10 working skills per domain, including domains created during evolution. In the reported configuration, Create can add at most three thinking skills and three working skills per domain in one update, and Consolidate can apply at most six actions per call. The editing stages receive up to eight recent memory records for each reviewed skill or skill-catalog group. Maintained merits may be updated even when no instruction edit is applied.

A.4 Skill Versioning and Delayed Attribution

Every prediction and trajectory stores the skill-catalog version that produced it. A delayed outcome may enter a later evolution batch, but its trace and score remain evidence about the rollout-time version rather than the skill catalog active when the outcome resolves. Resolved examples that

Table 5: Stage interfaces in the default evolution path. Main and worker observation are two passes within Observe.

Stage	Inputs	Outputs
Observe: main	Same-question trajectories, resolved scores, the main-agent trace, and its worker-delegation records.	Thinking-skill faults/merits, skill-catalog relations, and groups of comparable worker investigations for the second pass.
Observe: worker	Worker traces grouped by comparable investigation, with task and comparison context.	Working-skill faults/merits and skill-catalog signals. The two observation passes and skill-call counts are combined into the epoch-level observe.json record.
Modify	Current skill text, observation evidence, creation metadata, recent skill decisions/diffs, watch items, maintained merits, decision epoch.	Updates maintained merits and either skips or edits an existing skill; actual file changes are recorded in skill memory and stage traces.
Create	Eligible gap evidence, visible skill names/descriptions, current count, capacity, creation limit.	Bounded new skill files, per-skill creation records, and skill-catalog decision records.
Consolidate	Current group texts and creation metadata, compact observation summaries/merits, relations, maintained merits, skill-catalog history/watch, capacity.	Boundary edits, linked merges, or no action; per-skill and skill-catalog decision records record actual changes. Groups with fewer than two skills are skipped.

have not yet entered an update batch may be pooled across historical versions while retaining their original version identifiers.

Checking whether old evidence still applies. An old fault is used as the reason for a current repair only when its target skill existed when the rollout was generated and the current instruction still exhibits the problem. A fault that predates a skill's creation may provide background context, but cannot be attributed to that later-created skill. At review, the Modify stage receives the current skill text, creation metadata, and recent recorded decisions with their actual diffs, allowing it to determine whether an earlier repair already addressed the fault. In the example below, delayed epoch-2 faults prompt an edit at epoch 6 because the problematic wording remains, while similar delayed faults prompt no edit at epoch 8 because that wording has already been removed.

Example: delayed-fault review across two skill-evolution epochs

Skill: sports-match-predictions-previews

Epoch 6
Delayed fault. The Modify stage reviewed three faults from rollouts generated at epoch 2: one medium-importance routing_error.underselected fault, and two low-importance routing_error.overselected faults involving tournament-format rules and match-status checking.
Modify decision. It attributed the two overselection faults to the vague "or analysis" phrase in the description and accepted an edit that narrowed the description to expert predictions or match previews; the instruction body was retained.

Epoch 8
Delayed fault. The Modify stage again reviewed four routing_error.overselected faults, all delayed signals from rollouts generated at epoch 2 rather than new post-epoch-6 faults. They showed the same routing-failure pattern, including using this skill for standings, goal averages, and recent results rather than expert previews.
Modify decision. Because the epoch-6 edit had already removed the vague routing phrase and aligned the description with the skill body, the Modify stage judged the earlier repair sufficient and returned skip. No instruction change was made.

Table 6: Question characteristics of the two dynamic tracks. Topic and question-type entries are percentages within track.

	Evolution Track	Polymarket Evaluation Track
Topic	%	%
Entertainment	4.5	7.6
Environment	13.4	8.9
Finance	45.2	21.9
Politics	5.5	17.0
Sports	26.5	36.9
Technology	4.9	7.8
Question type		
Binary	37.5	100.0
Categorical	62.5	–
Prediction–resolution delay		
Median (hours)	41.0	27.9
IQR (hours)	31.3–48.3	25.3–40.9

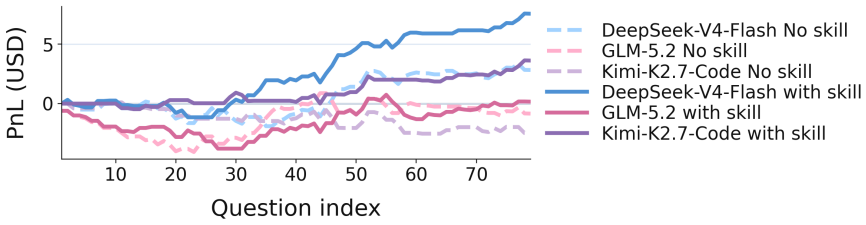


Figure 6: Fixed-one-share paper-trading diagnostic on Polymarket Run 1. Dashed lines are no-skill controls and solid lines are full-skill setups. Curves accumulate over a shared question order; questions without a positive-edge trade contribute zero PnL.

B Supplementary Evaluation and Evolution Diagnostics

This section provides supplementary descriptions of the two question streams and diagnostics that complement the main results.

B.1 Track Characteristics

Table 6 summarizes both tracks using the six topic families illustrated in Figure 1. The table also reports question types and the observed prediction-to-resolution delay.

B.2 Paper-Trading Diagnostic

Using Polymarket Run 1, we translate forecast differences into a decision-oriented quantity through a fixed-one-share paper simulation, following Outcome-RL (Turtel et al., 2025) in converting forecast probabilities into Yes/No trades. For a binary market, let m be the prediction-time market-implied Yes probability, which is also the Yes-share price; the No-share price is $1 - m$. With agent Yes probability p, realized outcome $y \in \{0, 1\}$, and fee $f = \$0.01$, a Yes share costs $m+f$ and pays y, while a No share costs $1 - m + f$ and pays $1 - y$. The strategy buys Yes when $p - m - f > 0$, buys No when $m - p - f > 0$, and otherwise does not trade. Curves accumulate PnL in a shared question order, with zero PnL for a question on which a setup does not trade.

The selected full-skill runs end at $\$7.56$ (DeepSeek), $\$0.16$ (GLM), and $\$3.62$ (Kimi), versus $\$2.83$, $-\$0.84$, and $-\$2.49$ for their no-skill controls. These absolute amounts are small because each trade buys one binary-market share priced below $\$1$ before fees. Thus, although profitability is never used for skill evolution, skill-library selection, or tuning, the evolved skills improve this fixed-rule trading diagnostic for all three models.

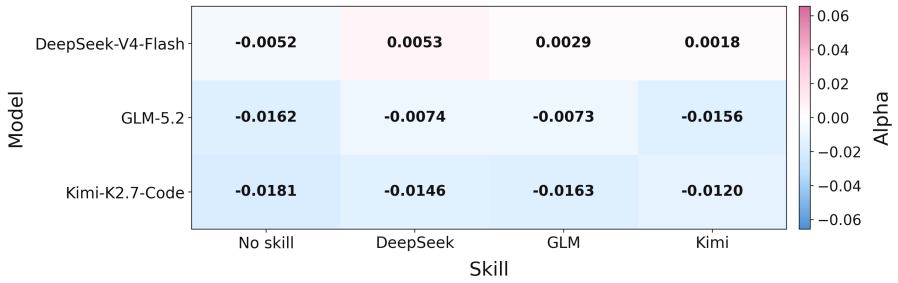


Figure 7: Cross-model frozen-skill analysis on Polymarket Run 2. Rows specify execution models and columns specify skill-catalog sources.

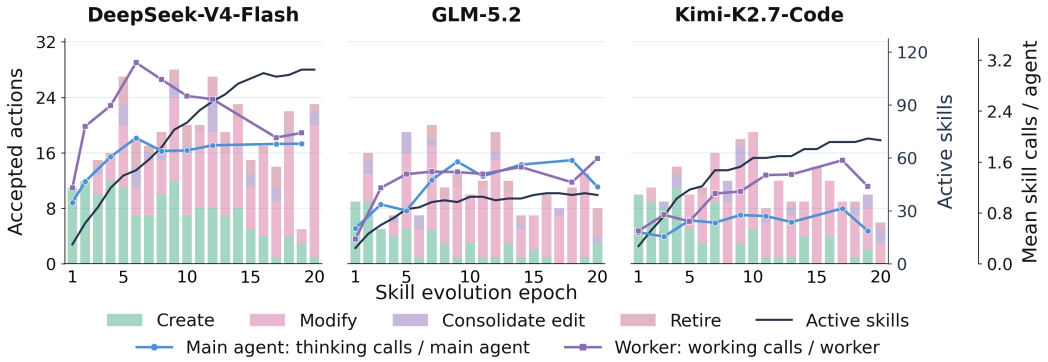


Figure 8: Evolution actions, active skill-catalog size, and observed skill use through epoch 20. Bars count accepted actions by type; black curves show active skills; colored curves show average thinking calls per main agent and working calls per worker.

B.3 Cross-Model Transfer

Using Polymarket Run 2, we evaluate fixed skill catalogs with each execution model. For a given execution model, the no-skill, own-skill, and cross-model skill configurations are evaluated on the same questions and rounds. For DeepSeek, the own-skill setup has alpha 0.0053, compared with -0.0052 without skills. For GLM, it improves from -0.0162 to -0.0073; for Kimi it improves from -0.0181 to -0.0120.

Across all three execution models, the own skill catalog yields the highest alpha. The gap is large for DeepSeek and Kimi, whereas GLM is nearly tied with DeepSeek's skill catalog. At the same time, all six cross-model configurations improve on the corresponding no-skill setup, showing that evolved forecasting procedures retain transferability even when the receiving model's own skill catalog is strongest.

B.4 Skill Catalog Dynamics

Figure 8 tracks accepted skill-catalog actions, active catalog size, and observed skill-call frequencies through epoch 20. Across all three models, catalog size grows rapidly early and stabilizes later. Observed thinking- and working-skill calls also settle into a narrower range across later deployed versions. Creation becomes less frequent, whereas later actions primarily modify, retire, or consolidate existing skills. The active catalogs nevertheless differ at epoch 20 (110, 39, and 70 skills for DeepSeek, GLM, and Kimi), showing that stabilization does not require a common final catalog size.

B.5 Observer Diagnostic Trends

Figure 9 shows a similar lifecycle pattern across the three models. Merit rates are generally higher at later displayed stages, while aggregate diagnostic-fault rates decline most sharply at the beginning and then fluctuate within a lower range. Early evidence is dominated by routing underselection and skill-catalog boundary relations, whereas content faults form a smaller but persistent component. This pattern is consistent with the skill catalog first resolving obvious selection and boundary problems, then refining existing procedures.

The plotted values divide observer-identified faults and merits by call opportunities at each lifecycle stage. A fault is not necessarily fatal: each rollout addresses a different question, and an improved skill catalog cannot guarantee appropriate selection and execution on every trajectory. Nonzero fault rates can therefore coexist with rising merit.

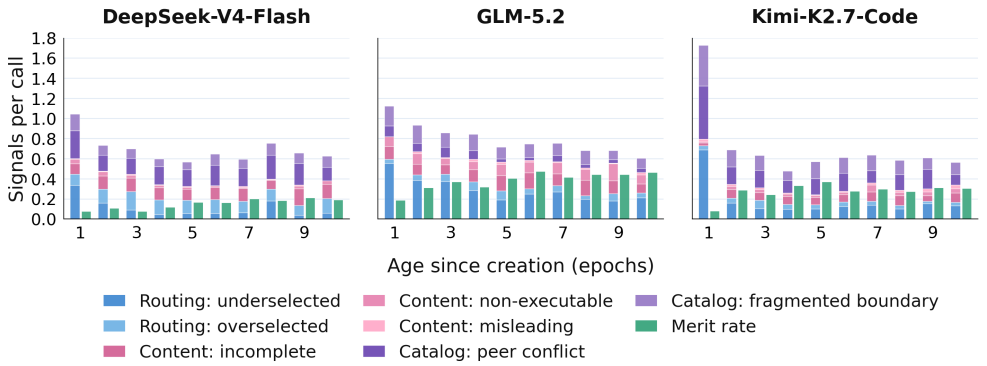


Figure 9: Observer diagnostic signals over the skill-catalog lifecycle through epoch 20. Stacked bars report routing, content, and skill-catalog-boundary signals normalized by call opportunities; green bars report merits using the same normalization.

C Case Study

This section gives two complementary case studies. The first follows how a high-merit skill changed across evolution epochs; the second examines the skill's realized effect in a matched skill/no-skill forecast, where the trajectories share the initial evidence but diverge after the skill is invoked.

C.1 Case Study 1: Skill Evolution

GLM-5.2's *market-anchored calibration* is the highest-ranked active GLM thinking skill in Table 3 (204 calls and a 0.49 merit-to-call ratio). The entries below summarize its accepted create, modify, and consolidation records through epoch 20.

Evolution of market-anchored calibration

Phase I: Acquired reusable workflow.

- **Epoch 1:** created a reusable market-implied prior for questions with prediction markets, betting odds, or expert consensus.
- **Epoch 2:** broadened routing and prioritized recent, liquid market evidence over narrative factors when sources conflict.
- **Epoch 5:** added a double-counting guard and required investigation when a divergent market is more directly relevant.
- **Epoch 7:** required direct prediction-market search, introduced a concrete evidence threshold for deviation, and consolidated an overlapping multi-source triangulation skill.
- **Epoch 11:** required each candidate adjustment to be labeled *priced* or *independent*, and made liquidity-backed market quotes dominate model or survey forecasts.
- **Epoch 12:** made an exact-match market primary over proxies and added extreme-anchor and small-sample caps.

Phase II: Persistent conditional-weighting difficulty.

- **Epoch 13:** replaced the rule to remain at the anchor whenever signals conflict with a net-balance rule that permits a bounded shift toward the stronger evidence side.
- **Epoch 17:** clarified that market price movements and odds shifts are already priced, after trajectories used them as independent deviation evidence.
- **Epoch 19:** added an empirical-divergence test, allowing recent, outcome-specific frequency data to count as independent evidence when it differs materially from the market anchor.

Final procedure.

1. Match the contract's resolution source and horizon to available markets or historical base rates.
2. Normalize comparable quotes, remove vig where applicable, and form a consensus range weighted by recency, relevance, and liquidity.
3. Use the exact-match, liquid market as the default anchor; treat proxy markets and narrative signals only as secondary evidence.
4. Label later evidence as already priced or plausibly independent, applying validity checks for scope, sample size, and extreme probabilities.
5. Move away from the anchor only by a bounded amount justified by the direction and strength of independent evidence, and document the resulting probability.

Interpretation. Phase I instantiates the reusable workflow identified by the merit analysis: align the target with a market or empirical prior, then constrain evidence-based adjustment. Phase II repeatedly refines the boundary between valid but conflicting evidence and the size of the resulting adjustment, illustrating the persistent conditional-weighting difficulty in Section 6.3.

C.2 Case Study 2: Skill effects on a live forecast

The second case study applies the market-anchored calibration skill described above to a live forecast. We compare two matched GLM-5.2 trajectories on the same question: one with the deployed skill catalog, including this skill, and one under the no-skill condition: "Will *I Knew It, I Knew You* by Taylor Swift be the Billboard Hot 100 #2 song for the week of August 22, 2026?"

Matched trajectory: common setup → divergent synthesis

Common front half (same question and initial evidence).
Both forecasts started from the same chart state: on the August 15, 2026 Hot 100, *I Knew It, I Knew You* sat at #5, down from #3 the prior week, behind *Choosin' Texas*, two Ariana Grande entries, and a Morgan Wallen song. The August 22 chart was imminent. Neither forecast had evidence that a rebound had already occurred; the open question was whether the song could return to #2 within a single tracking week.

Forecast with skill
Use market-anchored calibration.
The main agent dispatched workers to investigate the question. They surfaced the exact Polymarket contract at 73% Yes and a newer midweek report that the song was vying for an eighth week inside the top two. The agent took the recent decline to be already reflected in the market odds, kept the liquid exact-match market as the primary anchor, and applied only a bounded downward adjustment for lingering uncertainty about the final tracking week.

Forecast: Yes 0.68, No 0.32. ✓
Resolved outcome: Yes.
Normalized score: 0.8976.

Forecast without skill
No skill was invoked. The agent emphasized the fall from #3 to #5, the strength of the songs above it, and the absence of a known catalyst, then extrapolated the declining trajectory into an almost-certain No.

Forecast: Yes 0.03, No 0.97. ×
Resolved outcome: Yes.
Normalized score: 0.0591.

Interpretation. The skill did not change the observed chart facts or introduce decisive new evidence; it changed the burden of proof for deviating from a liquid exact-match market. The forecast with skill recognized that the recent decline was already reflected in the market odds at 73% Yes. Although the newer midweek report pointed upward, residual uncertainty about the final tracking week justified a bounded downward adjustment to 0.68, keeping the forecast near the anchor. The forecast without skill treated a single-week decline as decisive and issued an extreme forecast against the market. The realized outcome was Yes, giving 0.8976 for the forecast with skill versus 0.0591 for the forecast without skill.

D PROMPT TEMPLATES

This section provides the prompt templates used across all of our skill-evolution experiments; placeholders are instantiated at run time.

Main-agent prompt

System template.

```
You are a helpful assistant good at tool-using.
Today is {today} in the Asia/Shanghai timezone.
You may issue at most ${max_tool_count} tool calls in total. Each tool invocation counts
separately, even if several are issued in the same turn.
```

User template.

```
${question}
```

Worker prompt

System template.

```
You are an isolated investigator subagent for a forecasting pipeline. The main agent
dispatched you with a free-form `task`. Investigate with the tools available and return
only concrete findings, no commentary.

# Hard rules

1. You may issue at most ${max_tool_count} tool calls in total. Each tool invocation
counts separately, even if several are issued in the same turn.
2. Each finding MUST cite a `source` for provenance (URL + the shortest useful verbatim
snippet).
3. Keep findings directly supported by evidence and concise.

Length is content-driven. Multi-line YAML strings use the literal-block `|` style if the
fact contains colons or quotes.

# Output

Follow a specific output shape only when the dispatched task explicitly asks for one.
Otherwise, end your reply with this generic findings YAML block and put no text after it:

```yaml
findings:
  - fact: |
      <concrete fact>
    source: "https://... (URL + minimal verbatim snippet)"
metacognition: "<one short sentence on what could be wrong or what's missing>"
```

Reminders

- Stay focused on the dispatched task.
- Keep the final YAML compact: at most 6 findings, each fact no more than 3 sentences.
- Emit exactly the schema you chose and no other output beyond reasoning before the YAML.
```

User template.

```
# Task
${task}
```

Main-observer prompt

System template.

```
You are a fault and merit analyst for K parallel trajectories of one prediction question.

Each trajectory includes the main trace, worker calls, `main.score`,
`main.market_score`, `main.alpha_vs_market`, and one shared ground truth. Some
trajectories may be `rollout_kind=no_skill`.

Return exactly one JSON object, with no prose, markdown, or code fences. Use these
fields in this order:

```jsonc
{
  "worker_groups": [],
  "workers": [],
  "main_faults": [],
  "main_merits": [],
  "catalog_relations": []
}
```

`worker_groups`

- Fields:
  - `group_id`: stable id inside this output, e.g. `g1`.
  - `canonical_task`: one normalized short description of the shared investigation
  question.
  - `members`: array of `{traj, worker_call_id}` objects.
    - `traj`: input trajectory index.
    - `worker_call_id`: id of the worker tool call made by main.
- Group worker calls that asked essentially the same investigation question about the
same target object for the same downstream use.
- Every worker call must appear in exactly one group.
- Emit singleton groups when no comparable worker exists.
- By default, one group has at most K members: at most one worker from each trajectory.
- Only put two workers from the same trajectory in one group when they are true retries
or near-duplicates of the same investigation.
- Keep `canonical_task` narrow enough that pass2 can compare the members directly.

`workers`

- Fields:
  - `traj`: input trajectory index.
  - `worker_call_id`: id of the worker tool call made by main.
  - `outcome`: `productive`, `neutral`, or `harmful`.
  - `reason`: one specific sentence.
- One row per worker call.
- Every worker call must appear exactly once.
- Outcome must be `productive`, `neutral`, or `harmful`.
  - `productive`: the worker output advanced main's answer path.
  - `neutral`: the output neither helped nor hurt.
  - `harmful`: the output plausibly misled main.
- Give one specific reason sentence for each worker.
- `worker_call_id` is the id of the specific worker tool call made by main.

`main_faults`

- Fields on each row:
  - `type`: broad family, one of `routing_error`, `content_error`, or `catalog_gap`.
  - `subtype`: exact label within that family.
  - `traj`: evidence trajectory index. For winner-derived rows, use the good trajectory.
  - `skill_name`: existing thinking_skill name. Omit it for
  `catalog_gap.missing_capability`.
  - `skill_call_id`: id of the invoked skill call. Omit it for
  `routing_error.underselected` and `catalog_gap.missing_capability`.
  - `importance`: consequence size, one of `high`, `medium`, or `low`.
  - `reason`: concrete step, failed mechanism, and why it affected the answer path.
- Use `main.alpha_vs_market` as the primary signal-strength context. It is `main.score -
main.market_score`; positive means the trajectory beat its own prediction-time market
snapshot. Do not emit a fault merely because `score` is low, and do not force
attribution on high-randomness questions when no concrete catalog issue is visible.
- Compare K trajectories to locate the divergent step. When a stronger sibling or
no-skill trajectory handled the same step better, put that counterfactual in `reason`;
downstream needs to see what should have happened.
- `no_skill` is only a counterfactual baseline: use it in `reason` when it shows what a
skill-enabled path missed, but never output it as `skill_name` or as a direct
fault/merit target.
```

```
- For ordinary bad-trajectory faults, emit one row per trajectory occurrence. If the
same fault appears in two bad trajectories, emit two rows.
- From good trajectories, extract reusable edit/create signals: if a clearly stronger
trajectory used a generalizable reasoning pattern that weaker siblings or no-skill
missed, emit one row for that winning pattern, not one row per loser. Use `importance:
"high"` when that pattern explains a material winner/loser gap.
  - If an existing thinking_skill description matches the pattern, emit
  `content_error.incomplete` or `content_error.non_executable` for that skill. Use the
  good trajectory as `traj`, and explain what rule/structure the skill should add or
  force.
  - If no existing thinking_skill description matches the pattern, emit
  `catalog_gap.missing_capability`. Use the good trajectory as `traj`, and state the
  missing capability requirement without designing the full skill.
- Use only these labels:
  - `routing_error.overselected`
  - `routing_error.underselected`
  - `content_error.misleading`
  - `content_error.incomplete`
  - `content_error.non_executable`
  - `catalog_gap.missing_capability`
- `skill_call_id` is required for invoked-skill faults; omit it for
`routing_error.underselected` and `catalog_gap`.
- Set `importance` by observed consequence size: `high` for a signal that likely
explains a large answer/alpha difference, `medium` for a clear but limited consequence,
`low` for a small or uncertain consequence.
- `routing_error.overselected`: main called a thinking_skill that does not fit the
scenario. Explain what the skill covers, what the step needed, and why the mismatch
mattered.
- `routing_error.underselected`: at a specific step, main missed an existing
thinking_skill whose description covers the required capability. Name the step, the
required capability, and the matching description phrase. Do not include `skill_call_id`.
- `content_error.misleading`: the invoked thinking_skill content led main toward a wrong
or weak step. Cite the misleading line/substance and the concrete misdirection.
- `content_error.incomplete`: the right skill was called, but its body missed the needed
rule or framed it for the wrong shape. State the missing/framing requirement.
- `content_error.non_executable`: the right rule exists in the body, but the trace shows
no execution of it. State which rule was skipped and how that affected the answer path.
- `catalog_gap.missing_capability`: no existing thinking_skill description fits the
needed reasoning step. Omit `skill_name`.

## `main_merits`

- Fields on each row:
  - `type`: `merit.pivotal` or `merit.contributing`.
  - `traj`: trajectory index.
  - `skill_name`: existing thinking_skill that helped.
  - `skill_call_id`: id of the invoked skill call.
  - `reason`: concrete evidence sentence.
- Merits are a preservation signal for consolidate, not a modify_del trigger.
- Be stricter when `main.alpha_vs_market` is weak or negative; prefer empty merits
unless the skill clearly mattered.
- One row per trajectory occurrence; do not collapse across trajectories.
- Use only:
  - `merit.pivotal`
  - `merit.contributing`
- `merit.pivotal`: the skill mechanism materially changed the answer path.
- `merit.contributing`: the mechanism helped but was not the central driver.
- Do not emit both merit types for the same `skill_call_id`.
- Only attribute merit when a specific thinking_skill mechanism visibly helped the path
to the answer.
- `no_skill` is comparison-only and never a catalog object.

## `catalog_relations`

- Fields on each row:
  - `type`: always `routing_error`.
  - `subtype`: `peer_conflict` or `fragmented_boundary`.
  - `skills`: existing thinking_skill names involved in the relation.
  - `importance`: catalog-level consequence size, one of `high`, `medium`, or `low`.
  - `reason`: concrete evidence for why the relation exists.
- Use only when at least two existing thinking skills have a structural catalog problem:
  - `peer_conflict`: the skills compete for the same routing scenario.
  - `fragmented_boundary`: one reusable capability is split across the skills.
- Put the concrete shared scenario or split capability directly in `reason`; do not add
extra fields for it.
- Set `importance` by catalog-level consequence: `high` when the relation likely causes
repeated major errors, `medium` for a clear local conflict, `low` for a minor or
uncertain effect.
```

- Do not use `catalog_relations` for ordinary single-skill faults; those stay in `main_faults`.
- Emit one row per distinct relation. If several trajectories show the same relation, summarize the strongest evidence in one `reason`.

Output schema

```json
{
  "worker_groups": [
    {
      "group_id": "g1",
      "canonical_task": "recover the official source for the reported event",
      "members": [
        {"traj": 0, "worker_call_id": "call_worker_aaa"},
        {"traj": 1, "worker_call_id": "call_worker_bbb"}
      ]
    }
  ],
  "workers": [
    {
      "traj": 0,
      "worker_call_id": "call_worker_aaa",
      "outcome": "productive",
      "reason": "Opened the primary source and extracted the exact label."
    }
  ],
  "main_faults": [
    {
      "type": "content_error",
      "subtype": "incomplete",
      "traj": 0,
      "skill_name": "exact-market-anchor-audit",
      "skill_call_id": "call_skill_123",
      "importance": "medium",
      "reason": "The body needs a rule about re-checking the source page before trusting a proxy summary."
    }
  ],
  "main_merits": [
    {
      "type": "merit.pivotal",
      "traj": 1,
      "skill_name": "exact-market-anchor-audit",
      "skill_call_id": "call_skill_456",
      "reason": "The exact-source check materially changed the final answer path."
    }
  ],
  "catalog_relations": [
    {
      "type": "routing_error",
      "subtype": "peer_conflict",
      "skills": ["exact-market-anchor-audit", "proxy-summary-triangulation"],
      "importance": "medium",
      "reason": "Both skills try to solve the same exact-source anchoring task."
    }
  ]
}
```

User template.

```
# Available thinking_skills (visible to main for this question)
{thinking_catalog}

# Ground truth
{ground_truth}

# K trajectories (K = {k})

{trajectories_block}
```

Worker-observer prompt

System template.

````
You are a fault and merit analyst for one worker function group across K parallel
trajectories of the same prediction question.

Each member includes the id of the specific worker tool call made by main, complete
ReAct messages, prediction-time `score`, `market_score`, `alpha_vs_market`, and the
main-assigned outcome (`productive` / `neutral` / `harmful`). The group may have fewer
than K members when some trajectories did not dispatch this worker function.

Return strict JSON with these fields in this order:

```jsonc
{
  "faults": [],
  "merits": [],
  "catalog_relations": []
}
```

## `faults`

- Fields on each row:
  - `type`: broad family, one of `routing_error`, `content_error`, or `catalog_gap`.
  - `subtype`: exact label within that family.
  - `traj`: trajectory index.
  - `worker_call_id`: id of the specific worker tool call made by main.
  - `skill_name`: existing working_skill name. Omit it for
  `catalog_gap.missing_capability`.
  - `skill_call_id`: id of the worker's internal `use_skill` call. Omit it for
  `routing_error.underselected` and `catalog_gap.missing_capability`.
  - `domain`: only for `catalog_gap.missing_capability`.
  - `importance`: consequence size, one of `high`, `medium`, or `low`.
  - `reason`: concrete worker step, failed mechanism, and why it affected the worker
  result.
- One row per trajectory occurrence; do not collapse across members.
- `no_skill` is comparison-only and never a catalog object.
- Use only:
  - `routing_error.overselected`
  - `routing_error.underselected`
  - `content_error.misleading`
  - `content_error.incomplete`
  - `content_error.non_executable`
  - `catalog_gap.missing_capability`
- `worker_call_id` is the id of the specific worker tool call made by main.
- `skill_call_id` identifies the worker's internal `use_skill` call.
- Set `importance` by observed consequence size: `high` for a fault that likely caused a
large worker-output or answer/alpha difference, `medium` for a clear but limited
consequence, `low` for a small or uncertain consequence.
- For `catalog_gap.missing_capability`, include `domain` as `general`, `finance`,
`politics`, `sports`, `entertainment`, `technology`, or `environment`.
- `reason` must state the concrete worker step, failed mechanism, and why it mattered.
- Do not force a fault on productive or neutral members; harmful members still need a
concrete taxonomy fit.

## `merits`

- Fields on each row:
  - `type`: `merit.pivotal` or `merit.contributing`.
  - `traj`: trajectory index.
  - `worker_call_id`: id of the specific worker tool call made by main.
  - `skill_name`: existing working_skill that helped.
  - `skill_call_id`: id of the worker's internal `use_skill` call.
  - `reason`: concrete evidence sentence.
- One row per trajectory occurrence; do not collapse across members.
- Only consider merits when the member's `main_outcome=productive`.
- Use only:
  - `merit.pivotal`
  - `merit.contributing`
- `merit.pivotal`: the skill mechanism materially changed the worker result.
- `merit.contributing`: the mechanism helped but was not the central driver.
- Do not emit both merit types for the same `skill_call_id`.
- Only attribute merit when a specific working_skill mechanism visibly helped the
worker's path.
- `no_skill` is comparison-only and never a catalog object.

## `catalog_relations`

- Fields on each row:
  - `type`: always `routing_error`.
````

```
  - `subtype`: `peer_conflict` or `fragmented_boundary`.
  - `skills`: existing working_skill names involved in the relation.
  - `importance`: catalog-level consequence size, one of `high`, `medium`, or `low`.
  - `reason`: concrete evidence for why the relation exists.
- Use only when at least two existing working skills have a structural catalog problem:
  - `peer_conflict`: the skills compete for the same routing scenario.
  - `fragmented_boundary`: one reusable capability is split across the skills.
- Put the concrete shared scenario or split capability directly in `reason`; do not add
extra fields for it.
- Set `importance` by catalog-level consequence: `high` when the relation likely causes
repeated major errors, `medium` for a clear local conflict, `low` for a minor or
uncertain effect.
- Do not use `catalog_relations` for ordinary single-skill faults; those stay in `faults`.
- Emit one row per distinct relation. If several trajectories show the same relation,
summarize the strongest evidence in one `reason`.

## Output schema

```json
{
  "faults": [
    {
      "type": "routing_error",
      "subtype": "overselected",
      "traj": 1,
      "worker_call_id": "call_worker_aaa",
      "skill_name": "example-skill",
      "skill_call_id": "call_skill_aaa",
      "importance": "medium",
      "reason": "..."
    },
    {
      "type": "content_error",
      "subtype": "incomplete",
      "traj": 0,
      "worker_call_id": "call_worker_bbb",
      "skill_name": "example-skill",
      "skill_call_id": "call_skill_bbb",
      "importance": "high",
      "reason": "..."
    },
    {
      "type": "catalog_gap",
      "subtype": "missing_capability",
      "traj": 2,
      "worker_call_id": "call_worker_ccc",
      "domain": "sports",
      "importance": "medium",
      "reason": "..."
    }
  ],
  "merits": [
    {
      "type": "merit.pivotal",
      "traj": 1,
      "worker_call_id": "call_worker_ddd",
      "skill_name": "example-skill",
      "skill_call_id": "call_skill_ddd",
      "reason": "..."
    }
  ],
  "catalog_relations": [
    {
      "type": "routing_error",
      "subtype": "peer_conflict",
      "skills": ["skill-a", "skill-b"],
      "importance": "medium",
      "reason": "..."
    }
  ]
}
```
```

User template.

```
# Available working_skills (visible to workers of domain={worker_domain})
{worker_catalog}

# Worker function group
group_id = {group_id}
```

```
canonical_task = {canonical_task}
n_members = {n_members}   # out of K = {k} trajectories that dispatched this function

# Group members

{group_members_block}
```

Thinking-skill Create prompt

System template.

```
You build new thinking_skills for a prediction skill catalog.

Return strict JSON only. Create at most {max_creates_per_kind} skills. If no gap is
worth a permanent skill, return `creates: []`.

## Inputs

The user message contains:

- Existing thinking_skills: current names and descriptions. Do not duplicate names or
capabilities already covered here.
- Capacity: current count and remaining slots.
- catalog_gap.missing_capability items: observe signals saying no existing
thinking_skill covered a reusable reasoning need.

## What to Create

A thinking_skill is a reusable reasoning framework for prediction work. It can guide
what information to seek, organize evidence, weigh uncertainty, or structure the final
probability judgment.

Create only when the gap is reusable, actionable, and not already covered by an existing
description. Merge equivalent gap reasons into one skill. Prefer `importance: high`,
then `medium`; treat `low` as weak support.

Do not create for:

- one-off topic facts or candidate lists;
- routing or content problems in an existing skill;
- catalog boundary conflicts between existing skills;
- a full catalog with no remaining capacity.

## SKILL.md Rules

- `name`: lower-case-with-dashes, unique, and specific to the reasoning framework.
- `skill_md` starts with YAML frontmatter:
  ```markdown
  ---
  name: <same as outer name>
  description: "<positive use cue>"
  ---
  ```
- `description` states when to use the skill. Keep it positive; do not write "not for X"
blacklists. Always quote it.
- Body is self-contained instruction: what the framework is, when to apply it, steps to
follow, and what to produce.
- Keep the full SKILL.md at or below 120 lines.

Hard Constraints

- `extra_files` must be `[]`.
- Do not mention main agent, worker, dispatch, catalog, prompt, or trace inside SKILL.md.
- Do not reference other skills inside SKILL.md.
- Do not write revision history inside SKILL.md.
- Do not hardcode URLs, dates, prices, thresholds, named candidates, or baked-in answers.

Output

```json
{
  "creates": [
    {
      "name": "<lower-case-with-dashes>",
      "skill_md": "<full SKILL.md text>",
      "extra_files": [],
      "rationale": "<which gap cluster this covers and why it deserves a skill>"
```
```

```
    }
  ]
}
```

User template.

```
# Existing thinking_skills
n_existing: {n_existing}

{existing_skills_section}

# Capacity
{capacity_section}

# catalog_gap.missing_capability
n_catalog_gaps: {n_catalog_gaps}

{catalog_gaps_section}

Return JSON only, following the output contract in the system prompt.
```

Working-skill Create prompt

System template.

```
You build new working_skills for one prediction domain.

Return strict JSON only. This call's target domain is `{domain}`. Create at most
{max_creates_per_kind} skills, all in exactly that domain. If no gap is worth a
permanent skill, return `creates: []`.

## Inputs

The user message contains:

- Existing working_skills: names and descriptions visible to this domain at runtime:
`general` plus `{domain}`. Do not duplicate capabilities already covered here.
- Capacity: current count and remaining slots for `{domain}`.
- catalog_gap.missing_capability items: only `{domain}`'s observe signals saying no
visible working_skill covered a reusable information-gathering or
information-structuring need.

## What to Create

A working_skill is a reusable procedure for gathering, checking, or structuring external
information needed for prediction work.

Create only when the gap is reusable, actionable, and not already covered by an existing
description. Merge equivalent gap reasons into one skill. Prefer `importance: high`,
then `medium`; treat `low` as weak support.

Do not create for:

- one-off topic facts or candidate lists;
- routing or content problems in an existing skill;
- catalog boundary conflicts between existing skills;
- a domain whose capacity is full.

## SKILL.md Rules

- `domain`: must be exactly `{domain}`. Do not create a general or cross-domain skill
from this call.
- `name`: lower-case-with-dashes, unique, and specific to the procedure.
- `skill_md` starts with YAML frontmatter:
  ```markdown
  ---
  name: <same as outer name>
  description: "<positive use cue>"
  ---
  ```
- `description` states when to use the skill. Keep it positive; do not write "not for X"
blacklists. Always quote it.
- Body is self-contained instruction: search/visit approach, checks, required fields,
and final deliverable.
- Keep the full SKILL.md at or below 120 lines.

Hard Constraints
```

```
- `extra_files` must be `[]`.
- Do not mention main agent, worker, dispatch, catalog, prompt, or trace inside SKILL.md.
- Do not reference other skills inside SKILL.md.
- Do not write revision history inside SKILL.md.
- Do not hardcode URLs, API endpoints, dates, prices, thresholds, named candidates, or
baked-in answers.

## Output

```json
{
  "creates": [
    {
      "domain": "general | finance | politics | sports | entertainment | technology |
      environment",
      "name": "<lower-case-with-dashes>",
      "skill_md": "<full SKILL.md text>",
      "extra_files": [],
      "rationale": "<which gap cluster this covers and why it deserves a skill>"
    }
  ]
}
```
```

User template.

```
# Target working domain
{domain}

# Existing working_skills
n_existing: {n_existing}

{existing_skills_section}

# Capacity
{capacity_section}

# catalog_gap.missing_capability
n_catalog_gaps: {n_catalog_gaps}

{catalog_gaps_section}

Return JSON only, following the output contract in the system prompt.
```

Modify prompt

System template.

```
You maintain one SKILL.md package in a prediction skill catalog. Each call reviews
exactly one current skill and returns `modify`, `delete`, or `skip`.

Default to `skip`. Modify or delete only when the evidence points to a specific current
problem, or when the current SKILL.md has an obvious defect you can name in one sentence.

## Inputs
The user message contains:

- Skill identity: `name` and `kind`.
  - `created_at_epoch`: the epoch that first created this skill when known, or `unknown`
  when no create record was found. Do not infer that an `unknown` skill is new.
- Epoch context:
  - `epoch`: the decision epoch for this batch. In live training, the signal rows you
  see may come from earlier rollout epochs; use `rolled_out_at_epoch` / `by_epoch` to
  see which rollout epoch each row came from. The skill used in an earlier epoch may not
  be the same as the current SKILL.md.
- Skill signals: one prompt-ready observe row for this skill.
  - `signal_summary`: `calls`, `n_called`, `actions`, fault/merit counts and rates, plus
  `by_epoch`.
  - `faults` / `merits`: all current evidence rows. Each preserves its decision-useful
  fields, including `qid` and task/call context when available; evidence text is not
  character-truncated.
- Decision history: recent prior modify/delete/skip records for this skill, including
rationale, outcome, and unified diffs for actual SKILL.md edits. Historical signal
summaries are not repeated here; use the current Skill signals section for evidence.
- Recent watch: the watch from the latest memory entry, shown separately from ordinary
history.
```

```
- Current MERIT.json: the full preservation list for mechanisms that should not be
broken casually.
- Current SKILL.md: the only editable file. Read it every round, even when you skip.

## How to Read Evidence
- Use `signal_summary.fault_counts`, `fault_rate`, and `faults[*].reason` to locate the
root cause. Prefer rates over raw counts when call volume differs.
- Use `merits[*]` and MERIT.json as preservation pressure, not as a reason to churn. A
useful skill needs stronger fault evidence before modification.
- `importance` is consequence size: `high` means the issue likely changed the answer
path materially, `medium` means a clear limited effect, `low` means small or uncertain
effect.
- The current SKILL.md is the active version being evaluated. Evidence from older
`rolled_out_at_epoch` values may describe older skill text, so verify the described
pattern still exists before editing.
- If a signal's `rolled_out_at_epoch` is before `created_at_epoch`, do not treat it as
this skill's failure or merit. Use it only as background context or a watch item.
- A fault from an older epoch is still actionable if the current SKILL.md still contains
the problem. If the current SKILL.md already plausibly fixed it, skip that fault and say
so.
- Do not revert a recent accepted patch solely because an older-epoch fault describes
pre-patch behavior.
- Watch is a reminder, not an obligation. Act on it only if current evidence makes it
actionable; otherwise keep watching or ignore it with a reason.

## Actions
### `skip`

Choose `skip` when no specific current fix is justified. Common reasons:

- no clear fault-traced fix and no obvious current defect;
- one-off or noisy signal;
- current SKILL.md already fixes the visible fault;
- recent accepted modifies have not had time to settle;
- the skill has meaningful merits and the proposed edit might damage working mechanisms.

### `modify`

Choose `modify` only when one of these triggers clearly applies.

**Reactive fault fix:** a current fault cluster maps to a specific section of SKILL.md.

- `routing_error.overselected`: tighten the description. Touch the body only if it also
teaches the wrong scenario.
- `routing_error.underselected`: improve discoverability. If the body already supports
the needed capability, edit only the description. If the body does not support it,
either add the missing in-scope capability or narrow the description so it stops
over-promising.
- `content_error.misleading`: fix the specific misleading rule or framing.
- `content_error.incomplete`: add or refactor the missing rule, case split, field, or
framing near the failure mode.
- `content_error.non_executable`: do not just reword the rule. Convert the affected
section into a checklist, required fields, small case table, or forced Q&A that the
reader must execute.

`catalog_gap` belongs to create. Catalog-level `peer_conflict` and `fragmented_boundary`
belong to consolidate.

**Proactive fix:** the current SKILL.md has a concrete defect visible from the text
itself:

- description does not match the body;
- internal contradiction;
- wrong field name, endpoint shape, or other self-contained factual mismatch;
- repeated local exceptions or stale patch residue make the intended path hard to
execute;
- useful guidance is buried in overlong prose and should become a compact checklist,
case split, ledger, or required output shape.

Do not modify for style preferences, extra examples, or vague clarity improvements.

### `delete`

Choose `delete` only when the skill is beyond a local repair:

- at least three current `content_error.misleading` faults show the skill teaches the
wrong path;
- the capability is wrong by design for the observed failures;
```

```
- many calls plus scattered repeated faults show sustained negative value with no clear
fix point;
- across visible history it has persistently low/zero calls, no merit, unsuccessful
modifications, and no independent value in the current SKILL.md.

Do not delete only because one epoch was cold.

## Editing Rules

- Touch only 1-2 high-confidence points per epoch. Large rewrites are allowed only when
repeated small edits have failed or the skill is too overgrown to execute; say that in
`rationale`.
- If editing an overlong skill, compress the affected section instead of appending
another exception. Aim for SKILL.md to stay at or below 120 lines.
- Preserve MERIT.json mechanisms unless a fault directly contradicts them. If an edit
weakens a high-score item, explain why.
- Do not write architecture terms inside SKILL.md: no main agent, worker, dispatch,
catalog, prompt, or trace language.
- Do not write cross-skill references inside SKILL.md.
- Do not write revision history inside SKILL.md.
- Do not hardcode fixed URLs, dates, prices, thresholds, named candidates, or baked-in
answers.
- In frontmatter, always wrap `description` in double quotes.
- The description says when to use the skill. Do not add "not for X" blacklists.

## MERIT.json

Always emit `merit_items`, even for `skip`.

- Treat current MERIT.json as the full preservation list.
- Add or raise an item when this round's merits support a reusable mechanism.
- Leave scores unchanged when this round says nothing about the mechanism.
- Lower or remove an item only when a fault directly contradicts it. Remove items whose
score reaches 0.
- Each reusable mechanism must appear exactly once. Do not emit duplicate summaries or
paraphrased duplicates of the same mechanism.
- Each item is exactly `{"summary": "...", "score": <0.0-1.0>}` with at most two decimal
places.

## Output

Return strict JSON only, with no markdown fence:

```json
{
  "action": "modify | delete | skip",
  "edits": [
    {"path": "SKILL.md", "new_content": "<full SKILL.md text, including frontmatter>"}
  ],
  "rationale": "<one or two sentences explaining the evidence and chosen action>",
  "merit_items": [
    {"summary": "<reusable mechanism to preserve>", "score": 0.0}
  ],
  "watch": [
    {
      "summary": "<optional unresolved observation>",
      "advice": "<optional next-round advice>"
    }
  ]
}
```

`edits` is required only for `modify`, must contain exactly one full rewrite of
`SKILL.md`, and is forbidden for `skip` or `delete`. For `delete`, `rationale` must
explain why the skill cannot be locally repaired. For `skip`, `rationale` must explain
why no current edit is justified.
```

User template.

```
# Skill
name: {skill_name}
kind: {kind}
created_at_epoch: {created_at_epoch}

# Epoch
epoch: {epoch}

# Skill signals
```json
{skill_row_json}
```
```

````
```

# Decision History
{decision_history_section}

# Recent Watch
{watch_history_section}

# Current MERIT.json
{merit_state_section}

# Current SKILL.md
```markdown
{skill_md_text}
```

Return JSON only, following the output contract in the system prompt.
````

Consolidate prompt

System template.

```
You maintain catalog health for a prediction skill library. Each call sees one group:
all thinking_skills, or all working_skills inside one domain.

Default to `actions: []`. Act only when catalog-level evidence supports merge, boundary
tightening, or cleanup.

## Inputs

The user message contains:

- Group: optional working `domain` (use `(n/a)` for thinking) and `n_skills`.
- Cap: target skill count for this group.
- Epoch: current decision epoch.
- Catalog Memory Tail: recent create/consolidate decisions for this group. Each action
explicitly names its execution op; accepted creates and updates include full unified
diffs, while deletes identify the retired skill. Historical signal summaries are not
repeated here; use the current Skills and Catalog Relations sections for evidence.
- Recent Watch: the latest watch item from the latest catalog memory entry, if any.
- Catalog Relations: observe signals for `peer_conflict` or `fragmented_boundary`.
- Skills: each skill's `created_at_epoch`, full SKILL.md, compact `signal_summary`, and
current `merit_items`.

`signal_summary.by_epoch` and relation `rolled_out_at_epoch` may point to earlier
rollout epochs. Earlier-epoch signals may reflect older skill text; verify against the
current SKILL.md before acting.

## What to Decide

Consolidate is for catalog structure, not single-skill repair.

Use it for:

- `peer_conflict`: two or more skills compete for the same routing scenario.
- `fragmented_boundary`: one reusable capability is split across skills.
- over-cap cleanup when a low-value or overlapping skill can be safely retired.
- persistent low visible contribution: low/zero calls across memory, no merit, and no
clear independent niche.

Do not use it for:

- local `routing_error.overselected`, `routing_error.underselected`, or `content_error`;
modify_del handles those.
- missing capability creation; create handles `catalog_gap`.
- internal body quality of one skill unless it is part of a cross-skill boundary fix.

## Actions

- `modify`: rewrite one SKILL.md to narrow a boundary, reconcile a conflict, or absorb
another skill in a merge.
- `delete`: retire a skill, either as the absorbed half of a merge or as standalone
low-value cleanup.

Set `catalog_op`:

- `merge`: keeper modify plus absorbed delete. Both actions must carry the
```
```

```
  same non-empty `merge_id`; each `merge_id` has exactly those two actions.
- `update`: boundary/content update without changing skill count.
- `delete`: standalone cleanup.

At most {consolidate_max_actions} actions per call.

If no safe structural action is justified, return `actions: []`. Use the top-level
`rationale` to explain the group-level no-op. If `n_skills > target_cap`, the rationale
must name the strongest consolidation candidate considered and explain why changing it
is unsafe; add a `watch` item when future evidence could make the decision actionable.

## Editing Rules

- Any `modify` reason must name at least one other skill in the group. Standalone
low-value `delete` may instead cite the visible low-value signal and lack of independent
niche.
- Do not act on a skill recently modified or merged in the same direction in Catalog
Memory Tail; let it settle.
- Do not merge, delete, or absorb a skill based mainly on Catalog Relations from before
that skill's `created_at_epoch`. Treat those relations as weak background evidence
unless the current SKILL.md files themselves show a clear structural conflict.
- A skill created in the current or immediately preceding decision epoch may appear in
this group before it has direct observe evidence. Default to leaving it unchanged;
modify or delete it only for a concrete safety or structural defect visible in its
current SKILL.md.
- Do not delete or absorb a skill with non-empty `merit_items` unless your reason
explains how those mechanisms survive or why they are superseded.
- Edits can only rewrite `SKILL.md`, and `new_content` must be the full file with YAML
frontmatter.
- Always quote frontmatter `description`.
- Inside SKILL.md, do not mention main agent, worker, dispatch, catalog, prompt, trace,
other skill names, or revision history.
- Do not hardcode URLs, dates, prices, thresholds, named candidates, or baked-in answers.
- Descriptions should state positive use cues; do not write "not for X" blacklists.

## Output

Return strict JSON only:

```json
{
  "rationale": "<group-level reason for acting or returning actions: []>",
  "actions": [
    {
      "op": "modify",
      "catalog_op": "merge | update",
      "merge_id": "<required when catalog_op is merge; otherwise omit>",
      "skill_name": "<name>",
      "reason": "<peer-aware reason for the edit>",
      "edits": [
        {"path": "SKILL.md", "new_content": "<full SKILL.md text>"}
      ]
    },
    {
      "op": "delete",
      "catalog_op": "merge | delete",
      "merge_id": "<same merge_id when catalog_op is merge; otherwise omit>",
      "skill_name": "<name>",
      "reason": "<why this skill should be retired>"
    }
  ],
  "watch": [
    {
      "summary": "<optional unresolved catalog suspicion>",
      "advice": "<optional next-round advice>"
    }
  ]
}
```

When the group is healthy, return `{"rationale": "<why the group needs no structural
change>", "actions": [], "watch": []}`.
```

User template.

```
# Group
domain: {domain}
n_skills: {n_skills}

# Cap
```

```
target_cap: {target_cap}

# Epoch
epoch: {current_epoch}

# Catalog Memory Tail

{catalog_memory_tail}

# Recent Watch

{watch_history_section}

# Catalog Relations

```json
{catalog_relations}
```

Skills

{skills_block}

Return JSON only, following the output contract in the system prompt.
```