PERSPECTIVEGAP: A BENCHMARK FOR MULTI-AGENT ORCHESTRATION PROMPTING

Anonymous authors
Paper under double-blind review

ABSTRACT

In a multi-agent LLM system, the main agent must write each sub-agent's prompt, yet current models struggle to determine what each sub-agent needs to know. Existing benchmarks score belief tracking or downstream task success, not the prompts themselves. We introduce PerspectiveGap, a benchmark for evaluating LLMs' ability to compose orchestration prompts for multi-agent systems. PerspectiveGap contains 110 scenarios, each evaluated through two distractor-mixed task formats: role-fragment assignment and verbatim prompt assembly, in which the model assembles each role's prompt from the given fragments. These scenarios are organized into 10 loop-centered topologies distilled from the authors' real-world engineering practice. Across 35 commercial models from 10 companies, the average combined pass rate is only 17.6%, and in role-fragment assignment, models give roles an average of 2.1 out-of-role fragments per scenario. GPT-5.5 leads by a wide margin (62.0% pass rate). These results show that current models often fail to give each sub-agent exactly the context it needs, an ability that existing benchmarks do not measure.

1 INTRODUCTION

When a main agent orchestrates sub-agents, it must write each sub-agent's prompt, specifying that role's task scope, context boundaries, and expected handoffs; we call this orchestration prompting. Such orchestrations, which decompose a task into specialized roles that exchange messages and artifacts, are now common (Li et al., 2023; Qian et al., 2023; Hong et al., 2023; Wu et al., 2023; Chen et al., 2023; Lu et al., 2024; Tran et al., 2025). Yet current LLMs struggle to determine what each sub-agent needs to know. Main agents leak distractors, expose out-of-role information, drop shared context, and confuse artifact ownership. These errors produce incomplete or contaminated sub-agent prompts. Section 6 analyzes these failure modes in detail.

Existing evaluations do not target this orchestration-prompting ability. Theory-of-mind (ToM) benchmarks typically score question answering or belief tracking (Le et al., 2019; Kim et al., 2023; Sclar et al., 2024), multiple-choice action prediction (Gu et al., 2024), information management in dialogue (YS et al., 2026), or interaction regret (Riemer et al., 2025). Agent benchmarks instead score downstream task success, tool use, environment-level performance, or instruction following (Liu et al., 2023; Qin et al., 2023; Orogat et al., 2026; Qi et al., 2025). Neither line directly evaluates whether a main agent can write sub-agent prompts that respect asymmetric context and role-specific information needs. Section 2 discusses this boundary.

We introduce PerspectiveGap, to our knowledge the first benchmark for multi-agent orchestration prompting. PerspectiveGap contains 110 scenarios, each consisting of a role list, a shuffled set of information fragments $f_1, \ldots, f_N$, and a reference answer specifying which fragments each role needs. Each scenario includes two tasks: role-fragment assignment and verbatim prompt assembly. The former asks the model to output the fragment IDs for each role; the latter asks the model to assemble the actual sub-agent prompts by pasting the fragments each role needs. The two tasks share the same fragments, so together they measure both steps of orchestration prompting: choosing each role's information and writing it into that role's prompt. Each scenario also injects one distractor, such as prompt-engineering advice, that may look useful to the tested LLM but is useless or even burdensome for any sub-agent. The benchmark covers orchestrations with 2–6 roles and 7–13 information fragments. A deterministic, rule-only scorer evaluates each role's prompt for

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

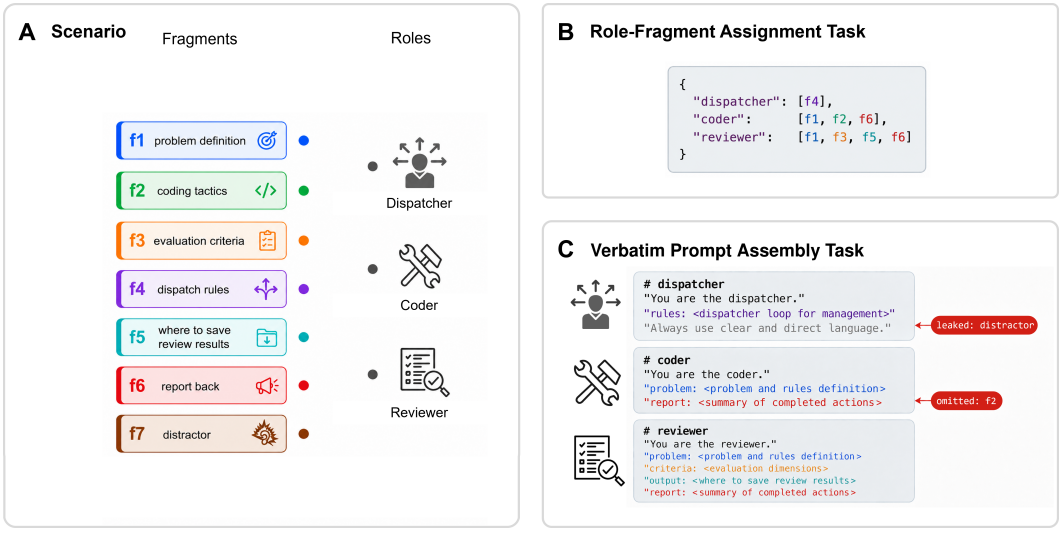


Figure 1: Overview of PerspectiveGap. (A) A scenario presents a set of information fragments $f_1, \ldots, f_N$, one of which is a distractor (f_7), and a set of sub-agent roles (here dispatcher, coder, and reviewer); each role needs only a specific subset of the fragments. (B) The role-fragment assignment task asks the model to output that subset for each role, as fragment identifiers. (C) The verbatim prompt assembly task asks the model to write each role's actual prompt from the same fragments, scored by checking that each prompt contains its required fragments and excludes out-of-role content. The example prompts illustrate the two error types the benchmark targets: a leaked distractor in the dispatcher prompt and an omitted required fragment (f_2) in the coder prompt. Fragment text is abbreviated for display.

inclusion of required fragments and exclusion of irrelevant content; a 716-row hand audit validates this containment detector. The construction pipeline records each scenario's topology, role definitions, fragments, and reference answers explicitly, allowing new scenarios to be added with auditable answer keys.

The 110 scenarios are organized into 10 topologies shown in Figure 2. These topologies are distilled from the authors' real-world prompt-engineering practice and from production agent-system patterns. Their loop-centered structure follows the Prompt Economy framing (Section 4): a role prompt is written once but runs many times, so a small set of reusable roles is preferable to many agents. The release uses six base patterns plus four pool variants, all built around one or more critic loops. As a set, they cover the main practical one- and two-loop orchestration patterns targeted by PerspectiveGap.

We evaluate 35 commercial models from 10 companies on PerspectiveGap. GPT-5.5 leads with a 62.0% combined pass rate; gpt-5.6-terra and grok-4.6 follow at 42.7% and 40.9%, and models ranked 4–9 cluster between 25% and 36%. The average combined pass rate is 17.6%. In role-fragment assignment, models give roles an average of 2.1 out-of-role fragments per scenario, compared with 0.5 for GPT-5.5. These results show that current models often fail to give each sub-agent exactly the context it needs.

This paper makes four contributions:

1. We build PerspectiveGap, a 110-scenario benchmark with two task formats per scenario, per-scenario answer keys, construction notes, and full logs, to be released upon publication.
2. We evaluate 35 commercial models from 10 companies and report a leaderboard under each provider's default settings, revealing low average performance.
3. We identify four recurring failure modes in multi-agent orchestration prompting.
4. We articulate **Prompt Economy** as the design framing behind the loop-centered topology family.

2 Related Work

Multi-agent orchestration. LLM applications increasingly split work across specialized workers, critic loops, and pipelines. Early multi-agent systems such as CAMEL, ChatDev, MetaGPT, AutoGen, and AgentVerse showed that role-specialized LLMs can collaborate through natural-language messages and structured workflows (Li et al., 2023; Qian et al., 2023; Hong et al., 2023; Wu et al., 2023; Chen et al., 2023). Subsequent surveys and pattern catalogues describe recurring orchestration motifs such as routing, planning, reflection, critique, and tool-mediated handoff (Tran et al., 2025; Liu et al., 2024; Dao et al., 2026; Gullí, 2025). Industrial guidance from Anthropic makes a similar distinction between workflows with prescribed control flow and more autonomous agentic systems, emphasizing task decomposition, routing, delegation, and tool-interface design as central engineering concerns (Schluntz & Zhang, 2024). Recent applied systems use multi-agent or multi-stage designs in optimization, scientific computing, data discovery, and automated research (Lu et al., 2024; Thind et al., 2025; Du et al., 2026; Sun et al., 2026). These works motivate our setting, but they do not test the fragile step we study: whether an LLM can write role-specific instructions that preserve context boundaries and handoff contracts instead of flattening every role into the same prompt.

Agent and tool-use benchmarks. AgentBench (Liu et al., 2023), ToolBench from ToolLLM (Qin et al., 2023), and related benchmark suites evaluate execution once the task, the available tools or APIs, and the instructions are given. They ask whether an agent can act in an environment, call the right tool, or complete a workflow under an existing scaffold. Other recent evaluations compare multi-agent frameworks, delegation behavior, instruction following, and failure attribution in existing agentic systems and applications (Orogat et al., 2026; Gao et al., 2026; Qi et al., 2025; Zhang et al., 2025b). TeamBench sharpens this scaffold further, enforcing role information boundaries at the OS level and studying whether agent teams coordinate well under that fixed separation (Kim et al., 2026); the boundary itself is fixed by the benchmark, not authored or scored as a model output. PerspectiveGap moves the evaluation one step earlier: before any worker acts, can the main agent allocate context and constraints into prompts that downstream workers can safely use?

Information asymmetry and ToM benchmarks. Theory-of-mind benchmarks study information asymmetry, but they do not ask the model to produce prompt artifacts. ToMi, FANToM, Hi-ToM, OpenToM, BigToM, and ExploreToM test belief tracking, higher-order mental-state reasoning, or adversarially generated belief structures through question answering and classification (Le et al., 2019; Kim et al., 2023; He et al., 2023; Xu et al., 2024; Gandhi et al., 2023; Sclar et al., 2024). Work on perspective-taking and hypothesis-driven ToM reasoning shows that making viewpoints explicit can improve model behavior (Wilf et al., 2023; Kim et al., 2025), and strategic social reasoning benchmarks expose planning failures in collaboration and negotiation (Yao et al., 2025). SimpleToM separates explicit mental-state inference from applying it to predict and judge behavior (Gu et al., 2024), and recent position and behavior-based work separates it from acting on it (Riemer et al., 2025; Ackerman, 2026). Our two tasks separate a related pair of steps in output form: assignment asks which fragments each role needs, and prompt assembly asks the model to place them in each role's prompt. SOTOPIA-TOM is closest to our setting because it studies information management in multi-agent interaction (YS et al., 2026). Yet even SOTOPIA-TOM, like the rest, tests belief tracking, action choice, or dialogue behavior inside an interaction. PerspectiveGap tests orchestration prompting: constructing the instructions that define each role's view of the task before the interaction begins.

Distributed information and multi-agent failures. Several recent benchmarks show that multi-agent LLM systems fail under distributed information even when individual models are strong. HiddenBench finds a large gap between collective reasoning with distributed information and single-agent reasoning with complete information (Li et al., 2025). Silo-Bench similarly shows that agents may exchange enough information but fail to integrate it into a correct joint answer (Zhang et al., 2026). MAST categorizes multi-agent failures and identifies inter-agent misalignment, such as withheld information and ignored input from other agents, as a recurring source of breakdowns (Cemri et al., 2025). Privacy and leakage benchmarks such as AgentLeak study how information can escape across a running multi-agent stack (Yagoubi et al., 2026). These works diagnose failures after agents interact; PerspectiveGap isolates a precursor artifact: the role prompts that determine each agent's initial information boundary.

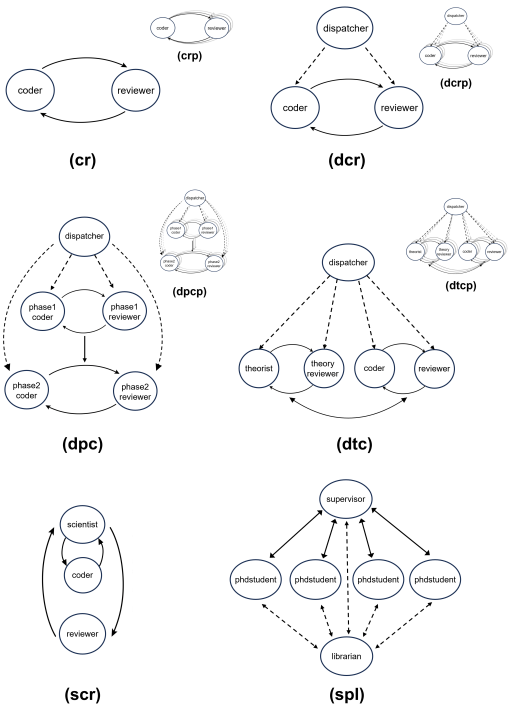


Figure 2: Base topology patterns in PerspectiveGap. Each pattern is a role-and-handoff graph: nodes are sub-agent role types, and repeated nodes of one type (e.g., the students in spl) are parallel instances of one role prompt; solid edges are the actor–reviewer feedback loops and the handoffs between them, and dashed edges are distribution or support links (a dispatcher feeding workers, or a shared librarian). The six base patterns span a single coder–reviewer loop (cr), a dispatcher-fed loop (dcr), sequential and parallel two-loop systems (dpc and dtc, respectively), a scientist–coder–reviewer arrangement (scr), and a supervisor–student–librarian hub (spl). Four of the six have pool variants (insets crp, dcrp, dpcp, dtcp) that add a managed pool of candidate ideas or plans, for 10 topologies in total. For readability, the dispatcher of scr and the plan refiner of dpcp are not drawn; Appendix Table 11 lists every topology's role count.

Prompt optimization and role-state management. Prompting and workflow methods can improve downstream behavior through refinement, role evolution, memory, or training-time optimization (Madaan et al., 2023; Chang & Geng, 2025; Mo et al., 2025). Other multi-agent methods make collaborator knowledge explicit, which is closely related to our premise that the main agent must reason about what each sub-agent knows and needs (Zhang et al., 2025a). However, these methods are usually evaluated by final task success or workflow reliability. PerspectiveGap instead evaluates the generated prompt artifact directly: whether it assigns information correctly can be read from the prompts themselves, not inferred from whether the downstream task succeeds.

3 PERSPECTIVEGAP BENCHMARK DESIGN

Scenario schema. As shown in Figure 1, each scenario is a small orchestration problem with three explicit parts: a list of sub-agent roles, a shuffled list of information fragments $f_1, \ldots, f_N$, and a reference role-to-fragment assignment stored with the scenario. The model sees the roles and the shuffled fragments, but not the reference assignment. The reference assignment applies the need-only rule: each role receives exactly the fragments it needs to do its stated job. A fragment written for one role is withheld from the others even when their work is related. Extra fragments lengthen a role's context without helping it, and some are harmful: if a reviewer's pass criterion, such as rejecting any plan that runs longer than two hours, reaches the planner, the planner can tailor every plan to just under the threshold instead of solving the task. This rule sets the role-context boundary used for scoring. In this sense, the reference assignment is the task contract: given the stated roles and the need-only rule, the model must preserve that boundary in both evaluated formats. All 110 scenario mappings were constructed under this rule and inspected by all authors for consistency. To test whether the rule determines a unique answer, five volunteers who were not involved in building the benchmark each solved all 110 scenarios in the role-fragment assignment format, seeing the same background, role list, and fragments as the models but not the reference assignments; every solution matched the reference. Appendix N gives an actual benchmark scenario, including the background, the full text of every fragment, the *"need-only"* instruction shown to the model, the reference assignment, and the textual evidence behind representative boundary decisions.

Deterministic rendering. The benchmark separates topology, instance, and scenario. A topology is one of the 10 role-and-handoff patterns used in the paper; an instance is a domain-specific realization

of a topology; a scenario is the rendered benchmark item given to the model. For each scenario, a shuffle seed fixes the displayed fragment order; the displayed identifiers are then relabeled as $f_1, \ldots, f_N$ in that order, as choices in a multiple-choice question would be relabeled after shuffling. Changing the seed changes the presentation order and displayed identifiers, but not the role definitions, fragment content, distractor, or reference assignment. This gives a deterministic way to test whether a model is assigning information by the need-only rule rather than by position in the prompt. The same topology-instance split also gives the benchmark breadth: Appendix M lists the professional-domain coverage. Across these domains, the same construction rule determines which fragments each role needs.

Extensibility. The topologies are reusable skeletons, not one-off examples. These 10 topologies cover the main practical one- and two-loop orchestration patterns targeted by PerspectiveGap[1]; more complex orchestrations can be composed from them as building blocks. New instances can be added by choosing a domain, naming the roles, writing the fragments, and applying the same reference-assignment rule; the renderer and scorers do not change.

Distractor insertion. Each main benchmark scenario includes one distractor fragment. A representative distractor used in the benchmark is:

> Best practices: Use consistent, descriptive tag names across your prompts. Nest tags when content has a natural hierarchy, such as documents inside `<documents>` and each document inside `<document index="n">`.

This looks useful to the tested model while it is writing prompts, but it is not useful to any downstream sub-agent. The tested model must separate information that helps itself compose the orchestration from information that a sub-agent needs to do its job. Leaking irrelevant content can add cognitive load and degrade the sub-agent's output; in higher-stakes settings, it can also expose out-of-role information that enables reward hacking or violates safety constraints.

Two evaluated formats. Each scenario is rendered in two formats from the same shuffled fragments. Role-fragment assignment asks for a structured mapping from roles to fragment identifiers. Verbatim prompt assembly asks for the final sub-agent prompts. Pairing the two formats separates choosing each role's fragments from writing them into natural-language instructions. The benchmark therefore reports the two formats separately and also reports their combined score.

Evaluation metrics. We treat Strict pass as the primary endpoint, report Net match score as a partial-credit companion, and use Required coverage, Boundary precision, Overall leakage, and Distractor leakage as diagnostic metrics. For each evaluation e (one model response to one scenario, shuffle seed, and task format), we first aggregate events over all requested roles: TP_e counts correctly included reference role-fragment events, FP_e counts extra out-of-role events, and FN_e counts omitted reference events. Strict pass is 1 iff $\mathrm{FP}_e = \mathrm{FN}_e = 0$. The average Strict pass over a task format's evaluations is that format's pass rate, and the average over all evaluations is the combined pass rate. This all-or-nothing criterion matches the benchmark's boundary-preservation objective: either an output preserves all requested role boundaries, or it contains an omission or cross-role inclusion. Net match score is $\max(0, (\mathrm{TP}_e - \mathrm{FP}_e - \mathrm{FN}_e)/(\mathrm{TP}_e + \mathrm{FN}_e))$ for each evaluation, averaged over evaluations. Required coverage and Boundary precision are micro-averages over evaluations: $\sum_e \mathrm{TP}_e / \sum_e (\mathrm{TP}_e + \mathrm{FN}_e)$ and $\sum_e \mathrm{TP}_e / \sum_e (\mathrm{TP}_e + \mathrm{FP}_e)$. Overall leakage is $\mathrm{avg}_e\, \mathrm{FP}_e$, counting every out-of-role leak event per role; Distractor leakage is its restriction to the injected distractor, $\mathrm{avg}_e\, \mathrm{D}_e$, where D_e counts only injected-distractor leak events and thus $\mathrm{D}_e \leq \mathrm{FP}_e$. Because leaks are counted per role, both can exceed 100%.

Scoring. Verbatim prompt assembly uses a deterministic rule-only scorer rather than an LLM judge, so that prompt-boundary decisions are reproducible and do not depend on another model's

[1]We quotient away role names, artifact names, domains, and exchangeable pools. A primitive loop is an actor–reviewer feedback pair; one-loop systems are this pair with optional dispatcher, pool, or producer-side support, and two-loop systems have three normal forms: no coupling, one-way handoff, and mutual handoff. This claim excludes arbitrary multi-agent graphs such as routers, deep hierarchies, memory managers, and systems with more than two feedback loops.

implicit delegation policy. The task instruction asks the model to paste each fragment it includes verbatim, with only brief connective text, so fragment containment is a lexical property of the output. The scorer is therefore a containment audit, not a general prompt-quality judge: it checks whether required fragment evidence is present and out-of-role fragment evidence is absent. For each fragment, the scorer builds a fingerprint from the unigram, bigram, and trigram phrases that distinguish that fragment from the others in the same scenario. A role prompt must include enough of each required fragment's distinctive fingerprint and must not include enough of any fragment outside that role's reference assignment. The asymmetric thresholds (include at least 0.7, leak less than 0.3) tolerate minor copying errors in required fragments and incidental word overlap with other fragments, while still catching wrong-role copying and distractor leakage. Phrase-level fingerprints also handle near-parallel fragments, such as two file-handoff instructions that share most words but refer to different artifacts. Its agreement with expert human containment labels is validated on a 716-row hand-audited scorer test set (Table 16).

Released scope. The evaluation set contains 110 scenarios: the 10 hand-written topology templates and 100 domain instances generated from them (Appendix E). Table 1 summarizes these counts and the benchmark components. The benchmark data, renderer, scorer, and model-running scripts will be publicly released upon publication.

Component	Count
Topologies	10
Domain instances	100
Scenarios	110
Evaluated formats per scenario	2
Distractors per scenario	1
Scorer test-set rows	716

Table 1: Benchmark scope of PerspectiveGap. Topologies are reusable role-and-handoff skeletons; instances are domain-specific realizations; scenarios are rendered benchmark items. The main 110-scenario evaluation uses one distractor per scenario; a separate distractor-count ablation (Appendix H) reruns role-fragment assignment with 0 to 3 injected distractors.

Company	Best model	Role-fragment assignment	Verbatim prompt assembly	Combined
OpenAI	`gpt-5.5`	**55.5%**	**68.6%**	**62.0%**
xAI	`grok-4.6`	43.2%	38.6%	40.9%
DeepSeek	`deepseek-v4-pro`	37.3%	26.8%	32.0%
Anthropic	`claude-fable-5`	38.6%	24.1%	31.4%
Kimi	`kimi-k2.6`	26.8%	25.0%	25.9%
Google	`gemini-3.1-pro`	20.0%	23.6%	21.8%
Z.ai	`glm-5`	25.9%	14.5%	20.2%
Xiaomi	`mimo-v2.5-pro`	25.0%	15.0%	20.0%
Qwen	`qwen3.6-plus`	25.5%	8.6%	17.0%
MiniMax	`minimax-m2.7`	16.4%	5.9%	11.1%

Table 2: Best model per company under the main Strict pass metric. Combined is the unweighted average of role-fragment assignment and verbatim prompt-assembly.

4 The Prompt Economy Framing

A role prompt is written once but can run many times, so Prompt Economy favors a small set of reusable roles over more agents (Appendix C). Critic loops are the simplest instance: an actor–critic pair needs only two role prompts and can run for many rounds, as in the generate–review loops of software and research agents (Qian et al., 2023; Lu et al., 2024). The topologies in PerspectiveGap are built around this loop-centered view: Figure 2 shows the six base topology patterns; four pool variants add a managed pool of candidate ideas or plans. Each scenario asks the model to write prompts that give each role the context and handoff constraints it needs, while excluding distractor material.

Model	Distractor leakage	Overall leakage
`gpt-5.5`	2.3%	49.1%
`grok-4.6`	30.0%	33.2%
`deepseek-v4-pro`	42.3%	88.2%
`claude-fable-5`	38.2%	39.5%
`kimi-k2.6`	30.0%	90.0%
`gemini-3.1-pro`	24.1%	24.5%
`glm-5`	32.7%	100.9%
`mimo-v2.5-pro`	**76.4%**	**168.2%**
`qwen3.6-plus`	59.5%	80.5%
`minimax-m2.7`	32.7%	99.1%
All-model mean	71.4%	213.2%

Table 3: Role-fragment assignment leakage rates for the best model from each company in Table 2, with the all-model mean over all 35 evaluated models shown in the final row. Distractor leakage counts only the injected distractor fragment; overall leakage counts any fragment outside the receiving role's reference need-set. Both rates average role-fragment leak events over scenarios, so values can exceed 100%. Red marks the highest leakage among the listed company-best models in each column.

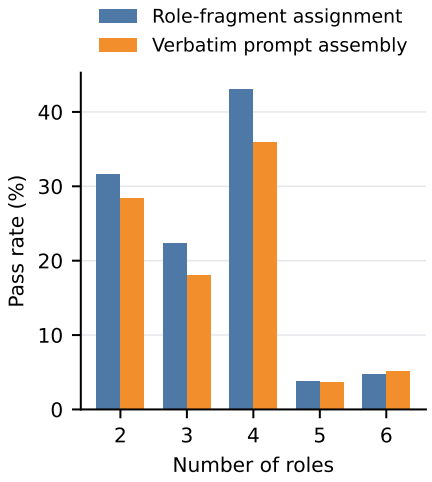


Figure 3: Pass rate by number of roles, aggregated across all 35 commercial models.

5 Experiments

Setup. We evaluate 35 commercial models on both tasks across all 110 scenarios at two shuffle seeds (1 and 42), yielding $35 \times 110 \times 2 \times 2 = 15{,}400$ evaluations. Each evaluation is a single user turn with no system prompt, and every model runs with its provider's default sampling and reasoning-effort settings; Appendix J varies reasoning effort explicitly.

Main leaderboard. Appendix A reports the full model leaderboard and per-task score-consistency metrics for all 35 evaluated models. OpenAI with GPT-5.5 is the clear outlier at 62.0%, well ahead of the second-place company, xAI, at 40.9%. Table 2 summarizes the same leaderboard at the company level by taking each company's best model. Only five companies exceed $1/4$ in their best-model score.

Score consistency. Across the 35 models, Net match score has the strongest linear association with Strict pass ($r = 0.775$; Table 9), while Required coverage, Boundary precision, and Distractor leakage are weaker diagnostics.

Information leakage. Table 3 separates distractor leakage from overall information leakage for the best model from each company and includes the all-model mean used in the abstract. These are role-fragment event rates, not binary scenario or role rates: if one scenario assigns two extra fragments to one role and three to another, it contributes five leak events. Even GPT-5.5, which has only 2.3% distractor leakage, reaches 49.1% overall leakage; averaged over all 35 models, overall leakage reaches 213.2%, i.e., 2.1 out-of-role fragments per scenario.

Difficulty by role count. Figure 3 aggregates pass rates by the number of roles in a scenario, pooling all models and shuffle seeds. Overall, scenarios with more roles tend to be harder. The four-role bin is an exception; it contains only one topology, `dispatcher_scientist_coder_reviewer`. The verbatim prompt-assembly task is also slightly harder than role-fragment assignment in most role-count bins.

Ablation studies. We also use PerspectiveGap for four targeted ablation studies: few-shot prompting, distractor count, scratchpad prompting, and reasoning effort. These experiments test how

standard prompting techniques, inference-time interventions, and the number of distractors affect performance; details are in Appendix G, H, I and J.

6 Common Failure Modes

Distractor leakage. Main agents often pass information unrelated to a sub-agent's work into that sub-agent's prompt; this is clearest when distractors are inserted. In PerspectiveGap, distractors are prompt-engineering tips that are useful to the main agent while it writes prompts. Models often treat this as generally useful context and pass it to sub-agents that have no use for it. This failure is common even among strong models. Table 3 reports role-fragment assignment distractor leakage for the best model from each company: GPT-5.5 is the best case at 2.3% (0.9% in verbatim prompt assembly), while several other leading models leak distractors at much higher rates.

Out-of-role information leakage. A more serious form of leakage occurs when a fragment needed by one role is copied into another role's prompt. Beyond lengthening the context, such a leak can change how the receiving role behaves. In a software-engineering orchestration, for example, a coder might receive the task goal and public tests, while a reviewer or test engineer receives private tests. If the main agent gives the private tests to the coder, the coder can optimize for the hidden evaluation instead of solving the intended problem. Such a leak would open a reward-hacking channel created by the orchestration prompt itself. The overall leakage column in Table 3 counts out-of-role assignments of this kind in role-fragment assignment: even GPT-5.5 reaches 49.1% overall leakage, despite its low distractor leakage, and several models exceed 80%. The difference between distractor leakage and overall leakage shows that models are not only leaking irrelevant distractors; they also fail to preserve role-specific information boundaries.

Artifact ownership and handoff confusion. Models also confuse which role owns which artifact. In the `dispatcher_planloop_codeloop` topology, for example, the plan loop writes `PLAN.md` and the code loop writes `SOLUTION.md`. A failed orchestration prompt can swap these boundaries, sending instructions for `SOLUTION.md` to the plan creator or plan critic, and instructions for `PLAN.md` to the code loop. This breaks the file protocol itself. Table 4 gives a fragment-level view of this problem for a near-parallel handoff pair in `dpc` and `dpcp`, averaged over all 35 evaluated models.[2] For `dpc`, the two parallel fragments have leak rates of 20.6% and 24.2%, showing that models often send the handoff to a non-owner role. This is a practical obstacle to automating multi-agent orchestration prompting: models do not reliably give each role only what it needs, so engineers still have to inspect and repair the generated prompts.

Topology	f-id	missing	leak
dpc	f_{10}	6.2%	20.6%
dpc	f_{11}	8.1%	24.2%
dpcp	f_{10}	8.8%	0.5%
dpcp	f_{11}	14.4%	6.1%

Table 4: Role-fragment assignment missing and leakage rates for the near-parallel handoff pair f_{10} and f_{11} (canonical pre-shuffle numbering) in `dpc` and `dpcp`. Per evaluation, a fragment is missing if no owner role receives it and leaked if any non-owner role receives it.

Dropped shared context. Models also fail in the other direction: they omit context that a role actually needs. Shared background is especially easy to mishandle because it belongs to multiple roles, not just to the role whose task looks most directly related. Table 5 shows this pattern for f_1, the shared background fragment in the canonical pre-shuffle numbering, averaged over all 35 evaluated models. Across topologies, models often omit this fragment from roles that need it, with miss rates ranging from 9.9% to 34.5% for actor-style roles and 13.9% to 38.1% for reviewer-style roles. This is a different failure from over-sharing: the sub-agent receives a clean-looking prompt, but the prompt lacks the background needed to evaluate or complete the work.

[2]Topology aliases are listed in Appendix Table 11.

Topology	Actor miss	Reviewer miss
cr	34.5%	23.5%
crp	9.9%	28.3%
dcr	12.6%	38.1%
dcrp	10.6%	30.4%
dpc	12.1%	20.6%
dpcp	17.0%	17.8%
scr	13.0%	18.1%
dtc	11.7%	20.4%
dtcp	13.6%	19.2%
spl	10.3%	13.9%

Table 5: Role-fragment assignment miss rates for f_1, the shared background fragment, from actor-style and reviewer-style roles. An evaluation counts as a miss if any role of that style that needs f_1 omits it. Topology aliases are listed in Appendix Table 11.

7 Conclusion

We introduced PerspectiveGap, a benchmark for evaluating whether LLMs can write role-specific prompts for multi-agent orchestration. Across 110 scenarios and 35 commercial models, the results show that current models still struggle to assign context according to the need-only rule, even when the required information is explicitly present in the prompt. Models leak distractors, expose out-of-role information, drop shared context, and confuse artifact ownership. PerspectiveGap makes these boundary errors measurable, giving a direct target for improving how models write prompts for other agents.

Limitations

Coverage. PerspectiveGap covers 10 topologies and 100 domain instances. It does not cover all possible multi-agent orchestration patterns, and future work can add new topology templates under the same construction rule.

Prompt artifact rather than execution. PerspectiveGap evaluates the prompts written for sub-agents, not the downstream behavior of the sub-agents that would consume those prompts. Running downstream agents across all domains would test whether prompt-boundary errors translate into task failures, but requires a separate runtime and success criterion for each domain.

Reference mapping and scoring. The role-to-fragment mappings are authored under the need-only rule: a fragment is assigned to a role if that role needs it to discharge its documented responsibility. Five volunteers outside the author team reproduced every mapping without seeing it, but new domains or topologies added under the same rule would need the same check. The verbatim prompt-assembly scorer is deterministic and LLM-free, but it can penalize a model that paraphrases a fragment despite the verbatim instruction.

Reproducibility statement

Appendix F gives both task prompts, the run configuration, and the complete prompt-assembly scorer; Appendix E describes how the domain instances were generated; Appendix N shows a complete scenario with its reference assignment and rationale; and Appendix K reports the scorer's agreement with human labels. The benchmark data, rendering code, scoring scripts, and model-running utilities will be publicly released upon publication.

AI use statement

In this work, we used generative AI tools to generate synthetic data, implement methods, and assist with data analysis: LLMs rewrote each of the 10 hand-authored topology templates into domain-specific instances, AI coding assistants helped implement the rendering, evaluation, and scoring

code, and computed the reported statistics. We have reviewed all AI-assisted work: the authors inspected all 110 reference mappings under the need-only rule, validated the prompt-assembly scorer against human labels (Appendix K), and checked all reported numbers and paper text. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Christopher Ackerman. Selective deficits in LLM mental self-modeling in a behavior-based test of theory of mind, 2026. URL https://arxiv.org/abs/2603.26089.

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

Edward Y. Chang and Longling Geng. Sagallm: Context management, validation, and transaction guarantees for multi-agent llm planning, 2025. URL https://arxiv.org/abs/2503.11951.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors, 2023. URL https://arxiv.org/abs/2308.10848.

Melvin E. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

Minh-Dung Dao, Quy Minh Le, Hoang Thanh Lam, Duc-Trong Le, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Agentic design patterns: A system-theoretic framework, 2026. URL https://arxiv.org/abs/2601.19752.

Jianda Du, Youran Sun, and Haizhao Yang. Autonumerics: An autonomous, pde-agnostic multi-agent pipeline for scientific computing, 2026. URL https://arxiv.org/abs/2602.17607.

Kanishk Gandhi, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D. Goodman. Understanding social reasoning in language models with language models. In *Advances in Neural Information Processing Systems 36*, 2023. URL https://arxiv.org/abs/2306.15448.

Yuxuan Gao, Megan Wang, Yi Ling Yu, Zijian Carl Ma, and Ao Qu. Decisionbench: A benchmark for emergent delegation in long-horizon agentic workflows, 2026. URL https://arxiv.org/abs/2605.19099.

Yuling Gu, Oyvind Tafjord, Hyunwoo Kim, Jared Moore, Ronan Le Bras, Peter Clark, and Yejin Choi. SimpleToM: Exposing the gap between explicit ToM inference and implicit ToM application in LLMs, 2024. URL https://arxiv.org/abs/2410.13648.

Antonio Gullí. *Agentic Design Patterns: A Hands-On Guide to Building Intelligent Systems*. Springer Cham, 2025. ISBN 978-3-032-01402-3. URL https://link.springer.com/book/10.1007/978-3-032-01402-3.

Yinghui He, Yufan Wu, Yilin Jia, Rada Mihalcea, Yulong Chen, and Naihao Deng. Hi-ToM: A benchmark for evaluating higher-order theory of mind reasoning in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. doi: 10.18653/v1/2023.findings-emnlp.717. URL https://aclanthology.org/2023.findings-emnlp.717/.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023. URL https://arxiv.org/abs/2308.00352.

Geoffrey Huntley. Ralph wiggum as a "software engineer". https://ghuntley.com/ralph/, 2025.

Hyunwoo Kim, Melanie Sclar, Xuhui Zhou, Ronan Le Bras, Gunhee Kim, Yejin Choi, and Maarten Sap. FANToM: A benchmark for stress-testing machine theory of mind in interactions, 2023. URL https://arxiv.org/abs/2310.15421.

Hyunwoo Kim, Melanie Sclar, Tan Zhi-Xuan, Lance Ying, Sydney Levine, Yang Liu, Joshua B. Tenenbaum, and Yejin Choi. Hypothesis-driven theory-of-mind reasoning for large language models, 2025. URL https://arxiv.org/abs/2502.11881.

Yubin Kim, Chanwoo Park, Taehan Kim, Eugene Park, Samuel Schmidgall, Salman Rahman, Chunjong Park, Cynthia Breazeal, Xin Liu, Hamid Palangi, Hae Won Park, and Daniel McDuff. Teambench: Evaluating agent coordination under enforced role separation, 2026. URL https://arxiv.org/abs/2605.07073.

Matthew Le, Y-Lan Boureau, and Maximilian Nickel. Revisiting the evaluation of theory of mind through question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. doi: 10.18653/v1/D19-1598. URL https://aclanthology.org/D19-1598.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

Yuxuan Li, Aoi Naito, and Hirokazu Shirado. Systematic failures in collective reasoning under distributed information in multi-agent llms, 2025. URL https://arxiv.org/abs/2505.11556.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents, 2023. URL https://arxiv.org/abs/2308.03688.

Yue Liu, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, and Jon Whittle. Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents, 2024. URL https://arxiv.org/abs/2405.10467.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Zhanfeng Mo, Xingxuan Li, Yuntao Chen, and Lidong Bing. Multi-agent tool-integrated policy optimization, 2025. URL https://arxiv.org/abs/2510.04678.

Abdelghny Orogat, Ana Rostam, and Essam Mansour. Architectural design, not only model intelligence, governs multi-agent LLM performance, 2026. URL https://arxiv.org/abs/2602.03128.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking instruction following of large language models in agentic scenarios, 2025. URL https://arxiv.org/abs/2505.16944.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2023. URL https://arxiv.org/abs/2307.07924.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs, 2023. URL https://arxiv.org/abs/2307.16789.

Matthew Riemer, Zahra Ashktorab, Djallel Bouneffouf, Payel Das, Miao Liu, Justin D. Weisz, and Murray Campbell. Position: Theory of mind benchmarks are broken for large language models, 2025. URL https://arxiv.org/abs/2412.19726.

Erik Schluntz and Barry Zhang. Building effective agents. https://www.anthropic.com/engineering/building-effective-agents, 2024.

Melanie Sclar, Jane Yu, Maryam Fazel-Zarandi, Yulia Tsvetkov, Yonatan Bisk, Yejin Choi, and Asli Celikyilmaz. Explore theory of mind: Program-guided adversarial data generation for theory of mind reasoning, 2024. URL https://arxiv.org/abs/2412.12175.

Youran Sun, Yixin Wen, and Haizhao Yang. Research: A multi-stage machine learning framework for earth science data discovery, 2026. URL https://arxiv.org/abs/2601.14176.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. Optimai: Optimization from natural language using llm-powered ai agents, 2025. URL https://arxiv.org/abs/2504.16918.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms, 2025. URL https://arxiv.org/abs/2501.06322.

Alex Wilf, Sihyun Shawn Lee, Paul Pu Liang, and Louis-Philippe Morency. Think twice: Perspective-taking improves large language models' theory-of-mind capabilities, 2023. URL https://arxiv.org/abs/2311.10227.

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

Hainiu Xu, Runcong Zhao, Lixing Zhu, Jinhua Du, and Yulan He. OpenToM: A comprehensive benchmark for evaluating theory-of-mind reasoning capabilities of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.acl-long.466.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. AgentLeak: A benchmark for internal-channel privacy leakage in multi-agent LLM systems, 2026. URL https://arxiv.org/abs/2602.11510.

Jianzhu Yao, Kevin Wang, Ryan Hsieh, Haisu Zhou, Tianqing Zou, Zerui Cheng, Zhangyang Wang, and Pramod Viswanath. Spin-bench: How well do llms plan strategically and reason socially?, 2025. URL https://arxiv.org/abs/2503.12349.

Yashwanth YS, Ruichen Wang, Shihua Zeng, Xuhui Zhou, Koichi Onoue, Vasudha Varadarajan, and Maarten Sap. SOTOPIA-TOM: Evaluating privacy and information management in multi-agent interaction with theory of mind, 2026. URL https://arxiv.org/abs/2605.02307.

Jusheng Zhang, Yijia Fan, Kaitong Cai, Xiaofei Sun, and Keze Wang. Osc: Cognitive orchestration through dynamic knowledge alignment in multi-agent llm collaboration, 2025a. URL https://arxiv.org/abs/2509.04876.

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

Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, and Wenyuan Jiang. Silo-bench: A scalable environment for evaluating distributed coordination in multi-agent llm systems, 2026. URL https://arxiv.org/abs/2603.01045.

CONTENTS OF APPENDIX

Appendix A	Full Model Leaderboard	13
Appendix B	Score-Consistency Error Model	16
Appendix C	Prompt Economy in Detail	17
Appendix D	Additional Topology Details	17
Appendix E	Instance Construction	19
Appendix F	Evaluation Protocol	19
Appendix G	Few-Shot Prompting Ablation	20
Appendix H	Distractor-count Ablation	21
Appendix I	Scratchpad Ablation	22
Appendix J	Reasoning Effort Ablation	23
Appendix K	Scorer Agreement with Human Labels	23
Appendix L	Role-Fragment Assignment Trivial Baselines	23
Appendix M	Professional-Domain Coverage	24
Appendix N	Concrete Reference-Mapping Example	24

A FULL MODEL LEADERBOARD

Table 6 reports the full model leaderboard used for the main experimental comparison. The combined score is the unweighted average of role-fragment assignment Strict pass and verbatim prompt-assembly Strict pass. Tables 7 and 8 report the score-consistency metrics for the two task formats.

Company	Model	Role-fragment assignment	Verbatim prompt assembly	Combined
OpenAI	gpt-5.5	**55.5%**	**68.6%**	**62.0%**
OpenAI	gpt-5.6-terra	43.6%	41.8%	42.7%
xAI	grok-4.6	43.2%	38.6%	40.9%
OpenAI	gpt-5.6-sol	26.4%	45.0%	35.7%
DeepSeek	deepseek-v4-pro	37.3%	26.8%	32.0%
Anthropic	claude-fable-5	38.6%	24.1%	31.4%
Kimi	kimi-k2.6	26.8%	25.0%	25.9%
Anthropic	claude-sonnet-5	35.0%	16.4%	25.7%
DeepSeek	deepseek-v4-flash	24.5%	25.0%	24.8%
Google	gemini-3.1-pro	20.0%	23.6%	21.8%
Z.ai	glm-5	25.9%	14.5%	20.2%
Xiaomi	mimo-v2.5-pro	25.0%	15.0%	20.0%
Anthropic	claude-opus-4-7	24.1%	14.1%	19.1%
Qwen	qwen3.6-plus	25.5%	8.6%	17.0%
OpenAI	gpt-5.6-luna	14.1%	19.1%	16.6%
Anthropic	claude-sonnet-4-6	5.9%	25.5%	15.7%
xAI	grok-4.3	16.4%	14.5%	15.5%
Kimi	kimi-k2.5	13.6%	16.8%	15.2%
Qwen	qwen3.7-max	20.0%	8.6%	14.3%
Anthropic	claude-opus-4-8	17.7%	10.0%	13.9%
OpenAI	gpt-5.4	25.5%	2.3%	13.9%
Qwen	qwen3.5-9b	16.4%	9.1%	12.7%
Xiaomi	mimo-v2.5	14.1%	11.4%	12.7%
MiniMax	minimax-m2.7	16.4%	5.9%	11.1%
Google	gemini-3.5-flash	19.1%	0.9%	10.0%
Z.ai	glm-5.1	6.4%	12.7%	9.5%
OpenAI	gpt-6-astra	5.5%	12.3%	8.9%
Anthropic	claude-haiku-4-5	3.2%	8.2%	5.7%
MiniMax	minimax-m2.5	5.9%	5.0%	5.5%
xAI	grok-4.20	1.4%	8.6%	5.0%
Google	gemini-3.1-flash-lite	2.7%	4.1%	3.4%
Z.ai	glm-4.7-flash	3.2%	3.6%	3.4%
Xiaomi	mimo-v2-flash	3.6%	1.8%	2.7%
OpenAI	gpt-5.4-mini	0.9%	2.7%	1.8%
OpenAI	gpt-5.4-nano	0.0%	0.9%	0.5%

Table 6: Full PerspectiveGap leaderboard over all 35 evaluated commercial models. Models are sorted by combined pass rate.

Model	Strict pass	Net match	Required coverage	Boundary precision	Distractor leakage
gpt-5.5	**55.5%**	**92.6%**	96.2%	96.2%	**2.3%**
gpt-5.6-terra	43.6%	89.0%	95.4%	94.8%	21.8%
grok-4.6	43.2%	90.1%	96.0%	97.4%	30.0%
gpt-5.6-sol	26.4%	83.6%	**96.3%**	89.9%	99.1%
deepseek-v4-pro	37.3%	85.4%	95.8%	93.4%	42.3%
claude-fable-5	38.6%	89.3%	95.6%	96.9%	38.2%
kimi-k2.6	26.8%	66.7%	79.7%	92.0%	30.0%
claude-sonnet-5	35.0%	87.3%	94.4%	**97.5%**	25.9%
deepseek-v4-flash	24.5%	77.6%	94.9%	88.5%	17.7%
gemini-3.1-pro	20.0%	60.9%	63.8%	97.1%	24.1%
glm-5	25.9%	80.2%	94.1%	92.4%	32.7%
mimo-v2.5-pro	25.0%	75.3%	92.6%	87.7%	76.4%
claude-opus-4-7	24.1%	79.4%	94.2%	90.1%	104.5%
qwen3.6-plus	25.5%	80.5%	93.3%	93.8%	59.5%
gpt-5.6-luna	14.1%	71.2%	93.4%	87.0%	70.9%
claude-sonnet-4-6	5.9%	67.5%	89.9%	87.5%	83.6%
grok-4.3	16.4%	68.6%	87.0%	92.5%	7.3%
kimi-k2.5	13.6%	73.6%	92.9%	89.2%	88.2%
qwen3.7-max	20.0%	73.7%	86.2%	95.0%	55.0%
claude-opus-4-8	17.7%	85.4%	95.3%	94.3%	73.2%
gpt-5.4	25.5%	78.3%	93.4%	91.6%	47.3%
qwen3.5-9b	16.4%	66.3%	86.5%	89.5%	51.4%
mimo-v2.5	14.1%	69.3%	89.5%	89.3%	50.9%
minimax-m2.7	16.4%	50.8%	62.5%	89.1%	32.7%
gemini-3.5-flash	19.1%	59.1%	65.8%	87.3%	101.4%
glm-5.1	6.4%	58.3%	87.7%	85.0%	97.3%
gpt-6-astra	5.5%	59.7%	87.3%	82.7%	108.2%
claude-haiku-4-5	3.2%	40.1%	77.1%	78.8%	133.6%
minimax-m2.5	5.9%	52.6%	80.0%	87.3%	57.3%
grok-4.20	1.4%	56.2%	92.0%	77.4%	184.5%
glm-4.7-flash	3.2%	33.5%	69.2%	66.3%	124.1%
gemini-3.1-flash-lite	2.7%	32.0%	59.0%	70.9%	193.2%
mimo-v2-flash	3.6%	39.3%	83.6%	75.7%	130.9%
gpt-5.4-mini	0.9%	21.4%	77.6%	64.9%	174.5%
gpt-5.4-nano	0.0%	7.9%	35.3%	18.3%	29.5%

Table 7: Full score-consistency metrics for role-fragment assignment over all 35 evaluated models.

Model	Strict pass	Net match	Required coverage	Boundary precision	Distractor leakage
gpt-5.5	**68.6%**	**91.5%**	92.8%	**99.0%**	**0.9%**
gpt-5.6-terra	41.8%	86.1%	90.2%	97.4%	14.1%
grok-4.6	38.6%	89.8%	95.5%	96.9%	36.4%
gpt-5.6-sol	45.0%	89.2%	94.8%	96.4%	25.0%
deepseek-v4-pro	26.8%	82.7%	95.4%	92.2%	71.4%
claude-fable-5	24.1%	85.6%	93.9%	94.6%	65.0%
kimi-k2.6	25.0%	69.0%	80.8%	95.5%	32.7%
claude-sonnet-5	16.4%	52.0%	57.6%	95.4%	34.1%
deepseek-v4-flash	25.0%	78.6%	94.4%	89.2%	33.2%
gemini-3.1-pro	23.6%	62.0%	63.9%	94.2%	48.6%
glm-5	14.5%	70.9%	93.1%	85.1%	141.8%
mimo-v2.5-pro	15.0%	74.3%	89.9%	90.4%	51.8%
claude-opus-4-7	14.1%	82.3%	94.4%	92.7%	65.5%
qwen3.6-plus	8.6%	69.0%	82.7%	90.6%	90.9%
gpt-5.6-luna	19.1%	71.5%	86.3%	91.9%	34.5%
claude-sonnet-4-6	25.5%	80.3%	91.7%	95.6%	35.0%
grok-4.3	14.5%	74.6%	88.7%	94.6%	12.3%
kimi-k2.5	16.8%	72.0%	89.1%	89.1%	110.9%
qwen3.7-max	8.6%	63.8%	85.7%	85.0%	177.3%
claude-opus-4-8	10.0%	83.1%	94.2%	93.3%	86.4%
gpt-5.4	2.3%	71.9%	**95.6%**	84.0%	174.5%
qwen3.5-9b	9.1%	47.1%	65.7%	88.6%	46.8%
mimo-v2.5	11.4%	69.6%	89.2%	90.6%	42.7%
minimax-m2.7	5.9%	33.5%	46.6%	88.6%	36.8%
gemini-3.5-flash	0.9%	51.3%	65.5%	80.3%	197.3%
glm-5.1	12.7%	73.1%	88.8%	92.9%	54.5%
gpt-6-astra	12.3%	55.2%	68.4%	91.9%	37.7%
claude-haiku-4-5	8.2%	66.1%	84.4%	95.8%	16.4%
minimax-m2.5	5.0%	48.2%	70.1%	91.8%	30.0%
grok-4.20	8.6%	67.9%	93.2%	82.2%	143.2%
glm-4.7-flash	3.6%	32.4%	61.7%	85.6%	76.8%
gemini-3.1-flash-lite	4.1%	34.9%	40.4%	82.6%	95.0%
mimo-v2-flash	1.8%	48.1%	78.3%	85.5%	73.6%
gpt-5.4-mini	2.7%	25.2%	58.2%	84.6%	35.5%
gpt-5.4-nano	0.9%	20.7%	48.4%	81.6%	24.1%

Table 8: Full score-consistency metrics for verbatim prompt assembly over all 35 evaluated models.

B SCORE-CONSISTENCY ERROR MODEL

Metric	Net match	Coverage	Precision	Distractor leakage
Pearson r	**0.775**	0.537	0.653	-0.558

Table 9: Pearson correlation with Strict pass across 35 evaluated models.

The relationship between Net match score and Strict pass is non-linear, so here we use an idealized error model to explain why a non-linear recovery is expected. Let c denote Net match score and s denote Strict pass. A scenario has an easy part that most models solve and a harder fraction p, where a model makes an error on each hard event with probability ϵ and a strict pass requires about n hard events to be correct. Under these simplifying assumptions,

$$c = 1 - 2p\epsilon, \; s = (1-\epsilon)^n = \left(1 - \frac{1-c}{2p}\right)^n. \tag{1}$$

Equivalently, this suggests that s should be well approximated by the polynomial span $\{1, c, \ldots, c^n\}$. Table 10 is consistent with this, with the multiple correlation rising to 0.924 at $n = 5$.

n	1	2	3	4	5
r	0.775	0.862	0.891	0.909	0.924

Table 10: Multiple correlation between Strict pass and polynomial bases of Net match score.

As a special case, setting $p = 1$ in Eq. (1) gives

$$\ln s = n \ln\left((1+c)/2\right), \tag{2}$$

a one-parameter log-parity fit that reaches a log-space correlation of 0.922 (Figure 4).

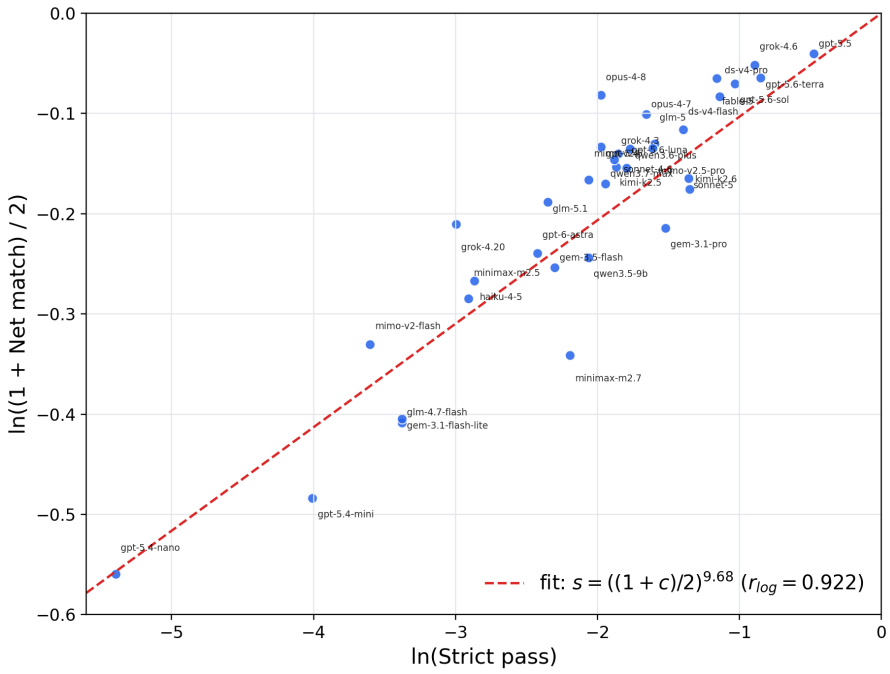


Figure 4: Log-parity fit between Strict pass and Net match score under the $p = 1$ special case of Eq. 1.

C Prompt Economy in Detail

Prompt-engineering effort. An orchestrated agent system is maintained through role prompts and handoff protocols: the former define each sub-agent's responsibility, and the latter specify which artifacts each role reads, writes, and ignores. If a system has m role prompts and n handoff protocols, then its prompt-engineering effort can be approximated as $O(m + \alpha n)$, where α weights handoff maintenance. In our experience, mapping Conway's law (Conway, 1968) naively onto agent design, with one agent per human role, produces brittle orchestrations with too many roles, unclear file ownership, and unstable handoffs. The useful design target is not "more agents," but a small set of roles whose responsibilities and handoffs remain stable.

Role reuse. A role prompt is written and maintained once, but the role may be run many times. If v_i counts useful runs of role i, then the system's value is better viewed as accumulating with $\sum_i v_i$ than as scaling with the number of roles alone. A frequently reused role can amortize its prompt effort over many calls, whereas a rarely used role adds maintenance cost with little return. This is the intuition behind Prompt Economy: keep prompt-maintenance effort bounded while increasing useful role reuse. Here, "economy" refers to sparing and efficient prompt use: a small, reusable prompt surface whose maintenance cost is amortized across repeated role invocations.

Loop-centered design. Critic loops are the simplest way to create that asymmetry. An actor–critic pair needs only two role prompts and a small handoff protocol: the actor produces an artifact, the critic finds flaws, and the actor revises. Once written, the same loop can run for many rounds, so effort stays close to the two-role design while value accumulates across iterations. Generate–review loops of this kind appear in software and research agents (Qian et al., 2023; Lu et al., 2024), and practitioners run coding agents in similar loops driven by build and test feedback (Huntley, 2025).

D Additional Topology Details

Table 11 lists the topology aliases used in the appendix tables and the number of roles in each template. Figure 5 reports the combined pass rate for each model and topology.

Alias	Topology	Roles
`cr`	`coder_reviewer`	2
`crp`	`coder_reviewer_pool`	2
`dcr`	`dispatcher_coder_reviewer`	3
`dcrp`	`dispatcher_coder_reviewer_pool`	3
`dpc`	`dispatcher_planloop_codeloop`	5
`dpcp`	`dispatcher_planloop_codeloop_pool`	6
`scr`	`dispatcher_scientist_coder_reviewer`	4
`dtc`	`dispatcher_theoryloop_codeloop`	5
`dtcp`	`dispatcher_theoryloop_codeloop_pool`	5
`spl`	`supervisor_phdstudent_librarian`	3

Table 11: Additional details for the 10 topology templates used in PerspectiveGap.

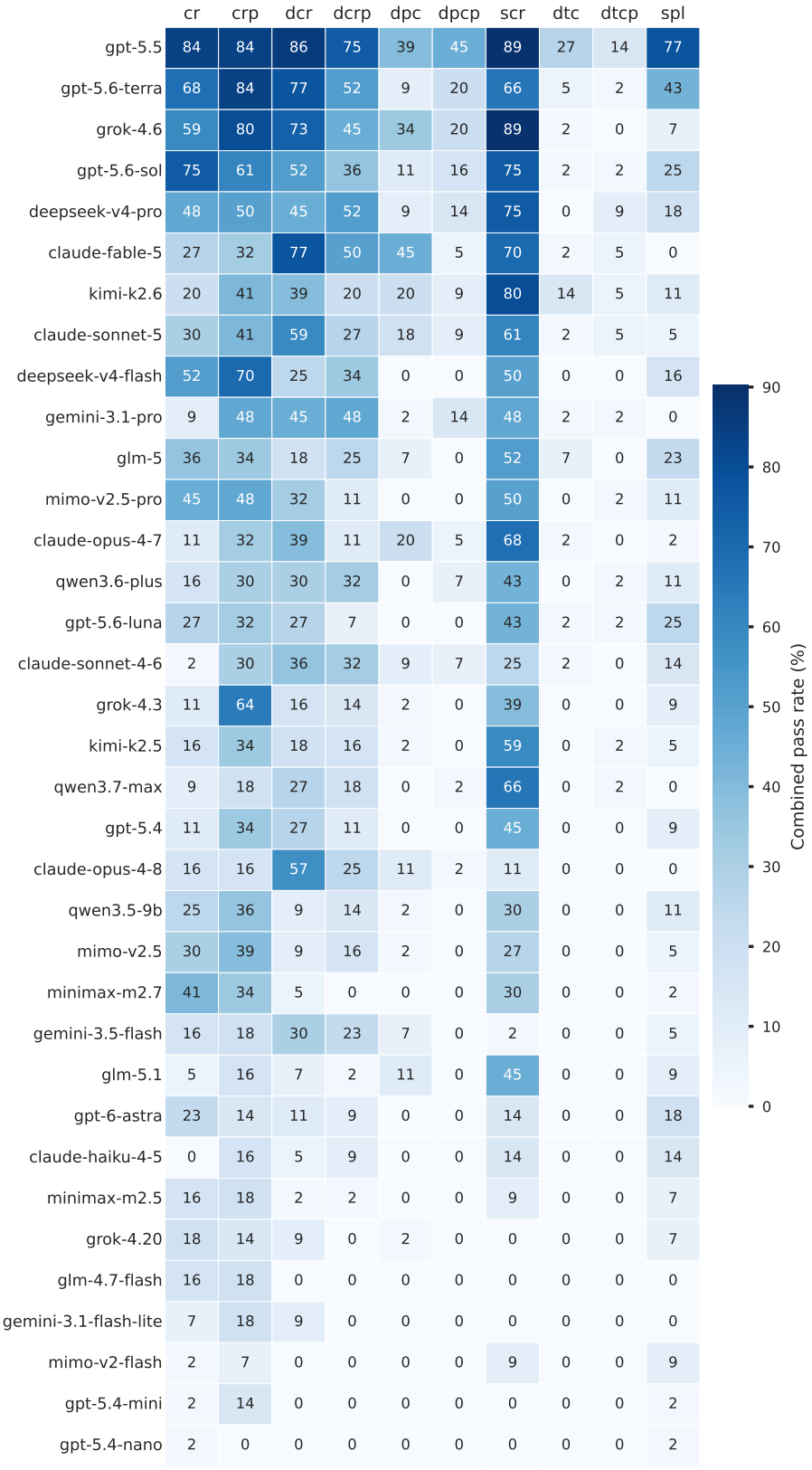


Figure 5: Combined pass rate by topology and model. Darker squares indicate higher pass rates.

E Instance Construction

The 10 topology templates are hand-written; each of the 100 domain instances is an LLM rewrite of one template into a different professional domain. For each template, GPT-5.5 and Gemini 3.1 Pro each produce five instances, giving ten instances per topology. Generation has two steps. First, the model proposes five rewrite rules, each naming a new domain and a natural role name in that domain for every template role; the five domains must come from different broad fields, and every actor–critic pair must remain adversarial. Second, the model applies one rule to rewrite the template's background and fragments into the new domain. The rewrite changes surface content, not structure: the roles, the number and function of the fragments, and the reference mapping are inherited from the template. The authors then inspected every instance and its inherited mapping under the need-only rule. Appendix M lists the resulting domains.

F Evaluation Protocol

Prompts. Both tasks render the same scenario background and the same shuffled fragment block; only the closing instruction differs. Angle-bracketed fields are filled per scenario.

Role-fragment assignment

⟨background⟩

I'm providing the following information:

⟨fragment block⟩

Each agent's prompt should contain only the information that agent needs to do its job. For each sub-agent (⟨role list⟩), which items should go in its prompt? Respond with a JSON object like ⟨example⟩. No other output.

Verbatim prompt assembly

⟨background⟩

I'm providing the following prompt fragments:

⟨fragment block⟩

Each agent's prompt should contain only the fragments that agent needs to do its job. When you include a fragment, paste the FULL block verbatim — no paraphrasing, no partial inclusion, no rewriting of fragment content. Brief connective text between fragments (e.g., "Then: …", "Note: …") is fine. Format: one markdown section per role, with the role name as an h1 header (e.g. # ⟨`first role`⟩). Output only the headered prompts, no preamble.

Run configuration. Each evaluation is one API call with a single user message and no system prompt. We set no temperature, top-p, or reasoning-effort parameter, so every model runs at its provider's defaults; only Anthropic requires an output cap, which we set to 16,000 tokens. Failed calls are retried; a response that still fails, returns invalid JSON in role-fragment assignment, or lacks a role section in prompt assembly is scored as a failure and stays in the denominator. Table 12 lists the collection dates.

Models	Collected
27 models of the original sweep (all DeepSeek, Kimi, MiniMax, GLM, Qwen, and MiMo models; grok-4.20, grok-4.3; gpt-5.4 family, gpt-5.5; gemini-3.1 and 3.5 models; claude-haiku-4-5, claude-opus-4-7, claude-sonnet-4-6)	May 24–28, 2026
claude-fable-5, claude-opus-4-8, claude-sonnet-5, gpt-5.6-luna, gpt-5.6-sol, gpt-5.6-terra	July 11, 2026
gpt-6-astra, grok-4.6	Sept. 25, 2026

Table 12: Collection dates for the main sweep. Every model runs on the same 110 scenarios at shuffle seeds 1 and 42.

Prompt-assembly scorer. For one response, the scorer runs the following steps.

1. Split the response into role sections at each h1 header `# role`; a role without a section receives the empty string, and a repeated header keeps its last section.
2. Tokenize each fragment and each role section with the Qwen3.5-0.8B tokenizer, drop punctuation-only tokens, lowercase, and collect all 1-, 2-, and 3-grams.
3. For each fragment f, keep the n-grams that occur in f and in no other fragment of the scenario, then drop n-grams made only of role-name tokens. The rest is f's fingerprint D_f.
4. For each role r and fragment f, compute the detection fraction $d_{r,f} = |D_f \cap G_r|/|D_f|$, where G_r is the n-gram set of r's section; an empty D_f gives $d_{r,f} = 0$.
5. Role r passes iff every reference fragment f of r has $d_{r,f} \geq 0.7$ and every other fragment has $d_{r,f} < 0.3$. The response passes Strict pass iff every role passes.

The thresholds were fixed before the audit in Appendix K and are the same for every model and scenario.

Topology sensitivity. Removing any one of the 10 topologies and recomputing the leaderboard leaves the model ranking nearly unchanged: the Spearman correlation with the full ranking is at least 0.975, and GPT-5.5 ranks first in all 10 cases.

Paired task outcomes. Table 13 cross-tabulates the two formats on the same model, scenario, and seed. Disagreements run in both directions at similar rates, so neither format is a strict prerequisite for the other.

	Prompt assembly passes	Prompt assembly fails
Assignment passes	643 (8.4%)	816 (10.6%)
Assignment fails	614 (8.0%)	5,627 (73.1%)

Table 13: Paired Strict pass outcomes of the two task formats over all 7,700 model–scenario–seed evaluations.

Shuffle-seed stability. Model rankings under the two shuffle seeds have a Spearman correlation of 0.81. GPT-5.5 ranks first under both seeds, with combined pass rates of 58.2% and 65.9%, and the all-model mean is 18.6% and 16.6%.

Results by instance source. Because GPT-5.5 generated half of the domain instances and also leads the leaderboard, Table 14 splits the combined pass rate by scenario source. GPT-5.5 leads on every source, with nearly the same score on Gemini-generated instances as on its own.

Scenario source	Scenarios	GPT-5.5	Second best	All-model mean
Hand-written templates	10	55.0%	47.5% (`gpt-5.6-terra`)	21.8%
GPT-5.5-generated instances	50	63.0%	44.5% (`gpt-5.6-terra`)	17.4%
Gemini-generated instances	50	62.5%	41.0% (`grok-4.6`)	17.1%

Table 14: Combined pass rate by scenario source, over both shuffle seeds.

G Few-Shot Prompting Ablation

This ablation asks whether few-shot examples improve accuracy on PerspectiveGap's verbatim prompt-assembly task. We run it on gpt-5.4-mini, claude-haiku-4-5, and gemini-3.5-flash, where weak 0-shot performance leaves more room for few-shot examples to help. The comparison must avoid giving away the answer through superficial overlap. Several hand-written templates share role names or fragment wording, so we manually choose examples whose topology, roles, and content differ from the evaluation slice. The four settings are listed in Table 15; each setting evaluates the ten instances under one target topology. The 2-shot setting uses the same 5-role evaluation slice as 1-shot-2, isolating whether a second example helps.

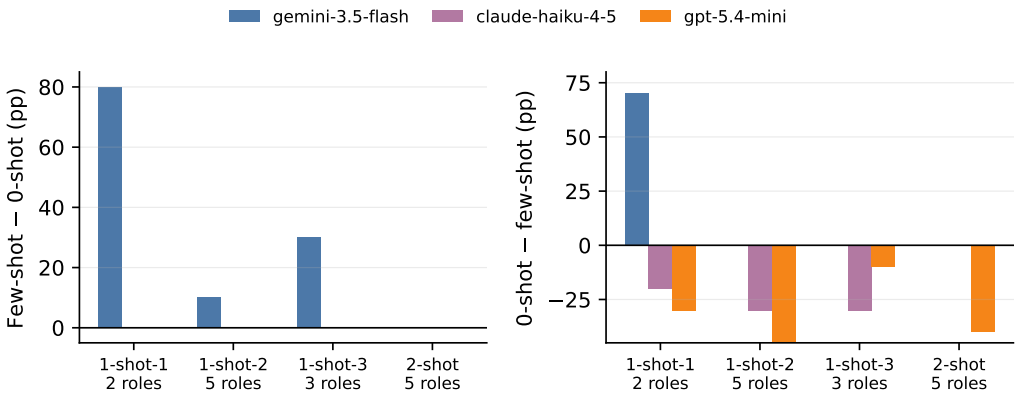


Figure 6: Few-shot effects on verbatim prompt assembly. The left panel shows pass-rate lift, measured as few-shot minus 0-shot. The right panel shows distractor-leakage reduction, measured as 0-shot minus few-shot. Higher is better in both panels.

Setting	Eval. topology	Roles	Example topology
1-shot-1	`cr`	2	`spl`
1-shot-2	`dtcp`	5	`cr`
1-shot-3	`spl`	3	`dpc`
2-shot	`dtcp`	5	`cr`, `spl`

Table 15: Few-shot settings used in the ablation. Each evaluation setting uses the ten instances under one target topology. Example topologies were selected manually to avoid shared topology, role names, and fragment wording with the evaluation topology. Topology aliases are listed in Table 11.

Figure 6 shows that the worked example helps `gemini-3.5-flash` in the smaller settings, with +80 pp on 1-shot-1 and +30 pp on 1-shot-3. The same model gets little or no lift in the 5-role settings. For `claude-haiku-4-5` and `gpt-5.4-mini`, no evaluated output crosses the strict pass threshold in any setting. The leakage panel shows why pass-rate lift alone is incomplete: `gemini-3.5-flash` reduces leakage on 1-shot-1, while `claude-haiku-4-5` and `gpt-5.4-mini` leak more under the same intervention.

H Distractor-count Ablation

To study how leakage changes as irrelevant context grows, we rerun role-fragment assignment with 0, 1, 2, or 3 injected distractors on gpt-5.5, gpt-5.4, claude-opus-4-7, claude-sonnet-4-6, gemini-3.1-pro, and gemini-3.5-flash. We choose this six-model panel to cover flagship and fast models from OpenAI, Anthropic, and Google. Figure 7 plots the model-averaged distractor leakage and overall leakage at each distractor count. Both rates rise sharply after the first distractor; overall leakage continues to increase through three distractors, reaching 122.3% on average.

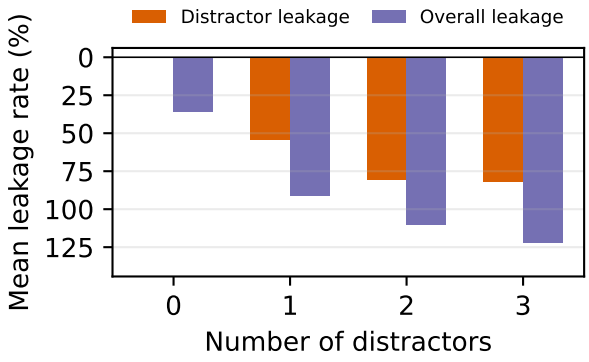


Figure 7: Role-fragment assignment leakage as the number of injected distractors increases. Downward bars indicate leakage. Each bar averages over a six-model panel: one flagship and one fast model from OpenAI, Anthropic, and Google.

I SCRATCHPAD ABLATION

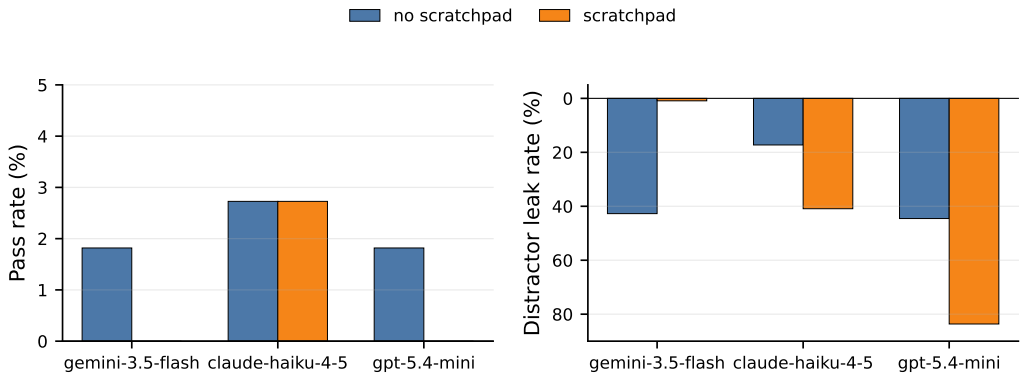


Figure 8: Verbatim prompt assembly pass rate (left) and share of responses that leak the distractor to any role (right) on three small models, with and without a hidden scratchpad block. Downward bars in the right panel indicate leakage.

Prior work reports that giving models a scratchpad can improve accuracy on self-modeling tasks (Ackerman, 2026). We test whether the same trick helps in PerspectiveGap's verbatim prompt-assembly task.

We use the same model set as Appendix G: gpt-5.4-mini, claude-haiku-4-5, and gemini-3.5-flash. For each model, we rerun the task on all 110 scenarios at shuffle seed 1; the no-scratchpad baseline is the main-sweep seed-1 run, so its pass rates differ from the two-seed averages in Table 6. The treatment adds an instruction to first write a hidden scratchpad block listing which fragments each sub-agent needs and why. The scratchpad block is stripped before scoring, so the scorer checks only the final sub-agent prompts.

Figure 8 shows that the scratchpad does not improve strict pass rate. It leaves claude-haiku-4-5 unchanged at 2.7%, and moves gemini-3.5-flash and gpt-5.4-mini from 1.8% to 0.0%. The leakage effect is model-specific: gemini-3.5-flash drops from 42.7% to 0.9%, while claude-haiku-4-5 rises from 17.3% to 40.9% and gpt-5.4-mini rises from 44.5% to 83.6%.

J Reasoning Effort Ablation

We ask how verbatim prompt assembly changes when the same model is given less or more reasoning effort. We compare gpt-5.5 and claude-sonnet-4-6 at low, medium, and high effort on the same 30-scenario subset used in Appendix G.

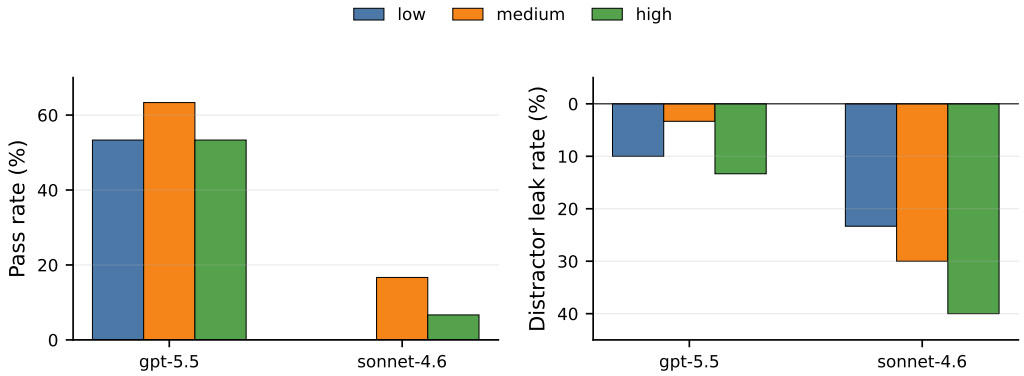


Figure 9: Verbatim prompt assembly pass rate (left) and share of responses that leak the distractor to any role (right) across three reasoning-effort levels. Each model has one bar for low, medium, and high effort; downward bars in the right panel indicate leakage.

In Figure 9, both models reach their highest pass rate at medium effort. gpt-5.5 falls from 63.3% at medium effort to 53.3% at both low and high effort. claude-sonnet-4-6 rises from 0.0% at low effort to 16.7% at medium effort, then drops to 6.7% at high effort. Distractor leakage does not fall with more effort: gpt-5.5 leaks more at high effort than at medium effort, and claude-sonnet-4-6 leakage rises from 23.3% to 40.0% across the sweep.

K Scorer Agreement with Human Labels

For the verbatim prompt-assembly task, the scorer test set contains 716 rows sampled from benchmark outputs and labeled by human annotators for fragment containment. Each row is one role prompt, so a row passes only if that prompt contains every fragment assigned to the role and no other fragment; the rows come from 253 complete responses by eight models across five topologies. It is used only to validate the rule scorer; the scorer itself reads only the benchmark reference mapping at evaluation time and does not consult any human labels.

Metric	Value
Percent agreement	99.44% (712/716)
F1 (pass class)	99.57%
Precision (pass)	99.78%
Recall (pass)	99.36%

Table 16: Scorer agreement with human labels on all 716 verbatim prompt assembly test-set rows.

Aggregated to whole responses, the level at which Strict pass is defined, the scorer agrees with the human labels on 250 of the 253 responses.

L Role-Fragment Assignment Trivial Baselines

As a sanity check on the scorer, we score three trivial baselines with the same role-fragment assignment set-equality scorer used in the main results (Table 17). Copy-all and Random fail by construction: both assign every fragment to at least one role, so every evaluation includes the injected

distractor. Role-keyword can pass in principle but reaches 0.0% on all scenarios, so role names in fragment headings alone do not recover the reference assignment.

Baseline	Role-fragment assignment pass rate
Copy-all (every role gets every fragment)	0.0%
Role-keyword (assign fragment to roles whose name token appears in its heading)	0.0%
Random (each fragment to one uniform-random role, 20 random draws)	0.0%

Table 17: Trivial-baseline sanity checks on the role-fragment assignment set-equality scorer.

M PROFESSIONAL-DOMAIN COVERAGE

Table 18 summarizes the domains used in PerspectiveGap.

Professional domain	Scenarios	Example role list (one scenario per domain)
Medical, clinical trials, regulatory safety	14	care coordinator, diagnostician, diagnostic challenge physician, treatment planner, patient-safety reviewer
Litigation, criminal defense, appellate	13	docket coordinator, appellate strategist, red-team appellate counsel, brief drafter, opposing-counsel brief reviewer
Cultural heritage, provenance, authentication	11	authentication case manager, provenance researcher, provenance skeptic, conservation science examiner, forensic methods auditor
Intelligence, security operations, vulnerability research	10	secops director, threat modeler, security architect, red-teamer, vulnerability analyst
Tax law, IRS audit, forensic accounting	7	audit docket clerk, tax credit preparer, IRS examining agent
Civil and structural engineering, plan check	7	project coordinator, structural scheme designer, design revision engineer, code-compliance examiner, calculation package engineer, independent structural reviewer
Investigative journalism, editorial	7	assigning editor, investigative reporter, skeptical investigations editor, story writer, fact-checking editor
Finance, quant trading, M&A, risk	6	deal orchestrator, valuation strategist, investment committee skeptic, term-sheet drafter, due-diligence auditor
Drug discovery, pharmacology, toxicology	4	program director, pharmacologist, toxicology skeptic, trial designer, regulatory auditor
Environmental permitting, impact assessment	4	permitting coordinator, impact assessment lead, regulatory challenge reviewer, EIA report author, permit-compliance auditor
Aerospace mission operations, flight dynamics	3	mission commander, astrodynamics architect, maneuver adjuster, orbital-mechanics reviewer, burn-sequence compiler, physics-simulator proxy
Patent law and IP	3	patent attorney, patent examiner
Policy, campaign, legislation	3	policy coordinator, macro-theorist, intervention tuner, economic skeptic, legislation drafter, compliance stress-tester
Other (game and screen production, fire incident response, mathematical proof verification, anomaly detection, culinary operations)	8	showrunner, narrative architect, plot-hole critic, scene writer, continuity auditor
Software engineering and research	10	coder, reviewer, dispatcher, scientist, theorist, phdstudent, librarian

Table 18: Professional-domain coverage of PerspectiveGap.

N CONCRETE REFERENCE-MAPPING EXAMPLE

This appendix makes the reference policy auditable on an actual benchmark scenario. We use the `dispatcher_theoryloop_codeloop` scenario. It shows the background given to the model, the full text of every fragment, the *"need-only"* instruction, the reference assignment, and textual evidence for representative boundary decisions.

> I need you to set up a 5-agent pipeline that solves mathematical optimization problems by developing theoretical analysis and solver code in parallel, with each side informing the other. A problem arrives as a natural-language statement in `problem.md`; read it before starting your part.
> The five agents:
> - **dispatcher** orchestrates the run by alternating one theory step with one code step and tracking progress. It does no modeling, coding, or reviewing itself.

- **theory loop** — the **theorist** writes the theoretical analysis to `THEORY.md`; the **theory-reviewer** judges it. The theorist reads `SOLUTION.md` on each revision pass to incorporate empirical evidence.
- **code loop** — the **coder** writes solver code and its results to `SOLUTION.md`; the **code-reviewer** judges them. The coder reads `THEORY.md` on each revision pass to align the implementation with the latest theoretical claims.

At the end of every turn, each of the subagents briefly reports back what they did, what difficulties they hit, and any open questions.
The pipeline hands work off through three files:

- `problem.md` — the original problem, read-only. All four domain agents read it.
- `THEORY.md` — the theoretical analysis. Theorist writes; theory-reviewer reviews; coder and code-reviewer read it to know which bounds and invariants the implementation must respect.
- `SOLUTION.md` — the solver code and its results. Coder writes; code-reviewer reviews; theorist and theory-reviewer read it to ground the theory in observed numbers.

Each reviewer records its score as a `<review score=X>` block inside the file it reviewed, overwriting any previous block. The dispatcher reads those scores to drive the loops. Unlike a plan-then-code pipeline where the plan must converge before the code starts, here neither side ever finishes first. Both loops alternate forever, capped at a fixed iteration count. Theory may tighten after seeing code's empirical numbers; code may correct after seeing theory's refined bounds.

The prompt then shows the twelve fragments below, listed here in canonical order; the rendered prompt shuffles and relabels them (Section 3) and adds one distractor.

`f1`: problem context.
You are working on a mathematical optimization problem, which may be linear, mixed-integer, nonlinear, or a combination. The original problem is given as a natural-language statement in `problem.md`; read it before starting your part.

`f2`: theorist deliverable rule.
Produce the theoretical analysis for the problem and write it to `THEORY.md`. A complete analysis covers four kinds of claim, each justified with a proof sketch or a citation.

- **Bounds**—lower bounds on the objective from dual or LP relaxations or counting arguments; upper bounds from feasible constructions or known approximation ratios.
- **Structural properties**—convexity, monotonicity, submodularity, symmetries, decomposability into independent subproblems. State what holds, where, and under which assumptions.
- **Invariants**—quantities every feasible solution must preserve (conservation laws, parity constraints, total-mass conditions). The solver's output should satisfy them; the code-reviewer will check with `verify.py`.
- **Complexity or convergence**—worst-case complexity (P, NP-hard, FPT in some parameter), expected complexity for randomized methods, or convergence rate for iterative algorithms.

Ground your claims against `SOLUTION.md` if it has reported numbers. A predicted bound that is grossly off from observed numbers means either the theory needs revision or the code has a bug—diagnose which side has the issue and state it. Be concrete. "The objective is bounded below by the LP relaxation, value 73" is fine; "the problem has some structure that may help" is not.

`f3`: theory toolbox.
Match the tool to the problem structure; do not force an ill-suited one. The list below is useful in theoretical analysis.

Tool	Suitable for
Lagrangian relaxation / dual bounds	MILP, MINLP; lower bound on minimization, primal-dual gap analysis
KKT conditions	Differentiable NLP; certify local optimality of an interior point
Convex analysis (strong duality, convexity certificates)	LP, QP, convex NLP; global optimality from local
Monotonicity / submodularity	Combinatorial selection; (1-1/e) and similar greedy approximation ratios
Fixed-point theorems (Banach, Brouwer, Kakutani)	Iterative methods; existence of equilibrium, geometric convergence
Concentration inequalities (Hoeffding, Chernoff, Azuma)	Randomized algorithms; high-probability bounds on cost or error
Complexity reductions (NP-hardness, APX-hardness)	Exact-solution lower bounds; justify resorting to a heuristic
Parametric / sensitivity analysis	Stability of optimal value under perturbation of data

`f4`: coder deliverable rule.
Read `problem.md` and `THEORY.md`, then implement the solver:
- Include every import. Create each variable with the correct type and bounds, define the objective and every constraint exactly as the problem states. Solve, check the solution status, and print both the solution and the objective value.
- Make the implementation testable against `THEORY.md`. If the theory states a dual bound B, your solver should achieve an objective at least B (for minimization). If the theory states an invariant, your output should satisfy it.
- Run the code yourself and watch it through to completion—do not stop while it is still running. If it errors, find and fix the root cause; do not bury errors under try/except.
- Sanity-check the answer against `THEORY.md`. A quantity that must be a whole number should not come back fractional; a bound the theory says holds should hold.

Write the code and its results up in `SOLUTION.md`.

`f5`: solution-doc writing rule.
Keep `SOLUTION.md` lean. Its body—everything except the `<review>` block—must stay under 3000 words (~20KB), about a four-page note; record only the final state and move iteration-by-iteration history into a separate `CHANGELOG.md`. Bold text must not exceed 5% of the file: if everything is bold, nothing is.

`f6`: theory-review rule.
Review `THEORY.md` against `problem.md` and against the numbers reported in `SOLUTION.md` (if any).
- **Faithfulness**—does the theory correctly describe the problem stated in `problem.md`? Check definitions, decision variable types, constraint translations.
- **Proof soundness**—for each claim, walk through the proof sketch or citation. Flag hand-waving steps. Verify cited inequalities apply under the stated assumptions.
- **Empirical grounding**—open `SOLUTION.md`. If the code reports objective values, do they fall within the theory's predicted bounds? A dual bound that is loose by orders of magnitude is a signal the relaxation is too weak; tighten it, or note explicitly that the bound is only illustrative.

Be brief and specific. Do not write theory yourself—only review. If a claim is wrong, say which one and why.

`f7`: code-review rule.
Review `SOLUTION.md` against `problem.md` and against `THEORY.md`.
- Inspect the reported result for anomalies, unreasonable objective values, or constraint violations stated directly in `problem.md`.
- Open `THEORY.md`. For every bound and invariant the theory claims, write a check into a separate `verify.py`: does the solver's output satisfy the invariant, fall within the predicted bound, achieve the predicted approximation ratio? Run it. Name any variable or quantity that violates.
- Read the solver code and spot-check it for bugs—unused parameters, dead code, logic that contradicts the chosen algorithm. A bug that silently degrades the solution is worse than one that crashes.
- Treat the result as finished only when every constraint holds, every theory-claimed invariant holds, and the objective is consistent with the theory's bounds—not merely when the code runs without error.
- Check the length of `SOLUTION.md`: if it exceeds 20KB, give a warning, but do not deduct points for it.
- Check the bold ratio of `SOLUTION.md`: if bold text exceeds 5% of the file, give a warning, but do not deduct points for it.

f8: reviewer scoring rule.
Score the work from 1 to 10, and record your verdict by overwriting the single <review score=X> block at the end of the document you reviewed—overwrite the existing one, never append a second.
Score strictly on real merit. Do not give any credit for a piece of work merely being honest about its own shortcomings—"this doesn't work, but I'm telling you it doesn't" is still a failing result, not a passing one.

f9: dispatch rule.
You orchestrate the run; you never model, code, or review yourself—you only dispatch the right agent at the right time.
- **Init**—create STATE.md with iter: 0 and last_side: none. Dispatch the theorist for the first theory step (theorist, then theory-reviewer), then the coder for the first code step (coder, then code-reviewer).
- **Alternation loop**—alternate one theory step with one code step. Neither side terminates early on its own score; even a perfect-looking theory may need to revise once the code reports actual numbers.
- Cap the total at 20 iterations (10 theory steps + 10 code steps). After the cap, stop and summarize.
- Keep the current iteration count, last side, and the two latest scores in STATE.md, accurate after each dispatch. Read each score from the reviewer's <review score=X> block. Commit progress after each step.
- Read each agent's brief narrative reply on return. If an agent reports an unresolved error, append it to error-log.txt (timestamp + agent name + content) and continue with the next step.

f10: theorist workflow.
The theoretical analysis lives in THEORY.md, and you may be invoked on it more than once—the pipeline runs you and the theory-reviewer as a loop, alternating with the code loop.
- If THEORY.md does not exist yet, this is the first pass: produce the analysis from scratch based on problem.md.
- If THEORY.md already exists, it carries the theory-reviewer's <review> block. Read that feedback. Then open SOLUTION.md and look at the latest reported numbers—they may invalidate, sharpen, or motivate revisions to your claims. Revise THEORY.md in place to address every reviewer point raised and to incorporate any empirical evidence that changes your analysis. Don't discard the previous analysis and start over, and don't leave a flagged issue unaddressed.

f11: coder workflow.
The solver code and its results live in SOLUTION.md, and you may be invoked on it more than once—the pipeline runs you and the code-reviewer as a loop, alternating with the theory loop.
- If SOLUTION.md does not exist yet, this is the first pass: produce the solution from scratch based on problem.md.
- If SOLUTION.md already exists, it carries the code-reviewer's <review> block. Read that feedback. Then open THEORY.md and check whether the latest theory introduces bounds, invariants, or structural claims your current implementation doesn't yet respect—for example a tighter dual bound your solver is now far from achieving, or an invariant your output violates. Revise SOLUTION.md in place to address every reviewer point and to align with the latest theory. Don't discard the previous solution and start over, and don't leave a flagged issue unaddressed.

f12: report-back instruction.
When you finish, briefly report to the dispatcher: what you did, what difficulties you hit (or didn't resolve), and any open questions.

After the fragment block, the prompt tells the model: *"Each agent's prompt should contain only the information that agent needs to do its job."* and the output format given in Appendix F. Table 19 shows the resulting reference assignment.

The main boundary decisions in this example follow from the stated role responsibilities:

1. f12 belongs to all four domain agents because the background says *"At the end of every turn, each of the subagents briefly reports back what they did, what difficulties they hit, and any open questions."* It does not belong to the dispatcher: the dispatcher receives those reports, but f12 tells domain agents what to report.

2. f6 belongs to the theory-reviewer, not to the theorist. Its first sentence is *"Review THEORY.md against problem.md and against the numbers reported in SOLUTION.md (if any)."* Giving it to the theorist leaks the critic's instructions into the actor's prompt.

3. f2 belongs to the theorist, not to the theory-reviewer. It begins with *"Produce the theoretical analysis for the problem and write it to THEORY.md."* Although it is about theory, it specifies the actor's deliverable rather than the reviewer's job.

Role	Reference fragments	Rationale
Dispatcher	`f9`	The background says the dispatcher *"orchestrates the run by alternating one theory step with one code step and tracking progress"* and *"does no modeling, coding, or reviewing itself."* It therefore needs the dispatch rule only.
Theorist	`f1`, `f2`, `f3`, `f10`, `f12`	The theorist needs the problem context, its deliverable rule, the theory toolbox, its workflow across turns, and the instruction to report back.
Theory-reviewer	`f1`, `f3`, `f6`, `f8`, `f12`	The theory-reviewer needs the problem context, enough theory-tool context to judge the analysis, the theory-review rule, the reviewer scoring rule, and the instruction to report back.
Coder	`f1`, `f4`, `f5`, `f11`, `f12`	The coder needs the problem context, its deliverable rule, solution-document writing rules, its workflow across turns, and the instruction to report back.
Code-reviewer	`f1`, `f7`, `f8`, `f12`	The code-reviewer needs the problem context, the code-review rule, the reviewer scoring rule, and the instruction to report back.

Table 19: Reference role-fragment assignment for one PerspectiveGap scenario.

4. The code side follows the same boundary. `f4` begins with *"Read problem.md and THEORY.md, then implement the solver:"*; `f5` says *"Keep SOLUTION.md lean."*; and `f11` says *"The solver code and its results live in SOLUTION.md, and you may be invoked on it more than once."* These are coder-side instructions. By contrast, `f7` begins with *"Review SOLUTION.md against problem.md and against THEORY.md,"* and `f8` says to *"Score the work from 1 to 10"*; those are reviewer-side instructions.