FinSkillBench: Evaluating AI Agents and Domain Skills for Investment Management

Jermyn Zhen Yong Bek*
jermynbek@gmail.com
Independent Researcher
Singapore

Zhuang Qiang Bok*
zq.bok@deepinsightlabs.ai
Deep Insight Labs
Singapore

Zhongtian Sun*
Z.Sun-256@kent.ac.uk
University of Kent
Canterbury, United Kingdom
University of Cambridge
Cambridge, United Kingdom

Abstract

Investment management is a high-stakes domain in which agentic AI systems must do more than generate plausible text. They must retrieve point-in-time data, assemble correct computational inputs, invoke specialized methods, and produce auditable structured outputs. We introduce **FinSkillBench**, an evaluation suite designed to measure whether language model agents can effectively use financial domain skills to solve investment management tasks. The benchmark spans three domains, portfolio construction, risk management, and fundamental analysis, and includes 12 subtasks with 2,603 task episodes. Each episode provides point-in-time inputs, hidden ground truth, and a task-specific verifier. We compare three conditions: *no skill*, *curated skill packages* consisting of procedural documents and executable components, and *self-generated skills* in which the agent writes and reuses its own procedures within an episode. Across 9 models and a large-scale evaluation, curated skills consistently improve performance, raising mean scores from 0.366 to 0.528, with the largest gains in portfolio construction and risk management. In contrast, self-generated skills provide little benefit despite higher computational cost. An independent evaluation using a separate agent framework (Hermes Agent, 8 models, 5,280 episodes total) reproduces the directional pattern across all three domains, with the magnitude of skill effects varying by subtask and harness.

These results show that in investment management agents, access to reliable procedural skills can be as important as model choice, while naive self-generation of skills is often ineffective. We release the benchmark, evaluation tools, curated skill packages, and full trajectories to support further research.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → *Information retrieval*; *Information extraction.*

Keywords: agent skills, LLM agents, benchmarking, evaluation, investment management, procedural knowledge

1 Introduction

Large language models are increasingly used as planning and analysis agents. In investment management, such agents are asked to solve tasks with quantitative structure: construct an optimal portfolio, monitor mandate violations, estimate stress losses, compute earnings-quality metrics, and normalize XBRL filings. These tasks differ from open-ended financial question answering. They require point-in-time discipline, structured outputs, reliable tool use, and verifiable numerical or symbolic correctness.

Existing evaluations often ask whether a model knows financial concepts or can answer finance questions. That is necessary but insufficient for evaluating agentic investment workflows. A useful investment agent must know when to load domain procedures, how to pass large numeric inputs without corrupting them, which formula applies to a filing metric, and how to return a result that can be audited by downstream software. The evaluation object is therefore not only the model, but the full process by which a model uses tools, documentation, and reusable skills.

We study this process through **FinSkillBench**, a benchmark and evaluation protocol for AI agents in investment management. The benchmark is organized around three domains: portfolio construction (mean-variance optimization, constrained optimization, tool-call parameterization, rebalancing, and Black–Litterman allocation), risk management (mandate monitoring, risk identification, stress testing, and remediation trades), and fundamental analysis (XBRL metric normalization, earnings quality, and driver decomposition).

The key intervention is *skill availability*. In the **no-skill** condition, the agent receives the task without any tools and reusable domain documentation. In the **curated** condition, it can discover and load human-authored skill documents and tool scripts. In the **self-generated** condition, it must first write skill documents for itself and then use them. This design evaluates not just "Can the model solve the task?" but "Does a reusable skill layer improve agentic financial analysis, and under what assumptions?"

Contributions.

*Equal contribution.

1. A point-in-time investment-management benchmark with 2,603 task episodes across twelve subtasks and three domains.
2. A factorial agentic protocol that isolates the effect of reusable skill access (no-skill, curated, self-generated) while holding tasks, tools, turn budget, and scoring fixed.
3. Task-specific automated verifiers for numeric accuracy, constraint satisfaction, ranked risk recovery, attribution accuracy, trade matching, and financial-statement metric extraction.
4. A full empirical run of **17,820 episodes** over 9 models showing that curated skills materially improve performance (+16.2 pp aggregate) while self-generated skills do not reliably do so in a one-episode setting (+0.5 pp).
5. An independent **cross-harness validation** on a Hermes Agent with 8 models and 5,280 task episodes, which reproduces the pattern at the domain level and shows that the gains concentrate in specific subtasks.
6. An analysis of failure modes and evaluation limitations for skill-based agent benchmarks, including scorer assumptions, benchmark heterogeneity, task-resource coupling, and the interpretation of skill gains.

2 Related Work

Agent skills. Anthropic introduced the Agent Skills specification as an open standard for equipping LLM agents with domain-specific procedural knowledge [1]. SkillsBench [8] provided a systematic evaluation across 86 tasks and 11 general domains, finding curated skills boost pass rates by approximately +16 pp on average while self-generated skills produce no average benefit. Our work extends this analysis to investment management.

Financial NLP benchmarks. FinQA, ConvFinQA, and TAT-QA [4, 5, 14] evaluate numerical reasoning over financial reports and tables. Those datasets are valuable, but they primarily assess question answering over given evidence. FinSkillBench instead evaluates agentic workflows that combine point-in-time data retrieval, domain procedures, executable tools, and structured submission.

Tool-use and agent benchmarks. AgentBench [9] and ToolBench [13] study multi-step tool interaction across broad domains; our benchmark narrows the domain and deepens the task contracts around investment-management methods. Classical financial methods such as mean-variance optimization [10], Black–Litterman allocation [3], factor exposure analysis [6], Piotroski F-Score [12], and Beneish M-Score [2] define many of the ground-truth computations.

3 Benchmark Design

3.1 Domains and Subtasks

FinSkillBench contains 12 subtasks (Appendix C). Each task episode has a unique `task_id`, a domain skill label, a subtask, an `as_of_date`, an input object shown to the agent, a hidden `expected_output`, and a `verification` object specifying the scorer. Portfolio-construction and risk-management subtasks each expose 40 tasks per subtask in the full run; fundamental-analysis subtasks have larger filtered manifests (243 / 179 / 148 for normalization / earnings quality / driver decomposition) of which the runner uses the first 100 each.

3.2 Data Stack and Point-in-Time Discipline

Tasks are anchored in three data substrates. Portfolio construction and risk management use a *market stack* (daily prices, Fama–French factors [6], macroeconomic series) from which we derive a Ledoit–Wolf shrinkage covariance [7], rolling factor exposures, a composite fundamentals-plus-momentum expected-return view, and a market-cap benchmark proxy. Fundamental analysis uses a *reporting stack* built from SEC EDGAR submissions and XBRL CompanyFacts. Risk management additionally relies on *constructed objects*—synthetic portfolios with planted risk profiles, mandate specifications, and stress-scenario definitions—which capture institutional middle-office inputs without conflating the task with filing parsing.

Three design choices govern the data flow. First, all episodes carry an explicit `as_of_date`; system prompts forbid reliance on post-date information and tooling returns only point-in-time slices. Second, task prompts apply *lossy compaction* to large numeric structures (a covariance matrix is summarized in the visible prompt, while available at full precision through tool calls). This separation measures whether agents retrieve and pipe accurate numeric input rather than reasoning from rounded text. Third, the universe is a fixed 30-name sector-stratified sample S&P 500, sacrificing breadth for controlled and reproducible evaluation.

3.3 Ground Truth and Scorers

The ground truth was generated offline, stored in each episode's `expected_output`, and never shown to the agent. Labels fall into three regimes: *exact numerical* (convex-solver outputs, Black–Litterman posteriors, stress P&L); *semi-structured ranked* (risk identification with partial credit for defensible alternative wordings); and *accounting-derived* (canonical line items, screen components, segment-level bridges with tolerances). Each subtask is paired with a verifier whose structure matches the finance-meaningful object the task produces (Table 1). Full derivation pipelines and validation checks are provided in Appendix A, abbreviated input/output examples in Appendix A.5, and complete scorer formulas in Table 10.

Table 1. Scorer-to-subtask mapping. Soft-vs-hard grading is calibrated to operational severity.

Subtask	Scorer	Object rewarded
Unconstrained optim.	`l2_distance_and_objective`	Proximity to MV optimum
Constrained optim.	`constraint_satisfaction_and_objective`	Constraint flags gate weight L2
Tool-use param.	`parameter_match`	Optimizer contract: nested constraint params
Rebalancing	`turnover_compliance_and_objective`	Post-trade weights, turnover, trade-list consistency
Black–Litterman	`view_specification_and_weights`	Posterior expected returns + posterior-optimal weights
Constraint monitoring	`exact_match`	Mandate status + per-constraint accuracy
Risk identification	`ranked_list_recall`	Recovery + ordering of exposure categories
Stress testing	`absolute_error`	Portfolio impact + sector/factor attribution
Risk remediation	`constraint_satisfaction_plus_cost`	Feasible trades restoring compliance
Normalization	`metric_absolute_error`	Canonical line items within tolerance
Earnings quality	`earnings_quality_composite`	Piotroski components + Beneish flag + ratio terms
Driver decomposition	`driver_f1_and_direction`	Named drivers, direction, magnitude

4 Skills as a Scientific Manipulation

4.1 Factorial Design Over Procedural Resources

The central manipulation is *which procedural resources an agent can access* while holding tasks, tools, turn budget, temperature, and scoring fixed. Three conditions define the factor:

- **No-skill.** The skill-access interface is present but the skill corpus is empty: discovery calls resolve to nothing. Finance domain tools (e.g., `compute_risk_report`, `optimize_portfolio`) were not available as well; the SKILL.md and scripts package is withheld. This is the reference baseline used for the main results in Section 6. The cross-harness validation in Section 7 uses the same no-skill variant where both the SKILL.md package *and* the domain tools are withheld; this design choice gave a fairer comparison, as discussed further in Section 7.1.
- **Curated.** Human-authored, domain specific skill and tool script packages were mounted. The agent can discover and read skills, and execute tools within the same episode.
- **Self-generated.** Curated skill material is withheld; the agent receives write access to a per-episode scratch space and an explicit instruction to author one to three modular skill documents *before* answering, then reload them.

The design is factorial over the *resource*, not the *interface*: tool signatures are invariant across conditions, isolating skill-content effects.

4.2 What a Curated Skill Contains

Each domain has exactly one curated package combining **declarative** content (Markdown procedures, formulas, common failure modes, API usage patterns) and **imperative** content (small validated Python entry points invoked through a constrained script-execution tool with JSON I/O). The split addresses a failure mode observed in piloting: language models reliably invent the shape of a computation but fail to pass large numeric structures through tool calls at full precision. A 900-float covariance matrix emitted by a model and reparsed into an optimizer differs from the source at the last few digits, and those differences change the optimum. The curated design therefore keeps numeric-heavy routines inside validated scripts and provides a server-side mechanism to inject full-precision task fields into those scripts without round-tripping through model-generated JSON.

4.3 Agent Loop

The agent is a function-calling, ReAct-style loop with fixed system prompt, task prompt, and tool schema. Tools cover skill discovery and loading, skill authoring (self-generated only), retrieval of full task fields, script execution scoped to the active skill directory, domain data queries, and final JSON submission. Turn budget is capped at 12 with up to 3 retries for transient provider failures. Episodes that fail to emit valid JSON through the submission tool are scored `invalid_submission` or `incomplete_submission` rather than silently dropped. Verbatim prompts and tool availability by condition are listed in Appendix B (Table 8).

5 Experimental Setup

Models. We evaluated 9 models spanning frontier and open-weight families: `gpt-5.4`, `claude-sonnet-4.6`, `gpt-4.1`, `gemini-2.5-pro`, `grok-4`, `DeepSeek-V3.2`, `Phi-4`, `glm-5.1`, `gemini-3.1-flash-lite`.

Run. Full coverage: 9 models × 3 conditions × 12 subtasks. Tasks per cell: 40 for PC and RM subtasks, 100 for FA subtasks. One sample per cell. Total: **17,820 episodes**. Execution: 12 concurrent OS processes (one per subtask runner), 16 `ThreadPoolExecutor` workers each. Temperature 0; max turns 12; eval retries 3.

Validity. 17,176 / 17,820 evaluations (96.4%) returned valid JSON. 644 (3.61%) were `invalid_submission`; 3,736 (21.0%) hit the turn budget (`max_turns_exhausted`); 75 (0.4%) were `incomplete_submission`. `Phi-4` contributed 1,980 rows with mean score 0.000 (turn-budget exhaustion dominated). Most condition analyses exclude Phi-4 to avoid having a single non-functional agent dominate aggregates; the overall row reports both.

6 Results

6.1 Aggregate Skill Effect

Pooled across all three conditions, the mean score over 17,820 evaluations is 0.375, or 0.422 once we drop Phi-4 (mean 0.000 across all 1,980 episodes). The per-condition means in Table 2 show that curated skills give a large, stable improvement over no-skill, while self-generated skills barely move the needle.

Table 2. Aggregate condition means, excluding Phi-4 (5,280 evaluations per condition). CIs are bootstrap 95% intervals (10,000 resamples) on paired deltas.

Condition	Mean	Δ vs. No-Skill	95% CI
No-Skill	0.366	—	—
Curated	**0.528**	**+0.162**	[+0.152, +0.171]
Self-Generated	0.371	+0.005	[−0.002, +0.011]

6.2 Domain-Level Effects

Curated-skill gains are largest where the skill provides executable computational primitives (Table 3).

6.3 Subtask-Level Effects

The picture sharpens at subtask resolution (Table 4). The largest improvements occur on subtasks where the no-skill baseline is near zero or modest; subtasks with already-high baselines show small or negligible gains.

The largest improvements occur in optimization and risk tasks where a correct answer depends on multi-step numerical computation. Tool-use parameterization has high no-skill performance because the expected output is a structured parameter translation rather than a full computation. Normalization shows no curated benefit because the base task is largely direct XBRL extraction plus simple arithmetic, already accessible via the global XBRL query tool.

6.4 Model-Level Effects

Skill effects are heterogeneous across base models (Table 5).

Curated skills do not simply help stronger models more. The largest gain accrues to `gpt-4.1` (+0.405), while `gpt-5.4` sees a smaller +0.091 gain and `claude-sonnet-4.6` starts from the strongest no-skill baseline (0.534) and gains +0.125. This suggests skill use is an interaction between model capability, tool-calling behavior, and skill-following reliability, not a monotone function of base model strength. Two failure modes prevent skill uptake entirely: `Phi-4` exhausts the turn budget on essentially every episode, and `grok-4` produces malformed final outputs at high rate.

6.5 Cost and Interaction Overhead

Self-generated skills are expensive in context and turns without delivering accuracy gains (Table 6).

Self-generated skills load in 97.7% of non-Phi-4 episodes, but the generated skills rarely improve accuracy. "Writing down a procedure" inside the episode is not equivalent to having a validated domain skill with tested scripts and precise data-access guidance.

7 Cross-Harness Validation

To test whether the no-skill vs. curated effect is robust to agent-implementation choices, we replicate the comparison on an independent harness: the Nous Research **Hermes Agent** [11] with the same fin domain skills, tools, prompt template, submission protocol, and turn budget (max_turns = 12). Only the Hermes `skills_list` and `skill_view` tools were activated, with all other Hermes built-in skills, toolsets, memory and personality turned off. Eight models—gpt-5.4, gpt-4.1, claude-sonnet-4.6, gemini-2.5-pro, glm-5.1, grok-4.20, DeepSeek-V3.2, gemma-4-31b-it—were each evaluated on 240 episodes (60 FA, 100 PC, 80 RM) across all the 12 subtasks , totaling **1,920** evaluations. Both runs were complete (zero runner errors) and used the same task data and scorers as the main study.

7.1 Aggregate and Domain Effects

Table 7 summarizes the Hermes replication at the aggregate, domain, and model levels.

The Hermes agent harness reproduces the directional pattern across all groups - at the overall level the no-skill baseline scored closely with the original harness (0.354 vs 0.366), and the Hermes harness had a higher uplift with curated

Table 3. Domain-level results (excl. Phi-4). PC and RM benefit most because curated skills contain validated solvers and precise input-passing instructions. CI shown for the curated − no-skill paired delta.

Domain	No-Skill	Curated	Self-Gen	Curated Δ	Self-Gen Δ
Portfolio construction	0.307	**0.585**	0.301	**+0.278** [+0.258, +0.298]	−0.006 [−0.015, +0.004]
Risk management	0.269	**0.486**	0.303	**+0.218** [+0.195, +0.239]	+0.034 [+0.018, +0.049]
Fundamental analysis	0.458	**0.512**	0.454	**+0.054** [+0.045, +0.064]	−0.004 [−0.014, +0.006]

Table 4. Subtask-level results (excl. Phi-4). Sorted by Curated − No-Skill Δ.

Subtask	Domain	No-Skill	Curated	Self-Gen	Δ
Unconstrained optimization	PC	0.247	0.719	0.236	**+0.472**
Constrained optimization	PC	0.148	0.505	0.127	**+0.358**
Risk identification	RM	0.023	0.332	0.055	**+0.309**
Rebalancing	PC	0.183	0.451	0.164	+0.268
Stress testing	RM	0.017	0.281	0.027	+0.265
Black–Litterman	PC	0.153	0.375	0.147	+0.222
Constraint monitoring	RM	0.597	0.775	0.651	+0.178
Risk remediation	RM	0.438	0.558	0.478	+0.120
Earnings quality	FA	0.436	0.535	0.474	+0.099
Tool-use parameterization	PC	0.803	0.872	0.831	+0.069
Driver decomposition	FA	0.256	0.322	0.249	+0.066
Normalization	FA	0.681	0.678	0.638	−0.003

Table 5. Per-model overall and condition means. "Overall" is the unweighted mean across all 12 subtasks averaged over the three conditions.

Model	Overall	No-Skill	Curated	Self-Gen	Curated Δ
gpt-4.1	0.488	0.330	**0.735**	0.398	**+0.405**
gemini-2.5-pro	0.490	0.380	**0.678**	0.412	**+0.298**
DeepSeek-V3.2	0.521	0.458	**0.658**	0.447	+0.200
gemini-3.1-flash-lite	0.487	0.419	**0.609**	0.434	+0.191
claude-sonnet-4.6	0.560	0.534	**0.659**	0.488	+0.125
gpt-5.4	0.452	0.430	**0.520**	0.405	+0.091
glm-5.1	0.268	0.274	0.285	0.245	+0.011
grok-4	0.107	0.104	0.077	**0.138**	−0.027
Phi-4	0.000	0.000	0.000	0.000	0.000

Table 6. Per-episode resource use, excluding Phi-4.

Condition	Median Tokens	Mean Tokens	Median Latency (s)	Mean Turns
no-skill	19,952	38,783	59.75	7.46
curated	22,546	35,378	49.79	7.30
self-generated	40,034	63,606	76.76	9.29

skills (+0.325 vs +0.162). At the domain level: portfolio construction and risk management showed a substantial, statistically significant gain (+0.4 non-overlapping CIs), while fundamental analysis showed a smaller but significant change.

Again, using the Hermes Agent provided a bigger uplift than the FinSkillBench harness. One factor could have drove this difference: the Hermes Agent harness load tools as Python

Table 7. Hermes harness: aggregate, by-domain, and by-model results. "†" indicates the curated and no-skill 95% CIs do not overlap. Numbers are unweighted means over 1,920 (overall, no-skill) and 5,280 (overall, curated skills) overall episodes. No-skill was a sample run of the first 20 episodes per each 12 subtask (240 total per model).

Group	*N*	No-Skill (95% CI)	*N*	Curated (95% CI)	Δ	
	Overall					
All	1920	0.354 [0.338, 0.370]	5280	0.679 [0.669, 0.688]	+0.325	†
	By domain					
Fundamental analysis	480	0.439 [0.408, 0.470]	2400	0.567 [0.553, 0.582]	+0.129	†
Portfolio construction	800	0.325 [0.301, 0.350]	1600	0.760 [0.744, 0.776]	+0.435	†
Risk management	640	0.325 [0.297, 0.354]	1280	0.785 [0.767, 0.802]	+0.459	†
	By model					
claude-sonnet-4.6	240	0.304 [0.257, 0.353]	660	0.795 [0.775, 0.815]	+0.490	†
DeepSeek-V3.2	240	0.371 [0.325, 0.419]	660	0.686 [0.659, 0.712]	+0.315	†
gemini-2.5-pro	240	0.333 [0.291, 0.375]	660	0.477 [0.441, 0.512]	+0.144	†
gemma-4-31b-it	240	0.375 [0.332, 0.418]	660	0.642 [0.614, 0.670]	+0.267	†
glm-5.1	240	0.397 [0.350, 0.444]	660	0.779 [0.758, 0.801]	+0.383	†
gpt-4.1	240	0.285 [0.244, 0.327]	660	0.699 [0.672, 0.726]	+0.414	†
gpt-5.4	240	0.408 [0.365, 0.451]	660	0.660 [0.631, 0.687]	+0.252	†
grok-4.20	240	0.357 [0.314, 0.401]	660	0.691 [0.664, 0.719]	+0.334	†

functions, instead of reading scripts in the skills folder. This made tool calling and usage more efficient.

Note on no-skill design across runs. Both the Section 6 and Section 7 no-skill condition withholds domain tools, so the +32.5 pp aggregate Hermes uplift measures the same combined effect of skill documents and executable tool access. This is almost two times higher than the +16.2 pp curated skills uplift from the FinSkillBench agent harness. A clean decomposition of documents, tools, and their interaction would require a four-arm design: no-skill plus no-tools, no-skill plus tools, skill plus no-tools, and skill plus tools. We leave this to future work.

7.2 Concentration of the Effect

At the model level with the Hermes Agent harness, claude-sonnet-4.6 showed the biggest improvements wih skills (+0.49). Open weight models like GLM-5.1, DeepSeek-V3.2 and gemma-4-31b-it also showed good improvements of 38.3 pp, 31.5 pp and 26.7 pp respectively, with GLM-5.1 scoring the second highest (0.779) with curated skills. This proves that with appropriate skills and tools, local deployment of LLMs and financial agents yield good results. This is crucial in some financial settings for data privacy and security.

7.3 Cross-Harness Comparison: Same Pattern, Different Magnitude

The Risk identification (domain: RM) subtask, one of the more difficult tasks, appears in both harnesses with similar baselines but larger skill gains in Hermes Agent: FinSkillBench harness 0.023 → 0.332 (+0.309); Hermes harness 0.043 → 0.457 (+0.414). Constrained optimization (domain: PC) in contrast, had similar baselines on the two harnesses (FinSkillBench: 0.148; Hermes: 0.177) and correspondingly different deltas (+0.358 vs. +0.087). The cross-harness data was consistent with one explanation that we leave to future work, larger studies to test rigorously: *when the no-skill baseline is near zero, curated skills can produce large gains; when the no-skill baseline is already near the ceiling, skills cannot help further.* We report this as a hypothesis suggested by the data, not as a tested claim.

7.4 Implications

The cross-harness results have two practical implications. First, claims of "*X* pp skill uplift" from any single benchmark are conditional on the no-skill baseline distribution implied by that harness's tools, prompts, and turn budget. Second, the signs of the cross-harness deltas are consistent (positive across all three domain; FA with the lowest improvement, PC and RM with higher improvements). A single benchmark and a single harness can therefore underestimate or overestimate skill effects depending on where its baselines sit. We recommend that future benchmarks report skill effects stratified by baseline level, not only as a single aggregate number.

8 Discussion

8.1 What Curated Skills Measure

Curated skill gains should not be interpreted as pure knowledge gains. In this benchmark, skills provide a combination of: (i) procedural reminders (which formula or workflow applies); (ii) API and tool-use guidance (correct argument names, parameter shapes); (iii) executable scripts implementing financial algorithms; and (iv) warnings about common failure modes (e.g., precision loss when passing covariance matrices through model-generated JSON). This combination reflects how practical agent systems are built. Production agents are rarely deployed as unaided language models; they are embedded in environments with documentation, tools, calculators, and guardrails. The benchmark therefore evaluates whether an agent can exploit such resources correctly.

8.2 Why Self-Generated Skills Underperform

The self-generated condition is intentionally strict. The agent must infer useful reusable knowledge, write it as a skill, reload it, and still solve the original task within 12 turns. This creates overhead and produces unvalidated instructions. The empirical result is that self-generated skills are usually not enough for one-shot improvement. They may still be useful in a multi-episode learning setting where generated skills persist, are tested, and can be corrected over time, but this benchmark does not evaluate that scenario.

8.3 Evaluation as a Scientific Object

The benchmark demonstrates that conclusions about "financial agent capability" depend strongly on evaluation design. For example: if invalid and max-turn submissions are dropped, score estimates are inflated relative to deployment-like reliability; if no-skill agents can call the same deterministic solvers as curated agents, the measured skill effect is confounded; if constrained-optimization is scored only by constraint-status labels, the metric may reward portfolios far from the optimizer solution; if Black–Litterman tasks expose fewer analyst views than the ground truth used, the task is unsolvable. These issues motivated the current scorer and runner design and illustrate why evaluation artifacts need documented assumptions, not just leaderboards.

9 Limitations

Single-sample cells. The full run uses one sample per (model, condition, task) cell. Our estimand is the population-level treatment effect of procedural-resource availability across the task distribution, not the stochastic success rate of any single trajectory. Because all conditions are evaluated on the same episodes with deterministic verifiers, we use paired comparisons, which absorb episode-level difficulty variance, and quantify uncertainty by bootstrap over paired deltas (10,000 resamples). We use temperature 0 to reduce within-cell variance, though this does not make provider-side execution fully deterministic. Repeated runs would tighten within-cell estimates but are not required to identify the population-level effect under our paired design. We acknowledge this leaves agent-loop variance unmeasured, particularly in turn-budget-exhausted episodes (21% of calls), and flag it as a target for future expansion.

Universe. The 30-ticker S&P 500 sample supports controlled evaluation but does not cover all asset classes, international markets, derivatives, private assets, or illiquid securities.

Engine-derived labels. Many ground-truth labels are produced by deterministic engines (convex solvers, Black–Litterman routines). The benchmark therefore evaluates agreement with those engines and their assumptions. Reasonable alternative modeling choices (e.g., shrinkage intensity) would yield slightly different labels.

Driver decomposition ceiling. Some expected driver labels depend on segment-level data not fully present in the processed XBRL panel. This caps attainable performance for agents that avoid unsupported domain guesses.

Skill content includes scripts. Curated skills include executable Python scripts. The evaluation is therefore about agent ability to discover and use domain skills, not about unaided financial reasoning—a feature for studying practical agents, but a caveat when comparing to text-only benchmarks.

Time-dependent providers. Tool-calling reliability, model versions, and API routing change. The artifact logs model identifiers and run configuration, but external reproducibility requires stable access to comparable model endpoints. `Phi-4` and `grok-4` in particular exhibited tool-calling failure modes on this run that may not generalize.

Cross-harness scope. While the Hermes replication had the same experimental scope and smaller sample episode size, it is sufficient to show that the directional skill effect on all domains transfers to a different harness. However, we should not establish equivalence of the two harnesses.

10 Responsible Use and Ethics

This benchmark is for evaluating analytical correctness in controlled investment-management tasks. It is not investment advice and should not be used to justify autonomous trading, suitability determinations, or client-specific recommendations. Financial agents can cause harm through erroneous calculations, hallucinated evidence, stale data, hidden assumptions, or inappropriate optimization objectives. The benchmark's point-in-time discipline, structured outputs, and explicit scoring are intended to make such failures easier to observe, not to eliminate them.

11 Conclusion

FinSkillBench evaluates investment-management agents as systems that combine language models, tools, task data,

and reusable procedural knowledge. Across 17,820 full-run episodes over 9 models, curated skills produce a large and statistically stable improvement over no-skill baselines (+0.162; CI [+0.152, +0.171]), especially in optimization and risk workflows. Self-generated skills, by contrast, increase token and turn overhead without improving aggregate accuracy in this one-episode setting (+0.005; CI [−0.002, +0.011]). An independent cross-harness replication on Hermes Agent, covering 8 models, 1,920 no-skill episodes, and 5,280 curated episodes, reproduces the same domain-level pattern, with non-overlapping 95% confidence intervals in every domain. The domain-level uplifts are +12.9 pp for FA, +43.5 pp for PC, and +45.9 pp for RM. The Hermes uplift is larger in absolute terms than the FinSkillBench-Skills uplift, partly because the no-skill condition in this cross-harness setting withholds domain tools as well as the `SKILL.md` package, as discussed in Section 7.1. These findings argue for evaluating domain agents under explicit resource conditions and for treating skills, tools, and scoring assumptions—and the harness itself—as first-class objects of evaluation.

Acknowledgments

We thank the Agent Skills '26 reviewers for their feedback. We release FinSkillBench task episodes, curated skill packages, evaluation harnesses, and full agent trajectories to support reproducibility.

References

[1] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering Blog, October 2025.

[2] Messod D. Beneish. 1999. The Detection of Earnings Manipulation. *Financial Analysts Journal* 55, 5 (1999), 24–36.

[3] Fischer Black and Robert Litterman. 1992. Global Portfolio Optimization. *Financial Analysts Journal* 48, 5 (1992), 28–43.

[4] Zhiyu Chen et al. 2022. ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[5] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameer Shah, et al. 2021. FinQA: A Dataset of Numerical Reasoning over Financial Data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[6] Eugene F. Fama and Kenneth R. French. 1993. Common Risk Factors in the Returns on Stocks and Bonds. *Journal of Financial Economics* 33, 1 (1993), 3–56.

[7] Olivier Ledoit and Michael Wolf. 2004. A Well-Conditioned Estimator for Large-Dimensional Covariance Matrices. *Journal of Multivariate Analysis* 88, 2 (2004), 365–411.

[8] X. Li et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670.

[9] Xiao Liu et al. 2023. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688* (2023).

[10] Harry Markowitz. 1952. Portfolio Selection. *The Journal of Finance* 7, 1 (1952), 77–91.

[11] Nous Research. 2026. Hermes Agent. https://github.com/nousresearch/hermes-agent. GitHub repository, MIT license. Accessed: 2026-05-03.

[12] Joseph D. Piotroski. 2000. Value Investing: The Use of Historical Financial Statement Information to Separate Winners from Losers. *Journal of Accounting Research* 38 (2000), 1–41.

[13] Yujia Qin et al. 2023. ToolBench: Towards Evaluating Large Language Models for Tool Learning. *arXiv preprint arXiv:2307.16789* (2023).

[14] Fengbin Zhu, Wenhu Lei, Yujia Huang, et al. 2021. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

A Data Processing and Ground Truth Derivation

A.1 Universe and Shared Infrastructure

All three domains share a common 30-ticker universe drawn from the S&P 500, stratified across all 11 GICS sectors. The universe is frozen at a single snapshot date to avoid survivorship bias. Sector distribution: Technology (5), Financial Services (4), Industrials (4), Healthcare (3), Consumer Cyclical (3), Consumer Defensive (2), Energy (2), Real Estate (2), Basic Materials (2), Utilities (2), Communication Services (1). Each record carries symbol, CIK, ISIN, CUSIP, sector, subsector, exchange, country, and S&P 500 addition date.

All three data pipelines follow the same architecture:

`Download` → `Process` → `Construct` → `Ground Truth` → `Tasks` → `Validate`

Each pipeline is orchestrated by a single `run_pipeline.py` that loads the security master, creates a shared context, resolves which phases to run, and executes them in dependency order.

A.2 Portfolio Construction Data

Raw data sources.

- **Daily Prices**: Daily OHLCV records per ticker. Each record contains `date`, `open`, `high`, `low`, `close` (split-adjusted), `adjClose`, and `volume`. 30 files, covering 2024-01 through 2025-12.
- **Dividend-Adjusted Prices**: Separate adjusted-close series accounting for dividends and splits.
- **Historical Market Capitalization**: Daily market cap per ticker, used to construct benchmark weights.
- **Fama–French Factors** [6]: Daily returns for the 5-factor model (Mkt-RF, SMB, HML, RMW, CMA) plus momentum.
- **Macroeconomic Series**: 8 time series—10-Year Treasury, 2-Year Treasury, 3-Month Treasury, yield curve spread, CPI, unemployment rate, Fed Funds rate, and VIX.

Processing logic.

1. **Price panel**: `processed/prices.parquet`. Per-ticker JSON files are merged into a wide-format panel (15,060 rows × 9 columns, 24 months, 30 symbols). Missing dates are forward-filled; tickers with fewer than 120 observations are flagged.

2. **Returns**: `processed/returns.parquet`. Daily simple returns computed as $(P_t - P_{t-1})/P_{t-1}$ from adjusted close prices.
3. **Factor exposures**: `processed/factor_exposures.parquet`. For each ticker and each month-end date, a rolling 252-day OLS regression of excess returns against the 6 factors (FF5 + momentum) produces per-ticker factor betas. The regression requires a minimum of 120 observations.
4. **Benchmark weights**: `processed/benchmark_weights.parquet` (240 rows). Monthly market-cap-weighted benchmark across the 30 symbols. For each month-end, weights are proportional to market capitalization: $w_i = \text{mcap}_i / \sum_j \text{mcap}_j$.
5. **Covariance matrices**: `processed/covariance/{date}.npz` (12 files). Ledoit–Wolf shrinkage covariance [7] estimated from a 252-day trailing window of daily returns, annualized by multiplying by 252. Stored as compressed NumPy archives with `cov` (30×30 array) and `symbols` (ticker ordering). Positive semi-definiteness is verified; if the minimum eigenvalue is negative, a ridge correction $\Sigma \leftarrow \Sigma + (|\lambda_{\min}| + 10^{-8})I$ is applied.
6. **Expected-return views**: `processed/views/{date}.json` (22 files). A composite signal from 5 alpha factors, each cross-sectionally z-scored and combined with fixed weights: earnings yield (25%), ROE (20%), 12-1 momentum (25%), EPS revision (15%), low-volatility (15%). The composite z-score is scaled to daily expected returns via Grinold–Kahn alpha methodology. Each file contains per-ticker signal values, composite z-scores, and the resulting expected returns.
7. **Constraint sets**: `processed/constraints/{easy,medium,hard}.json`. Three tiers of increasing restrictiveness:
   - `easy`: long-only, max 6%/name, max 25%/sector, 20–30 names.
   - `medium`: long-only, max 6.6%/name, max 20%/sector, 18–30 names, turnover ≤ 20%, tracking error ≤ 3%.
   - `hard`: long-only, max 7.7%/name, max 15%/sector, 15–30 names, turnover ≤ 15%, tracking error ≤ 2%, factor-neutral on Mkt-RF, SMB, HML.

Ground truth derivation. For each (date, difficulty, objective) triple where both covariance and views exist, we run `optimize_portfolio()` via `cvxpy`—a convex quadratic program with the specified objective and constraints. The solver tries SCS first (handles SOCP/conic programs), falling back to OSQP (QP only). The four objectives are formulated as:

- `max_sharpe`: maximize $\mu^\top w - r_f - \lambda \cdot w^\top \Sigma w$ (scalarized utility).
- `min_variance`: minimize $w^\top \Sigma w$.
- `max_return`: maximize $\mu^\top w$.
- `risk_parity`: minimize $\|w - w_{\text{inv-var}}\|_2^2 + 0.5 \cdot w^\top \Sigma w$, where $w_{\text{inv-var}}$ are inverse-variance weights.

This produces 144 main ground truth files (12 dates × 3 difficulties × 4 objectives). Of these, 96 are optimal and 48 are infeasible (all hard-difficulty, where factor neutrality plus tight constraints are infeasible with 30 symbols).

For rebalancing ground truth, a two-stage procedure is used: Stage 1 solves the standard optimization; Stage 2 minimizes one-way turnover $\|w - w_0\|_1/2$ subject to staying within a tight tolerance of the Stage 1 objective value.

For Black–Litterman, for each of 12 dates × 4 view variants (different random seeds controlling number of views, symbols, return magnitudes, confidence levels), we run the BL model [3]. The model computes equilibrium returns $\pi = \delta \Sigma w_{\text{mkt}}$ from market-cap weights, constructs the pick matrix P and view vector Q from analyst views, and computes the posterior via $\mu_{\text{BL}} = [(\tau\Sigma)^{-1} + P^\top \Omega^{-1} P]^{-1}[(\tau\Sigma)^{-1}\pi + P^\top \Omega^{-1} Q]$ where $\Omega_{kk} = (1 - c_k) \cdot P_k \Sigma P_k^\top \cdot \tau$ and c_k is the confidence of view k. This produces 48 BL ground truth files.

Example: ground truth for `2025-01-31_easy_max_sharpe`. The optimizer produces weights across 30 tickers: AIG 6.0%, BF-B 6.0%, BR 6.0%, CMG 6.0%, CTRA 6.0%, DHR 6.0%, KHC 6.0%, L 6.0%, LHX 6.0%, LNT 6.0%, REG 6.0%, TTWO 6.0% (all at the 6% cap), with smaller allocations to BAX 1.3%, CPRT 4.8%, DD 2.5%, HUM 2.2%, LYV 0.6%, SBUX 3.5%, SRE 4.7%, TT 3.5%, ZBRA 4.8%, and zero weight on C, CARR, CPAY, EFX, EQT, ON, USB, VMC. Expected return: 0.077%/day; risk: 10.76%; Sharpe: 0.0071. All constraints satisfied.

Ground truth validation. All 320 PC and RM ground truth entries were independently validated through two layers: (i) *property checks* verifying structural invariants (weight sums, non-negativity, constraint satisfaction, rank ordering, trade arithmetic), and (ii) *re-derivation* re-running the same computation from stored inputs and comparing against stored expected outputs. Re-derivation confirms exact reproducibility for unconstrained optimization, constrained optimization, Black–Litterman, and stress testing (all L2 < 0.001 or absolute error < 0.005). Zero errors, zero warnings across all 320 tasks.

Ground truth patches. Two bugs were discovered and fixed before the evaluation run. (i) The original pipeline generated "unconstrained" GT using the `easy` constraint set; a patch re-ran the optimizer with truly unconstrained settings (`long_only=True` only). (ii) The `black_litterman()` function overwrote `view_adjustments` each loop iteration instead of appending; a patch fixed the bug, reconstructed original views from the same RNG seed, and re-ran the BL model. All original files were backed up.

A.3 Risk Management Data

Raw data sources.

- **Daily Prices**: Shared price data from the PC pipeline (30 tickers, 2017–2025).
- **Fama–French Factors**: Daily factor returns for FF5 + momentum.
- **Macroeconomic Series**: 9 series—Fed Funds Effective Rate, VIX, 2-Year Treasury, 10-Year Treasury, yield curve spread, high-yield credit spread, CPI, unemployment rate, and trade-weighted USD index. Daily series are used directly; monthly series (CPI, unemployment) are forward-filled to daily frequency.

Processing logic.

1. **Returns**: `processed/returns/`. Daily simple returns, log returns, 21-day rolling annualized volatility ($\sigma_i = \text{std}(r_{i,t-20:t}) \times \sqrt{252}$), and running max drawdown from trailing peak.
2. **Factor exposures**: `processed/factor_exposures.parquet`. Rolling 252-day OLS regression of each ticker's excess returns against the 6 factors. Minimum 120 observations required. Produces per-ticker, per-date betas: $\beta_{\text{mkt\_rf}}, \beta_{\text{smb}}, \beta_{\text{hml}}, \beta_{\text{rmw}}, \beta_{\text{cma}}, \beta_{\text{mom}}$. Median absolute beta across all estimates: 0.429; range: $[-2.55, +4.52]$.
3. **Covariance**: `processed/covariance/cov_{date}.npz`. Monthly Ledoit–Wolf shrinkage covariance, same methodology as PC. All 12 matrices verified PSD.
4. **Macro panel**: `processed/macro_panel.parquet`. Merged macro indicators, forward-filled to daily frequency for point-in-time access.

Constructed data. This is the RM-specific step that creates synthetic evaluation objects with deliberately planted risk characteristics.

Portfolios (200 files, 5 types × 40 each).

- *Concentrated*: 4 names at 15% each (60% total in one sector), remaining 15 names at ∼2.7% each. Planted risks: `sector_concentration`, `single_name_risk`, `low_diversification`. Example: AIG, C, L, USB each at 15% (Financial Services sector), plus 15 diversifying names.
- *Factor-tilted*: Overweight high-beta or high-momentum names to create extreme factor exposures ($|\beta| > 0.5$ on at least one factor).
- *Liquidity-stressed*: Overweight small-cap or low-volume names from the universe.
- *Mandate-violating*: Deliberately violates common mandate constraints (position limits, sector limits, beta range).
- *Stress-vulnerable*: High sensitivity to specific stress scenarios (e.g., concentrated in rate-sensitive sectors for interest-rate shocks).

Mandates (12 files across 3 difficulty tiers).

- *Easy* (2 mandates): Simple Long-Only (max 8%/name, max 30%/sector); Diversified Equity (max 5%/name, min 50bp, max 25%/sector).
- *Medium* (3 mandates): Balanced with tracking error < 3%, beta 0.8–1.2, turnover < 15%; Conservative with max 4%/name, max 20%/sector, TE < 2%, beta 0.7–1.0, HHI < 0.08.
- *Hard* (3 mandates): Institutional Core (max 4%/name, TE < 2.5%, beta 0.9–1.1, turnover < 10%, market beta < 1.1); Enhanced Index (max 3.5%/name, TE < 1.5%, beta 0.95–1.05, turnover < 8%, HHI < 0.05).

Stress scenarios (12 files: 4 historical replays + 8 hypothetical).

- *Historical replays*: COVID shock (Feb–Apr 2020), 2018 selloff (Oct–Dec 2018), 2022 tightening (Jan–Oct 2022), 2023–24 rebound.
- *Hypothetical (rates/credit/equity)*: Rates +200bp (equity −5%, credit +50bp); Rates −100bp (equity +2%, credit −20bp); Equity crash −20% (credit +200bp); Credit crisis (equity −10%, credit +300bp).
- *Hypothetical (macro/rotation)*: Correlation spike (target $\rho = 0.8$); Sector rotation (tech −15%, value +5%); Strong dollar (USD +10%, EM equity −8%); Stagflation (CPI +3%, GDP −2%, rates +100bp).

Each scenario specifies factor-level shocks and per-sector sensitivities.

Ground truth derivation. All RM ground truth is deterministic—computed from the constructed portfolios, mandates, scenarios, and processed market data using a reference risk engine (`_risk_engine.py`).

- **Risk identification** (200 GT): For each portfolio, the engine computes concentration metrics (HHI = $\sum w_i^2$, top-5 weight share), sector exposures, portfolio-level factor betas ($\beta_p = \sum w_i \beta_i$), and VaR/CVaR at 95% confidence from the trailing 252-day return distribution. Risks are ranked by magnitude into types: `factor_tilt`, `sector_concentration`, `single_name`, `concentration`, and `liquidity_risk`.
- **Constraint monitoring** (600 GT): For each (portfolio, mandate) pair, each constraint is evaluated: position limits, sector limits, beta range, long-only ($w_i \geq 0$), and turnover (if applicable). Each constraint produces a PASS/FAIL status with the worst offender.
- **Stress testing** (2,400 GT): For each (portfolio, scenario) pair, the engine applies factor-level shocks to the portfolio's factor exposures,

$$\text{PnL}_i = \sum_f \beta_{i,f} \cdot \text{shock}_f + \text{sector\_sensitivity}_i, \quad \text{PnL}_p = \sum_i w_i \cdot \text{PnL}_i.$$

Attribution decomposes total P&L into sector and factor contributions. All 40 stress-testing GT entries reproduce to within 0.005 absolute error on re-derivation.

- **Risk remediation** (400 GT): For each non-compliant (portfolio, mandate) pair, a constrained optimizer generates minimum-turnover trades to restore compliance. Trade arithmetic is verified via $w_{\text{current}} + \Delta w = w_{\text{target}}$ for every trade (tolerance 10^{-4}).

Example: constructed portfolio. `portfolio_concentrated_000`: 4 Financial Services names (AIG, C, L, USB) each at 15.0%, plus 15 diversifying names at 2.7% each. Total NAV: $100M. Planted risks: sector concentration (Financial Services at 60%), single-name risk (4 names > 10%), low diversification (HHI ≈ 0.10).

Example: stress scenario. `stress_rates_down_100`: Hypothetical flight-to-safety scenario. Shocks: interest rates −100bp (parallel shift), equity market +2% (correlated move), credit spread −20bp. Sector sensitivities: Real Estate −15%, Utilities −10%, Technology −8%, Healthcare −3%, Financial Services +3%.

Example: mandate. `mandate_enhanced_index` (hard): 8 constraints—position limit max 3.5%, sector limit max 20%, tracking error < 1.5%, beta 0.95–1.05, monthly turnover < 8%, gross exposure ≤ 100%, long-only, HHI < 0.05.

Task episodes are stratified samples of 40 tasks per subtask from the GT pool, balanced across portfolio types, regimes, and difficulty levels (160 RM tasks total).

A.4 Fundamental Analysis Data

Raw data sources.

- **Regulatory Filing Metadata**: Filing metadata per company (30 files). Contains filing history, company info, and identifier mappings.
- **XBRL CompanyFacts**: All reported XBRL facts for a company across all filings, organized by taxonomy and tag (30 files). This is the primary structured data source for financial metrics.
- **Pre-Parsed Financial Statements**: Quarterly and annual income statements, balance sheets, cash flow statements, key metrics, and financial ratios. Used as cross-reference for ground truth validation.
- **Revenue Segmentation**: Revenue breakdown by product/business segment and by geographic region. Used for driver decomposition ground truth.
- **Parsed Filing Sections**: Extracted sections from 10-K/10-Q filings—MD&A, financials, footnotes, and semantic tree. Coverage: 93% for MD&A, 17% for footnotes.

Processing logic.

1. **Filing timeline**: `processed/filing_timeline.parquet`. Merges regulatory filing metadata with pre-parsed financial statement period data. Maps each (symbol, period_end) to form type (10-K, 10-Q, 20-F, 40-F), filing date, accession number, filer type (domestic/foreign), and XBRL taxonomy (us-gaap/ifrs-full). 287 total filings, 281 cross-matched (98%).
2. **XBRL panel**: `processed/xbrl_panel.parquet` (2,880 rows × 17 columns). Extracts canonical financial metrics from CompanyFacts JSON. XBRL tag names are normalized via an alias mapping to canonical metric names. Example revenue tags include `Revenues`, `RevenueFromContractWithCustomerExcludingAssessedTax`, and `SalesRevenueNet`, all mapped to `revenue`. Each row contains: symbol, period_end, canonical metric name, value, duration class (`instant` for balance-sheet items; `quarterly`, `ytd`, or `annual` for flow items), and filing metadata. Coverage: 30 symbols × 46 unique periods.
3. **YTD-to-quarterly conversion**: For flow metrics (revenue, net income, operating cash flow, etc.) where only year-to-date values are available, quarterly values are derived by differencing consecutive YTD entries: $Q_t = \text{YTD}_t - \text{YTD}_{t-1}$. The pipeline uses actual period lookups from the XBRL panel rather than fixed calendar offsets, handling fiscal-year misalignment correctly.
4. **Computed fallbacks**: When `total_liabilities` is not directly reported in XBRL, it is computed as total_assets − stockholders_equity. Stock metrics (balance sheet) prefer `instant` duration class; flow metrics prefer `quarterly`, then derived from YTD.

Ground truth derivation. All FA ground truth is computed from raw XBRL and pre-parsed financial statement data—no optimizer or model involved. Cross-referencing between the two sources provides validation (75.4% exact agreement rate).

Normalization (244 GT): For each (ticker, period_end), 10 canonical metrics are extracted and cross-referenced:

- Direct from XBRL and pre-parsed statements: revenue, operating_income, net_income, eps_diluted, total_assets, total_liabilities, stockholders_equity, operating_cash_flow.
- Computed: operating_margin = operating_income / revenue; ebitda = operating_income + depreciation_amortization.
- Each metric carries a confidence level: `high` (both structured sources agree within 1%), `medium` (computed/derived), `low` (single-source flow metric).

Earnings quality (1,002 GT): For each (ticker, period_end), the pipeline computes from quarterly financial statements:

- *Piotroski F-Score* [12]: 9 binary components testing profitability (ROA positive, CFO positive, ΔROA positive, accruals negative), leverage (Δleverage negative, Δcurrent ratio positive, no dilution), and efficiency (Δgross margin positive, Δasset turnover positive). Each component requires current and prior-year-same-quarter data.
- *Beneish M-Score* [2]: 8 ratio components (DSRI, GMI, AQI, SGI, DEPI, SGAI, LVGI, TATA) combined into a composite score. $M > -1.78$ flags manipulation risk.
- *Accruals ratio*: (net_income − operating_cash_flow) / total_assets.
- *Income quality ratio*: operating_cash_flow / net_income.
- *Flags*: Triggered by low Piotroski (≤ 3), Beneish manipulation flag, high accruals (> 0.05), or negative income quality.

Driver decomposition (997 GT): For each consecutive quarter pair, the pipeline computes:

- Revenue delta: $\Delta R = R_{\text{current}} - R_{\text{prior}}$.
- Top revenue drivers: Decomposed into product-segment and geographic-segment contributions from revenue segmentation data. Each driver has a name, type (`product_segment` or `geographic_segment`), direction (increase/decrease), magnitude in USD, and contribution percentage.
- Margin drivers: Changes in operating margin, gross margin, SGA ratio, R&D ratio, tax ratio, COGS ratio, D&A ratio, and interest ratio between the two periods, each with direction and delta in percentage points.

Example: normalization ground truth for TRV (2024-12-31, 10-K).. Revenue: $46.4B (high confidence); net income: $5.0B (high); EPS diluted: $21.47 (high); total assets: $133.2B (high); stockholders' equity: $27.9B (high); operating cash flow: $9.1B (high); operating margin: 13.31% (computed); EBITDA: $6.9B (computed, medium confidence). Cross-source agreement: true.

Example: earnings quality ground truth for AAPL (2022-09-24, 10-Q).. Piotroski score: 8/9 (strong). Components: all positive except `delta_gross_margin_positive` = 0. Beneish M-score: −2.31 (no manipulation flag). Component detail: DSRI 1.19, GMI 1.02, AQI 0.91, SGI 1.09, DEPI 1.02, SGAI 0.99, LVGI 1.06, TATA −0.01. Accruals ratio: −0.0097 (high quality). Income quality ratio: 1.16. Flags: none.

Example: driver decomposition ground truth for AAPL (Q2→Q3 2017). Revenue delta: −$7.5B. Top 5 geographic drivers: Greater China −$2.7B (36.4%), Europe −$2.1B (27.5%), Rest of Asia Pacific −$1.1B (14.2%), Japan −$0.9B (11.5%), Americas −$0.8B (10.4%). Margin drivers: operating margin decreased 2.94pp (26.65%→23.71%), driven by SGA ratio +1.30pp, R&D ratio +1.22pp, partially offset by tax ratio −1.20pp.

Task episodes map 1:1 to ground truth entries (2,243 FA tasks total). Each task bundles input references (symbol, period_end, xbrl_ref, tools_available) with a pre-authored natural-language prompt.

Data quality validation. Each domain pipeline ends with automated validation: PC (9 checks), RM (11 checks), FA (12 checks). Checks include coverage, temporal ordering, point-in-time leakage, cross-source agreement (75.4% exact match for FA), covariance positive semi-definiteness, and ground truth consistency. All validation reports are stored as JSON.

A.5 Abbreviated Input/Output Examples by Subtask

This section shows abbreviated input and expected output for one representative task from each subtask. Full payloads are available in the released dataset.

Unconstrained optimization. Input includes objective `max_sharpe`, daily expected returns for 30 tickers, a covariance reference, and the risk-free rate. The expected output is a weight dictionary, e.g., AIG 0.060, BF-B 0.060, LNT 0.060, and CPRT 0.048.

Constrained optimization. Input adds a constraint object to the optimization inputs: `long_only`, maximum weight 0.066, sector cap 0.20, minimum names 18, turnover cap 0.20, and tracking-error cap 0.03. The expected output is a weight dictionary plus a `constraint_satisfaction` map, e.g., `max_weight`, `long_only`, `min_names`, and `max_names` all true.

Tool-use parameterization. Input is a mandate-style natural-language request, such as tracking the benchmark within 3% tracking error, maximum 5% per name, long-only, targeting maximum Sharpe ratio. The expected output is an optimizer-call specification with objective `max_sharpe`, `long_only=true`, `max_weight_per_name=0.05`, and `tracking_error_limit=0.03`.

Rebalancing. Input includes the current portfolio, target objective, covariance reference, and risk-free rate. The expected output includes new weights, a trade list, and total turnover, e.g., AIG increases by 0.011, BAX decreases by 0.039, CMG increases by 0.019, and turnover is 0.15.

Black–Litterman. Input contains market-cap weights, analyst views, and a covariance reference. Example views include a relative AAPL–MSFT view with return 0.05 and confidence 0.7, plus an absolute LNT view with return 0.08 and confidence 0.6. The expected output contains posterior returns and optimized weights.

Constraint monitoring. Input includes portfolio holdings and mandate constraints such as max position 0.05, max sector 0.25, and beta range 0.8–1.2. The expected output reports overall compliance, violation count, and per-constraint

statuses such as AIG failing the position limit, Financial Services failing the sector limit, and beta passing.

Risk identification. Input includes holdings plus market-data references for factor exposures and daily returns. The expected output is a ranked risk list, e.g., factor tilt with market beta 0.997, Financial Services sector concentration at 60%, and AIG single-name exposure at 15%.

Stress testing. Input includes portfolio holdings and a stress scenario such as `stress_rates_down_100`, with interest-rate and equity-market shocks plus sector sensitivities. The expected output reports portfolio P&L and attribution, e.g., estimated P&L of −0.032 with sector and factor contribution entries.

Risk remediation. Input includes a non-compliant portfolio, a mandate ID, and known violations such as `sector_concentration` and `single_name_risk`. The expected output is a trade list and compliance flag, e.g., sell AIG from 0.15 to 0.05, buy ZBRA from 0.00 to 0.03, total turnover 0.25, and post-trade compliance true.

Normalization. Input is a ticker-period-form tuple such as TRV, 2024-12-31, 10-K, with `query_xbrl` available. The expected output contains canonical metrics such as revenue, operating income, net income, diluted EPS, total assets, total liabilities, stockholders' equity, operating cash flow, operating margin, and EBITDA.

Earnings quality. Input is a ticker-period-form tuple such as AAPL, 2022-09-24, 10-Q. The expected output includes Piotroski score and components, Beneish M-score and flag, accruals ratio, income-quality ratio, and flags. Example values include Piotroski 8, Beneish −2.31, accruals ratio −0.0097, and no flags.

Driver decomposition. Input is a ticker and two period ends, such as AAPL from 2017-04-01 to 2017-07-01. The expected output includes revenue delta, top revenue drivers, and margin drivers. Example drivers include Greater China and Europe revenue declines, plus operating-margin and SGA-ratio changes.

B System Prompts and Task Prompts

B.1 FinSkillBench Harness (Experiment 05)

B.1.1 System Prompt. The system prompt is identical across all conditions and subtasks:

> You are a financial analysis agent participating in a controlled evaluation benchmark. Your task is to analyze point-in-time financial data and produce structured outputs. You have access to tools for data retrieval and computation. Use them when needed.
> **Rules:** Use only task-provided data; do not use knowledge after the `as_of_date`. You may call tools to retrieve data and run computations. If skills are available, load a matching skill for guidance; otherwise proceed directly. Submit the final answer with `submit_answer` as JSON matching the expected output schema. Do not answer in plain text, do not refuse the task, and use null for fields that cannot be determined.

B.1.2 Task Prompt Template. Each task prompt is constructed programmatically from the episode JSON. For PC and RM subtasks, the template contains the subtask name, `as_of_date`, difficulty, task description, output instructions, compacted input JSON, and an expected-output schema derived from the ground-truth object.

For FA subtasks, the `task_description` and `output_instructions` are pre-authored natural-language prompts stored in the episode JSON's `prompt` field. For PC and RM subtasks, these are generated programmatically from the input fields.

The `compact_input_json` applies lossy compaction: covariance matrices are replaced with shape summaries (e.g., "30×30, full matrix available via tools"), floats are rounded to 3 significant figures, and the total is capped at 12K characters. The `schema_derived_from_expected_output` replaces ground-truth values with type hints (e.g., `<number>`, `<boolean>`).

B.1.3 Condition-Specific Modifications.

No-skill. The skill-access tools (`load_skill`, `load_references`) are present in the tool schema but the skill corpus is empty: discovery calls resolve to "No skills available." Domain-specific tools (optimizer scripts, risk calculators) are *not* available. The agent has only `get_task_data`, `submit_answer`, and the global `query_xbrl` tool (for FA tasks).

Curated. Human-authored skill packages are mounted. The `load_skill` tool description lists available skills (e.g., "portfolio-construction: Construct optimal portfolio allocations…"). The agent can discover and load SKILL.md content, then call `run_skill_script` to execute validated Python scripts within the skill directory. Domain tools are accessed exclusively through these scripts.

Self-generated. Curated skill material is withheld. The agent receives the `save_skill` tool and an appended instruction:

> **Important: Generate Skills First.** Before solving the task, call `save_skill` one to three times to capture reusable domain knowledge, formulas, workflows, data-access patterns, or edge cases. Then call `load_skill` to read the generated skills back and solve the task using them.

The agent writes SKILL.md files to a per-episode scratch directory, then reloads them. The skill index is rebuilt after each `save_skill` call so subsequent `load_skill` calls can find the newly created skills.

B.2 Hermes Agent Harness (Cross-Harness Validation)

The Hermes harness uses a nearly identical system prompt:

> You are a financial analysis agent participating in a controlled evaluation benchmark. Your task is to analyze point-in-time financial data and produce structured outputs. You have access to deterministic financial tools and should use them when computation is needed.
> **Rules:** Use only task-provided data; do not use knowledge after the `as_of_date`. Call tools when needed, submit the final JSON with `submit_answer`, do not respond in plain text, use null for unknown fields, and do not refuse the task.

The key difference is that the Hermes harness uses a *Pattern B* architecture: heavy data (returns, covariance matrices, factor exposures) is published via a context variable and injected into tool calls server-side, rather than being echoed through model-generated JSON. The user prompt is domain- and subtask-aware, constructed by a dedicated `build_user_prompt()` function that formats portfolio holdings, mandate constraints, stress scenarios, and analyst views into structured sections.

For the curated condition, Hermes's native `skills` toolset is enabled (`skills_list` + `skill_view`), providing progressive disclosure of `SKILL.md` content. The `skill_manage` tool is deregistered to prevent the agent from editing skills during evaluation. All other Hermes built-in toolsets (memory, session search, web, browser, etc.) are disabled to maintain the stateless per-task protocol.

B.3 Tool Schemas

Table 8 summarizes the tools available to the agent across conditions.

Table 8. Tool availability by condition. "Always" means available in all three conditions; "Curated only" means available only when the curated skill package is mounted; "Self-gen only" means available only in the self-generated condition.

Tool	Availability	Description
`submit_answer`	Always	Submit final JSON answer (session-ending)
`load_skill`	Always	Load a `SKILL.md` into context (returns "not found" when empty)
`load_references`	Always	Load supplementary reference documents for a loaded skill
`get_task_data`	Always	Retrieve specific fields from the task input at full precision
`query_xbrl`	Always (FA)	Query the XBRL financial data panel by ticker, period, and metrics
`run_skill_script`	Curated only	Execute a validated Python script from the skill directory
`save_skill`	Self-gen only	Write a `SKILL.md` to the per-episode scratch directory

C Detailed Subtask Descriptions

This section describes each of the 12 subtasks in detail: what the agent receives as input, what it must produce as output, why the task is difficult, and what cognitive or procedural capabilities it tests.

C.1 Portfolio Construction

C.1.1 Unconstrained Optimization (40 tasks).

Input. A dictionary of expected returns per ticker (30 daily return estimates), a reference to a 30×30 Ledoit–Wolf covariance matrix, a risk-free rate, an objective (`max_sharpe`, `min_variance`, `max_return`, or `risk_parity`), and a risk aversion parameter.

Output. A weight dictionary (`{ticker: float}`) summing to approximately 1.0.

Difficulty. The agent must solve a convex optimization problem. Without the curated skill (which provides a validated `optimize.py` script), the agent must either implement mean-variance optimization from scratch within the turn budget or reason about the solution analytically. The covariance matrix contains 900 floating-point values; passing it through model-generated JSON introduces precision loss that changes the optimum. For `max_return`, the unconstrained solution is a single-stock portfolio (100% in the highest-return ticker), which is mathematically correct but counterintuitive.

Cognitive demands. Numerical optimization, matrix algebra, understanding of mean-variance theory [10], tool parameterization, precision management.

Scorer. `l2_distance_and_objective`: score $= \max(0, \min(1, 1 - L_2/(4 \times \theta)))$ where L_2 is the Euclidean distance between predicted and expected weight vectors and $\theta = 0.05$.

C.1.2 Constrained Optimization (40 tasks).

Input. Same as unconstrained, plus a constraint object with fields such as `long_only`, `max_weight`, `min_weight`, `sector_limits`, `min_names`, and `max_names`, and optionally `max_turnover`, `max_tracking_error`, and `factor_neutrality`. A benchmark weight dictionary and sector mapping are also provided.

Output. A weight dictionary plus a `constraint_satisfaction` dictionary mapping each constraint name to a boolean pass/fail.

Difficulty. The agent must correctly translate natural-language and structured constraint specifications into optimizer parameters. Constraint interactions are non-trivial:

sector limits combined with position limits and minimum-names requirements can produce solutions far from the unconstrained optimum. The agent must also correctly report which constraints are satisfied—a separate verification step from computing the weights.

Cognitive demands. Constraint formulation, understanding of portfolio mandate structures, multi-objective reasoning, tool parameterization with nested constraint objects.

Scorer. `constraint_satisfaction_and_objective`: exact match on all expected `constraint_satisfaction` keys acts as a gate. Only if the gate passes is weight L_2 scored; missing or incorrect constraint map → score 0.

C.1.3 Tool-Use Parameterization (40 tasks).

Input. A natural-language scenario description (e.g., "Build a diversified equity portfolio tracking the benchmark within 3% tracking error, maximum 5% per name, long-only"), tool documentation for the optimizer API, and references to covariance and return data.

Output. A structured `optimizer_call` object with the correct `objective` string and nested `constraints` dictionary matching the scenario description.

Difficulty. This subtask isolates the *translation* step: converting a mandate description into correct API parameters. The agent does not need to run the optimizer—it only needs to produce the correct call specification. Errors include wrong objective selection, missing constraints, incorrect numeric thresholds, and wrong parameter names. The task tests whether the agent can read API documentation and map domain concepts to tool interfaces.

Cognitive demands. Natural language comprehension, API contract understanding, parameter mapping, attention to numeric detail.

Scorer. `parameter_match`: recursively compares the union of predicted and expected parameter leaves, so missing and extra leaves are both penalized. Numeric values match if relative error $< 20\%$; categorical values require exact match. Score = fraction of matching leaves.

C.1.4 Rebalancing (40 tasks).

Input. A current portfolio (weight dictionary from a previous optimization), a current portfolio date, a covariance matrix reference, a risk-free rate, and an objective.

Output. A `trade_list` (signed weight changes per ticker), `new_weights` (post-rebalance weights), and `turnover` (total one-way turnover).

Difficulty. The agent must compute optimal new weights *and* derive the trade list and turnover from the difference between current and new weights. Turnover calculation has a specific convention (one-way: $\sum |w_{\text{new}} - w_{\text{old}}|/2$). The trade list must be consistent with the weight change—a common failure mode is producing trades that do not sum to the correct weight delta.

Cognitive demands. Portfolio rebalancing mechanics, trade arithmetic, turnover conventions, multi-output consistency.

Scorer. `turnover_compliance_and_objective`: weighted combination of post-rebalance weight L_2 score (50%), turnover relative-error ramp (25%), and trade-list magnitude consistency over expected tickers (25%).

C.1.5 Black–Litterman (40 tasks).

Input. Market-cap weights (equilibrium prior), a covariance matrix reference, a risk-free rate, and 2–4 analyst views. Each view specifies a type (`absolute` or `relative`), symbols, expected return, and confidence level.

Output. `posterior_returns` (per-ticker posterior expected returns) and `optimal_weights` (weights from optimizing on the posterior).

Difficulty. The Black–Litterman model [3] requires: (i) computing equilibrium returns from market-cap weights and the covariance matrix, (ii) constructing the pick matrix P and view vector Q from the analyst views, (iii) computing the posterior via the BL formula involving τ, Σ, P, Q, and Ω, and (iv) optimizing on the posterior returns. Each step can introduce errors. A common failure is using `expected_returns` as the prior instead of deriving equilibrium returns from market-cap weights. The model is sensitive to τ and confidence parameters.

Cognitive demands. Bayesian updating, matrix algebra, understanding of equilibrium models, multi-step financial computation, correct handling of relative vs. absolute views.

Scorer. `view_specification_and_weights`: averages (i) posterior returns MAE score $(\max(0, 1 - \text{MAE}/0.05))$ and (ii) posterior weights L_2 score $(\max(0, \min(1, 1 - L_2/(4 \times 0.10))))$.

C.2 Risk Management

C.2.1 Constraint Monitoring (40 tasks).

Input. A portfolio object (holdings with symbols and weights, total NAV), a mandate specification (position limits, sector limits, beta range, long-only, turnover limits), and market data references.

Output. `overall_compliant` (boolean), `violations_count` (integer), and a `constraints` array where each entry has a `type`, `status` (PASS/FAIL), worst offender, actual value, and limit.

Difficulty. The agent must evaluate 4–6 heterogeneous constraints against the portfolio. Constraint types use different units (weights, percentages, ratios, booleans) and require different computations (max over positions, sum over sectors, dot product for beta). The agent must also correctly classify the overall compliance status and count violations. Type normalization is non-trivial: the same constraint may be labeled `position_limit_max`, `max_position`, or `max_weight` across different mandates.

Cognitive demands. Mandate interpretation, multi-constraint evaluation, portfolio analytics, attention to edge cases (e.g., beta range is a two-sided constraint).

Scorer. `exact_match`: $0.4 \times \mathbb{1}$[compliant match] $+ 0.6 \times$ constraint accuracy, where constraint accuracy is the fraction of expected constraint types whose predicted status matches after type and status normalization. The stricter `jb` scorer variant also withholds the compliance credit unless a constraint list is supplied and penalizes contradictory duplicate predicted constraint types.

C.2.2 Risk Identification (40 tasks).

Input. A portfolio object and market data references (factor exposures, daily returns).

Output. An array of up to 5 ranked risk exposures, each with `rank`, `type` (e.g., `factor_tilt`, `sector_concentration`, `single_name`, `concentration`), `detail` (specific description), and `magnitude`.

Difficulty. This is the most open-ended RM subtask. The agent must: (i) compute concentration metrics (HHI, top-N weight share), (ii) analyze sector distributions, (iii) review factor exposures for notable tilts, (iv) synthesize these into a ranked list ordered by magnitude. The ground truth is derived from a deterministic risk engine, but the agent must independently identify and rank the same risks. Without the curated skill's `compute_risk_report.py` script, the agent must perform all computations from raw data. The no-skill baseline is near zero (0.023) because most agents cannot independently compute factor exposures and concentration metrics within the turn budget.

Cognitive demands. Portfolio risk analytics, factor model interpretation, concentration analysis, multi-criteria ranking, domain knowledge of risk taxonomy.

Scorer. `ranked_list_recall`: $0.4 \times$ recall@k $+ 0.3 \times$ precision@k $+ 0.3 \times$ NDCG@k, where recall and precision use multiset matching on risk types, and NDCG uses graded relevance from expected magnitudes. $k = 5$. In the `exp05` scorer used by the all-fin Hermes runs, NDCG maps each risk type to the maximum expected magnitude among the top-k expected items and does not clamp the final composite; duplicate predicted types can therefore make this scorer slightly exceed 1. The stricter `jb` scorer variant uses greedy one-to-one duplicate matching and reports DCG/IDCG diagnostics.

C.2.3 Stress Testing (40 tasks).

Input. A portfolio object, a stress scenario specification (scenario ID, name, description, market shocks as factor-level percentage changes), and factor exposure references.

Output. `estimated_pnl_pct` (decimal portfolio return, e.g., -0.167 for -16.7%) and `attribution` with `by_sector` and `by_factor` arrays.

Difficulty. The agent must: (i) resolve the stress scenario (which may be a historical replay or hypothetical), (ii) apply factor-level shocks to the portfolio's factor exposures to estimate P&L, (iii) decompose the P&L into sector and factor contributions. A critical failure mode is unit confusion: the scenario specifies shocks as percentages (e.g., "equity -20%"), but the P&L must be reported as a decimal return. Another failure mode is attempting to estimate P&L from headline shock percentages rather than using the full factor model—this produces systematically wrong answers because it ignores cross-factor correlations and idiosyncratic risk.

Cognitive demands. Factor-based stress testing, P&L attribution, unit conversion discipline, understanding of historical replay vs. hypothetical scenarios.

Scorer. `absolute_error`: $s_{\text{pnl}} = \max(0, \min(1, 1 - (|\hat{y} - y| - 0.1\tau)/(3\tau)))$ where τ is the P&L tolerance (typically 1%). When expected sector attribution is scored, the final score is $0.7\, s_{\text{pnl}} + 0.3\, s_{\text{attr}}$. In `exp05`, s_{attr} is the mean per-expected-sector relative-error ramp with a default 5% relative tolerance; missing expected sectors receive zero and extra predicted sectors are ignored. In the stricter `jb` scorer, sector attribution is enabled when `sector_attribution_mae` appears in `verification.metrics`, and s_{attr} is a triangular ramp on absolute MAE over the union of predicted and expected sectors.

C.2.4 Risk Remediation (40 tasks).

Input. A non-compliant portfolio, a mandate specification, a list of known violations (e.g., "sector_concentration", "single_name_risk"), and market data references including transaction costs.

Output. A `trades` array (each with symbol, action, current weight, target weight, trade percentage), `total_turnover`, and `post_trade_compliant` (boolean).

Difficulty. The agent must generate a feasible set of trades that restores compliance while minimizing turnover. This requires: (i) understanding which constraints are violated and by how much, (ii) determining which positions to reduce and which to increase, (iii) ensuring the post-trade portfolio satisfies all constraints simultaneously, (iv) considering diversification benefits of adding new positions from

the broader universe. The curated skill provides a constrained optimizer (`remediate_portfolio.py`) that solves this as a minimum-turnover optimization; without it, the agent must reason about trade feasibility manually.

Cognitive demands. Constrained portfolio optimization, trade generation, multi-constraint satisfaction, turnover minimization, understanding of remediation as an optimization problem.

Scorer. `constraint_satisfaction_plus_cost`: $0.3 \times \mathbb{1}$[compliant match] $+ 0.3 \times \mathbb{1}$[turnover OK] $+ 0.4 \times F_1$(trade pairs), where trade pairs are normalized (symbol, action) tuples. The `exp05` scorer computes a multiset F_1; the stricter `jb` scorer computes set-based precision/recall over normalized trade pairs.

C.3 Fundamental Analysis

C.3.1 Normalization (100 tasks evaluated; 244 total).

Input. A ticker symbol, a reporting period end date, a form type (10-Q or 10-K), and access to the `query_xbrl` tool.

Output. A `metrics` dictionary with 10 canonical financial metrics: revenue, operating_income, net_income, eps_diluted, total_assets, total_liabilities, stockholders_equity, operating_cash_flow, operating_margin, and ebitda.

Difficulty. The agent must: (i) query the XBRL panel for the correct ticker and period, (ii) map returned fields to the expected output schema, (iii) compute derived metrics that are not directly available (operating_margin = operating_income / revenue; ebitda = operating_income + depreciation_amortization). The task is relatively straightforward when the `query_xbrl` tool is available, which explains the high no-skill baseline (0.681). The curated skill provides no additional benefit ($\Delta = -0.003$) because the global XBRL query tool already provides direct access to the data.

Cognitive demands. Financial statement literacy, XBRL data navigation, metric computation, understanding of GAAP reporting conventions.

Scorer. `metric_absolute_error`: unwraps a single `{metrics: {...}}` container when both predicted and expected outputs use it. For each scored metric, match if relative error $\leq$ its per-metric tolerance (default 5%); metadata fields such as `source`, `confidence`, `tier`, and `xbrl_fmp_agreement` are ignored. Score = fraction of scored metrics that match.

C.3.2 Earnings Quality (100 tasks evaluated; 1,002 total).

Input. A ticker symbol, a reporting period end date, a form type, and access to the `query_xbrl` tool.

Output. Piotroski F-Score (integer 0–9), 9 binary Piotroski components (e.g., `roa_positive`, `delta_leverage_negative`), Beneish M-Score (float), `beneish_flag` (boolean: $M > -1.78$), accruals ratio, income quality ratio, and a flags array.

Difficulty. The agent must: (i) retrieve financial data for both the current and prior comparable period (same quarter previous year), (ii) compute 9 Piotroski components requiring year-over-year comparisons of profitability, leverage, and efficiency metrics, (iii) compute the Beneish M-Score from 8 ratio components (DSRI, GMI, AQI, SGI, DEPI, SGAI, LVGI, TATA), (iv) compute accruals and income quality ratios, (v) flag concerns. The multi-period requirement doubles the data retrieval burden. Delta components (e.g., `delta_roa_positive`) require correct identification of the prior period and correct sign interpretation.

Cognitive demands. Multi-period financial analysis, Piotroski F-Score methodology [12], Beneish M-Score methodology [2], ratio computation, temporal reasoning about comparable periods.

Scorer. `earnings_quality_composite`: renormalized weighted mean over active components: Piotroski component accuracy (episode weight 40%), Beneish flag match (30%), and flag F_1 (30%). Numeric ratio accuracy for Beneish M-score, accruals ratio, and income-quality ratio is included only when `numeric_ratio_weight` > 0 in `exp05`/`zqbok_experiment05`; the stricter `jb` scorer defaults that weight to 0.3 if selected.

C.3.3 Driver Decomposition (100 tasks evaluated; 997 total).

Input. A ticker symbol, current and prior period end dates, and access to the `query_xbrl` tool.

Output. `revenue_delta` (absolute change in revenue), `top_revenue_drivers` (up to 5 named drivers with type, direction, magnitude in USD, and contribution percentage), and `margin_drivers` (operating margin, gross margin, SGA ratio, etc., with direction and delta in percentage points).

Difficulty. This is the most challenging FA subtask. The agent must: (i) retrieve aggregate financial data for two periods, (ii) compute the revenue delta, (iii) attribute the revenue change to specific business or geographic segments—but the XBRL panel contains only aggregate data, not segment-level data. The curated skill's `driver_decomposition.py` script can process segment data when available, but the agent must use domain knowledge about the company's business segments to attribute revenue changes when segment data is unavailable from the tool. (iv) Compute margin drivers by comparing expense ratios across periods. The ground truth is derived from revenue segmentation data, creating a ceiling effect: agents without access to segment data cannot fully match the expected drivers.

Cognitive demands. Revenue attribution, segment analysis, margin decomposition, domain knowledge of company business structures, multi-period comparison, understanding of contribution analysis.

Scorer. `driver_f1_and_direction`: averages active component scores: revenue-driver name recall/precision (case-insensitive set matching), direction accuracy for matched revenue drivers, numeric magnitude/contribution accuracy for matched revenue drivers when enabled (default enabled), margin-driver recall plus direction accuracy, numeric margin Δ accuracy when enabled, and revenue-delta relative-error accuracy.

C.4 Summary of Cognitive Demands

Table 9 maps each subtask to the primary cognitive and procedural capabilities it tests.

Table 10 provides the complete mathematical specification of each scorer. The primary formulas match `experiments/zqbok_experiment05/lib/scorers.py` and `experiments/jb_experiment/scoring/scorers_exp05.py`, which are the `exp05` scorers used by the all-fin Hermes runs. Where `experiments/jb_experiment/scoring/scorers.py` differs, the stricter `jb` variant is noted in the subtask descriptions above.

Table 9. Cognitive and procedural demands by subtask. ✓= primary demand; ○ = secondary demand.

Subtask	*Numerical optim.*	*Tool param.*	*Multi-step comp.*	*Domain knowledge*	*Data retrieval*	*Precision mgmt.*	*Multi-output*
Unconstrained optim.	✓	✓	○	○	○	✓	
Constrained optim.	✓	✓	○	✓	○	✓	✓
Tool-use param.		✓		✓			
Rebalancing	✓	✓	✓	○	○	✓	✓
Black–Litterman	✓	✓	✓	✓	○	✓	✓
Constraint monitoring			✓	✓	○		✓
Risk identification			✓	✓	✓		✓
Stress testing		○	✓	✓	✓	✓	✓
Risk remediation	✓	○	✓	✓	○		✓
Normalization			○	✓	✓		
Earnings quality			✓	✓	✓		✓
Driver decomposition			✓	✓	✓		✓

Table 10. Complete scorer specifications. Most component scores are clipped or averaged into $[0, 1]$; the `exp05` ranked-list scorer does not final-clip NDCG or the composite and can slightly exceed 1 when duplicate predicted risk types receive repeated relevance credit.

Scorer	Subtask	Formula
L2 distance	Unconstrained optim.	$\max\big(0,\ \min\big(1,\ 1 - \|w_p - w_e\|_2 \,/\, (4\theta)\big)\big),\quad \theta$=0.05
Constraint gate + L2	Constrained optim.	All expected `constraint_satisfaction` keys must be present and equal; extra predicted keys are ignored. If the gate passes, weight L_2 score as above; else 0
Parameter match	Tool-use param.	Fraction of union leaf parameters matching (numeric: relative error < 0.2; categorical: exact; extra/missing leaves count against the denominator)
Turnover compliance	Rebalancing	$0.5 \cdot s_{L_2} + 0.25 \cdot s_{\text{turn}} + 0.25 \cdot s_{\text{trade}}$, where $s_{L_2} = \max(0, \min(1, 1 - L_2/0.2))$, $s_{\text{turn}} = \max(0, 1 - \min(1, \lvert\hat{T} - T\rvert / \max(\lvert T\rvert, 10^{-9})))$, and s_{trade} is mean expected-ticker trade consistency
View + weights	Black–Litterman	Mean of active components: posterior-returns MAE score $\max(0,\ 1 - \text{MAE}/0.05)$ and weight L_2 score $\max(0, \min(1, 1 - L_2/(4\theta)))$ with θ=0.10
Exact match	Constraint monitoring	$0.4 \cdot \mathbb{1}[\text{compliant match}] + 0.6 \cdot$ constraint accuracy, where accuracy is status match over expected normalized constraint types
Ranked recall	Risk identification	$0.4 \cdot \text{recall@}k + 0.3 \cdot \text{precision@}k + 0.3 \cdot \text{NDCG@}k$; recall/precision use multiset type overlap; `exp05` NDCG uses max magnitude per expected type and is not final-clipped
Absolute error	Stress testing	P&L: $s_{\text{pnl}} = \max\big(0,\ \min\big(1,\ 1 - (\lvert e\rvert - 0.1\tau)/(3\tau)\big)\big)$; with scored sector attribution, $0.7 \cdot s_{\text{pnl}} + 0.3 \cdot s_{\text{attr}}$; else s_{pnl} alone
Cost + compliance	Risk remediation	$0.3 \cdot \mathbb{1}[\text{compliant match}] + 0.3 \cdot \mathbb{1}[\text{turn. OK}] + 0.4 \cdot F_1(\text{normalized trade pairs})$
Metric MAE	Normalization	Fraction of scored non-metadata metrics with relative error ≤ per-metric tolerance (default 5%); unwraps `metrics` wrapper when present on both sides
EQ composite	Earnings quality	Weighted mean over active terms: component accuracy (episode weight 0.4), Beneish-flag match (0.3), flag F_1 (0.3); numeric-ratio accuracy is included only if `numeric_ratio_weight`>0 in `exp05/zqbok_experiment05`
Driver F_1	Driver decomposition	Mean of active terms: revenue-driver recall/precision, matched revenue-driver direction, matched revenue numeric accuracy, margin recall/direction, matched margin numeric accuracy, and revenue-Δ accuracy