DisasterBench: Benchmarking LLM Planning under Typed Tool Interface Constraints

Anonymous ACL submission

Abstract

Disasters cause severe societal impacts, demanding rapid coordination of heterogeneous AI tools, from satellite analysis to flood prediction and damage assessment, into coherent multi-step workflows. As LLMs increasingly serve as orchestrators of such pipelines, effective coordination requires more than selecting semantically plausible tools: LLMs must generate executable workflows with correct parameter binding and dependency propagation. We introduce **DisasterBench**, a benchmark for evaluating structured multi-agent planning over semantically similar but operationally distinct disaster-response tools. To enable step-level failure attribution, we further propose **First-Point-of-Failure (FPoF)**, which localizes the earliest root cause in a predicted workflow, separating primary errors from downstream cascading effects. Our evaluation reveals three findings: planning method effectiveness depends strongly on model capacity; tool mismatch and parameter-binding errors dominate first failures, revealing semantic grounding and execution consistency as distinct bottlenecks; and verbose intermediate reasoning can create instruction clash with structured output requirements, disrupting plan generation. Together, these findings highlight a fundamental gap between semantic reasoning and execution-grounded coordination, underscoring the need for planning frameworks that jointly model semantic intent, execution constraints, and workflow consistency. All code, data, and evaluation resources are available at the project page.

1 Introduction

Modern disaster response increasingly relies on coordinating heterogeneous AI capabilities into coherent multi-stage workflows. Emergency management agencies, first responders, and domain analysts must rapidly orchestrate specialized tools for tasks such as satellite image analysis, precipitation nowcasting, flood modeling, and damage assessment. These workflows are operationally constrained: outputs from one stage often serve as structured inputs to downstream stages, making workflow correctness dependent on both semantic grounding and execution consistency. Recent advances in LLM-powered multi-agent systems (Tran et al., 2025) have created new opportunities for automating such workflow orchestration. However, effective deployment requires reliable *workflow grounding*, where models must not only identify relevant tools but also generate executable workflows with correct parameter bindings and inter-step dependencies.

Despite recent progress in multi-agent benchmarking (Liu et al., 2024; Shen et al., 2024) and tool-augmented LLMs (Qin et al., 2024; Patil et al., 2024), existing benchmarks provide limited evaluation of workflow grounding in disaster-response settings. This limitation is especially critical because disaster-response tools are often semantically similar but operationally distinct. For example, multiple flood-analysis tools may appear equally relevant for a task while requiring different inputs, spatial assumptions, or downstream compatibility. Selecting an operationally incompatible tool may therefore invalidate downstream workflow stages even when the high-level intent appears reasonable.

To address these challenges, we introduce **DisasterBench**, a benchmark for evaluating workflow grounding in disaster-response multi-agent systems. DisasterBench is built around 26 openly available disaster-response tools spanning satellite image analysis, adverse-weather perception, hydrological modeling, and post-disaster mobility prediction. From expert-designed disaster-response workflows, we derive 233 planning tasks ranging from single-step perception to multi-stage branching workflows. Models are provided only with tool specifications and interface schemas, without access to the underlying workflow structure, requiring them to infer executable plans directly from natural-language

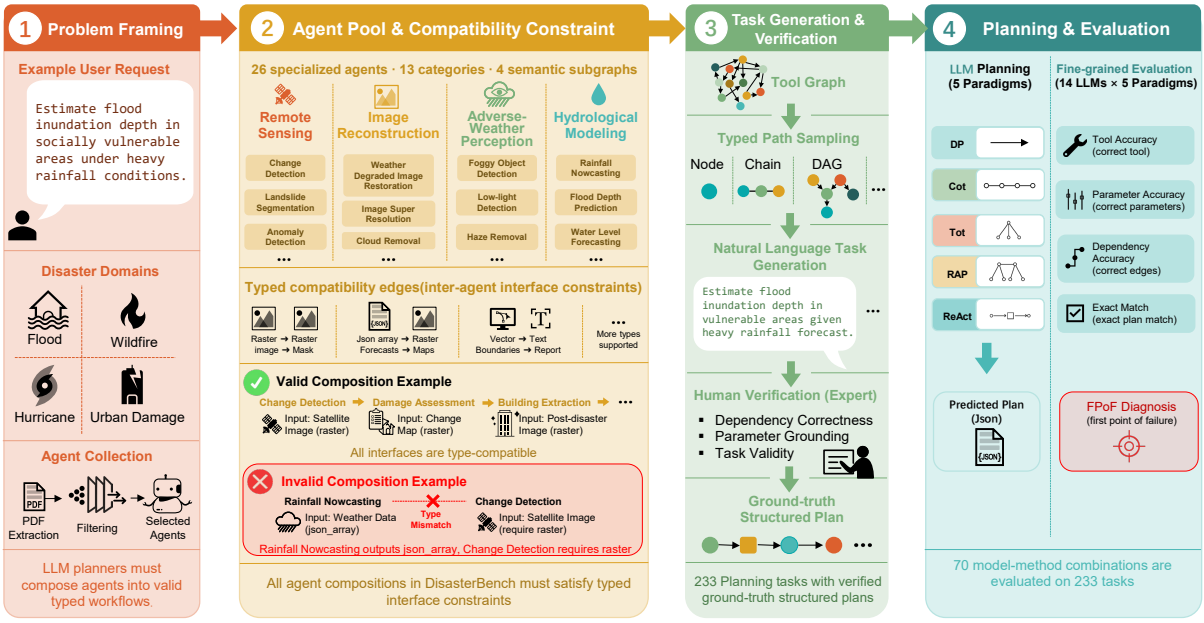


Figure 1: Overview of the DisasterBench benchmark construction and evaluation pipeline, comprising four stages: tool pool construction with typed inter-agent interface constraints, workflow execution constraint definition, natural-language task generation with expert verification, and multi-paradigm LLM evaluation with FPoF diagnosis.

task descriptions. To enable precise failure attribution beyond aggregate metrics, we further propose **First-Point-of-Failure (FPoF)**, a diagnostic framework that localizes the earliest root cause in a generated workflow, separating primary failures from cascaded downstream errors.

Our evaluation of 14 LLMs across five planning paradigms yields three primary findings. First, plan depth is a universal bottleneck: performance degrades sharply beyond two steps across all models and methods, suggesting that multi-step coordination imposes compounding difficulty that increased model capacity alone does not resolve. Second, tool mismatch is the dominant workflow-grounding failure, revealing that distinguishing among semantically similar but operationally distinct tools remains a major challenge. Third, verbose intermediate reasoning can exhibit instruction clash with structured output requirements, disrupting workflow generation rather than improving it. Our contributions are as follows:

DisasterBench. We introduce DisasterBench, a benchmark for evaluating workflow grounding in disaster-response multi-agent systems.

First-Point-of-Failure (FPoF). We propose a step-level diagnostic framework for localizing root causes in predicted workflows.

Empirical Analysis. We provide a systematic empirical analysis of structured multi-agent planning failures, identifying bottlenecks including depth-driven collapse, workflow-grounding failures, and instruction clash.

2 Related Work

2.1 Tool-Augmented and Multi-Step Planning Benchmarks

Prior benchmarks such as ToolBench (Qin et al., 2024), APIBench (Patil et al., 2024), AgentBench (Liu et al., 2024), and TaskBench (Shen et al., 2024) evaluate LLMs on tool selection, invocation, and multi-step workflow orchestration, but treat agents as freely composable without evaluating whether resulting workflows remain executable under downstream compatibility and execution constraints. DisasterBench instead evaluates executable workflow grounding over semantically similar but operationally distinct tools, and provides step-level failure attribution through FPoF to diagnose why semantically plausible workflows fail to remain executable.

2.2 Structured Generation and Executable Planning

Structured generation and semantic parsing study how models map natural-language intent into executable representations (Yin and Neubig, 2017; Scholak et al., 2021; Shin et al., 2021), while reasoning paradigms such as CoT (Wei et al., 2022),

Category	Tool(s)	#
Image Gen./Restoration	Temporal_High_Resolution_Image_Generation, High-Resolution_Image_Reconstructor, Metadata_and_Text_Prompt_Image_Generation, Weather_Degraded_Image_Restoration	4
Classification	Temporal_Image_Sequence_Classifier, RGB_GeoImage_Classifier, Multi_Spectral_Classifier	3
Anomaly Detection	Anomaly_Detection_Forest, Landslide_Segmentation, Urban_Anomaly_Detection	3
Change Detection	Change_Mapping_and_Detection	1
Object Det./Seg.	Foggy_Scenario_Object_Detection, Low-Light_Object_Detection, Geospatial_Object_Segmentation	3
Crowd Counting	Crowd_Counting_in_Adverse_Weather	1
Building Damage	Building_damage_assessment	1
Flood & Precipitation	Precipitation_Nowcasting, Flood_depth_prediction, depth_speed_model	3
Text & Event Proc.	Toponym_Detection, Event_detection	2
Video Analysis	Video_anomaly_detection	1
Mobility Prediction	Post_Disaster_Mobility_Recovery, Multimodal_mobility_prediction_under_events	2
Multimodal Reasoning	GeoChat	1
Data Conversion	precipitation_data_convert_tool	1
Total		**26**

Table 1: Summary of the 26 specialized agents in DisasterBench, grouped into 13 functional categories spanning perception, prediction, generation, and reasoning tasks in disaster-response workflows.

ToT (Yao et al., 2023), ReAct (Yao et al., 2022), and RAP (Hao et al., 2023) improve multi-step planning through explicit intermediate reasoning. DisasterBench is complementary to both lines of work: it evaluates executable workflow grounding beyond syntactic validity alone, and reveals that verbose reasoning traces can interfere with structured executable generation. We term this phenomenon *instruction clash*.

2.3 AI for Disaster Management

Prior work has advanced individual disaster-response capabilities including remote-sensing-based disaster analysis (Algiriyage et al., 2022), flood mapping and forecasting (Bentivoglio et al., 2022), and damage assessment (Gupta et al., 2019), with recent benchmarks evaluating disaster-domain QA (Chen et al., 2026) and text-to-SQL (Liu et al., 2025). DisasterBench shifts focus to multi-agent workflow grounding, studying whether LLMs can coordinate semantically similar but operationally distinct disaster-response tools into execution-consistent workflows and diagnosing why fail.

3 Benchmark Construction

Disaster-response workflows present a distinctive multi-agent coordination challenge: tools are often functionally overlapping but operationally distinct, and valid workflows must maintain consistent parameter bindings and inter-step dependencies across multiple stages. As a result, workflow correctness depends not only on selecting plausible tools, but also on generating execution-consistent plans under downstream execution constraints. DisasterBench is designed to systematically evaluate this workflow grounding problem. Unlike existing benchmarks that primarily evaluate tool invocation or workflow decomposition, DisasterBench requires models to infer executable workflows over structured disaster-response pipelines without access to the workflow structure. Figure 1 provides an overview of the benchmark construction pipeline; the following subsections detail each component.

3.1 Tool pool

We construct a pool of 26 specialized tools spanning 13 functional categories covering core perception, prediction, and reasoning tasks in disaster-response workflows (Table 1). These agents span heterogeneous modalities and task settings, from satellite image analysis and precipitation nowcasting to adverse-weather perception, damage assessment, and post-disaster mobility modeling, reflecting the diverse capabilities required in disaster-response workflows.

The pool is designed to probe fine-grained workflow grounding rather than simple tool retrieval. Many tools are *functionally overlapping but operationally distinct*: forest anomaly detection and urban anomaly detection both identify abnormal visual patterns but apply to different disaster contexts; foggy-object detection and low-light-object

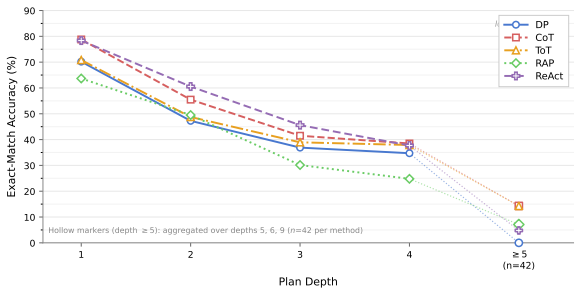


Figure 2: Exact-match accuracy (%) as a function of ground-truth plan depth, micro-averaged over all models for each planning method. All methods exhibit a sharp performance cliff as workflow depth increases, confirming that execution-consistency requirements compound with plan depth in DisasterBench.

detection both target degraded visual conditions but require different operational assumptions. Such overlaps require models to perform precise semantic disambiguation while preserving valid downstream dependencies. The difficulty of DisasterBench arises not from the number of tools, but from the high semantic similarity among operationally distinct candidates, as further supported by the tool-grounding failure patterns in Section 4.5 and quantified through description-level semantic similarity analysis (Appendix D.1).

Each tool is described by its functional role and input/output schema, defining the structured fields used in ground-truth workflows and model predictions. Tools are sourced from publicly available models reported in top-tier venues, selected through semi-automated filtering and manual verification for domain relevance, functional diversity, and implementation availability (Appendix H.2).

3.2 Workflow Execution Constraints

We define an expert-curated set of workflow execution constraints over the 26 tools, specifying which tool outputs are admissible inputs to downstream stages. The resulting structure contains 81 directed dependencies organized into four coherent disaster-response workflow families; full details are provided in Appendix C. These constraints define the executable planning space in DisasterBench: a workflow is considered valid only when tool selection, parameter bindings, and inter-step dependencies remain consistent across stages. DisasterBench uses a single global workflow structure shared across all tasks, reflecting the reusable nature of disaster-response tool ecosystems. The same tool may participate in multiple workflows, and the same intermediate output may support different downstream analyses depending on the response objective. Models are provided only with tool specifications and interface schemas, requiring them to infer executable workflows from task context alone without exposure to the underlying workflow structure itself.

3.3 Task Generation

Each task in DisasterBench is grounded in a valid executable workflow derived from the workflow execution constraints and transformed into a naturalistic planning scenario through an LLM-assisted pipeline with human verification. A workflow specifies the sequence of tool calls, parameter bindings, and inter-step dependencies required to complete the task; workflow lengths range from 1 to 9 steps.

Workflow Sampling. To capture the compositional diversity of disaster-response workflows, we categorize tasks into three compositional settings: *Node* (single-step tasks invoking one tool), *Chain* (multi-step sequential workflows), and *Branching* (multi-step workflows containing one-to-many dependencies). The benchmark contains 233 tasks in total (Node: 35, Chain: 166, Branching: 32). Compositional difficulty increases with workflow depth: Node tasks primarily test tool grounding, while Chain and Branching tasks additionally require parameter propagation and dependency consistency across multiple stages. Detailed statistics are provided in Appendix D.

Description Synthesis. Task descriptions are generated through a self-instruct procedure (Wang et al., 2022a) using GPT-4o, deliberately reflecting user intent without exposing the underlying execution constraints. Descriptions are designed to prevent models from exploiting surface-level patterns rather than inferring executable workflows compositionally. Prompts are provided in Appendix H.

Human Verification. All task–workflow pairs undergo two rounds of expert review by annotators with backgrounds in remote sensing and disaster management. Annotators verify semantic alignment between task descriptions and ground-truth workflows, ensure that descriptions do not leak structural cues such as tool names or intermediate representations, and confirm that workflows remain valid under the execution constraints. Ambiguous or inconsistent instances are revised or discarded, yielding a dataset with expert-verified one-to-one alignment between task descriptions and executable workflows.

3.4 Evaluation Protocol

Strict Exact Match. We evaluate model-generated plans against ground truth using a strict exact-match (EM) criterion. Given a predicted plan $\hat{P} = (\hat{s}_1, \ldots, \hat{s}_{\hat{T}})$ and a ground-truth plan $P = (s_1, \ldots, s_T)$, where each step s_t specifies the agent identity and its execution-consistent bindings and dependencies, a prediction is correct if and only if:

$$\mathrm{EM}(\hat{P}, P) = \mathbb{I}(\hat{T} = T \wedge \forall t,\ \hat{s}_t = s_t). \quad (1)$$

Step equality $\hat{s}_t = s_t$ is defined jointly over tool identity, parameter bindings, dependency structure, and dependency content. This strict criterion is necessary because even a single incorrect binding or dependency reference may route downstream tools to different intermediate outputs, obscuring the workflow-grounding failure that DisasterBench is designed to diagnose. Step order is likewise non-interchangeable because downstream steps consume outputs produced by specific upstream tools.

Partial-Credit Diagnostic Metrics. While EM captures end-to-end correctness, it does not reveal which dimension of a plan failed. We complement EM with three plan-level partial-credit metrics, each a task-level boolean averaged over all tasks: *Tool Accuracy* (step indices and agent identities match), *Parameter Accuracy* (tool conditions plus matching input/output bindings at every step), and *Dependency Accuracy* (tool conditions plus matching inter-step dependencies and references at every step). By construction, both Parameter and Dependency Accuracy imply Tool Accuracy, and EM implies all three; however, Parameter and Dependency Accuracy are not mutually comparable, since they isolate disjoint plan dimensions: parameter binding and inter-step dependency propagation, respectively. The gaps between these metrics quantify the separation between semantically plausible tool selection and execution-consistent workflow generation, a diagnostic property we term the *semantic-execution gap*.

First-Point-of-Failure (FPoF). To attribute failures at the step level, we propose First-Point-of-Failure (FPoF), which localizes the earliest divergence between a predicted and ground-truth plan. Each first failure is classified into three categories: *tool mismatch* (incorrect tool grounding), *Parameter Binding Error* (incorrect inputs, outputs, or upstream references), and *Structural Error* (failure to produce an executable plan, including hallucinated steps, premature termination, or format violations). By isolating the earliest root cause, FPoF separates primary execution failures from cascaded downstream errors. Sub-category definitions and per-method distributions are provided in Section 4.5 and Appendix F.

4 Benchmark Evaluation

Our evaluation is designed not only to rank models, but also to diagnose where executable workflow grounding fails. We therefore report exact-match accuracy together with baselines, partial-credit metrics, and First-Point-of-Failure analyses, separating semantic tool-grounding failures from execution-level parameter binding, dependency consistency, and structural generation errors.

4.1 Experimental Setup

Models. We evaluate 14 large language models spanning a wide range of scale, training recipe, and reasoning specialization, grouped into three tiers: *Frontier* proprietary models, *Strong open-source* models from competitive open-weight families and reasoning-optimized variants, and *Efficiency-oriented* models covering smaller Llama/Qwen variants and the Ministral family. The full model list with category assignments is shown in Table 2; complete identifiers, API endpoints, and access dates are provided in Appendix A. This selection enables controlled comparisons across model capacity and reasoning specialization in executable workflow grounding. Complete implementation details and hyperparameter configurations are provided in Appendix B.

Planning Methods. We evaluate five representative planning strategies.

Direct Prompting (DP): Generates the complete plan in a single forward pass.

Chain-of-Thought (CoT) (Wei et al., 2022): Elicits intermediate reasoning before generating the final plan.

Tree-of-Thought (ToT) (Yao et al., 2023): Performs breadth-first search over candidate plans with LLM-based pruning.

Reasoning via Planning (RAP) (Hao et al., 2023): Uses Monte Carlo Tree Search guided by usefulness and self-consistency rewards.

ReAct (Yao et al., 2022): Interleaves reasoning and tool selection over the current workflow state.

Structural Baselines. We include four non-LLM baselines to isolate the contribution of language-grounded workflow generation; all achieve 0% EM

Category	Model	DP	CoT	ToT	RAP	ReAct
Frontier	Gemini 3.1 Pro Preview	68.24	72.53	<u>**73.39**</u>	72.10	69.96
	GPT-5.4	54.08	62.23	62.23	**65.67**	64.38
Strong Open-source	DeepSeek-V3.2	57.94	36.05	50.21	34.33	**61.80**
	DeepSeek-R1	54.08	**62.66**	53.22	35.19	56.22
	Qwen3-Max	55.79	62.66	**63.95**	63.09	62.23
	Qwen3-Max-Thinking	57.51	**66.52**	62.23	61.37	61.37
	Qwen3.5-27B	58.37	**62.23**	59.23	58.80	59.23
	Gemma-4-31B	**66.52**	63.09	58.37	50.64	62.23
	Llama-3.3-70B	38.63	**58.80**	41.63	28.76	38.63
Efficiency-oriented	Qwen3.5-9B	45.92	61.37	59.23	42.92	**65.67**
	Llama-3.1-8B	15.88	26.18	16.31	**28.33**	26.61
	Ministral-14B	20.17	33.48	16.31	18.88	**50.64**
	Ministral-8B	25.32	31.33	21.03	12.88	**44.21**
	Ministral-3B	10.30	21.03	18.88	3.86	**36.91**
Baselines	Random	0.00 (Agent Acc.: ∼0%)				
	DAG-Greedy (TF-IDF)	0.00 (Agent Acc.: 5.58%)				
	Shortest Path	0.00 (Agent Acc.: 51.07%)				
	Oracle-Random	0.00 (Agent Acc.: 100%)				
Average (LLMs)		44.91	51.44	46.87	41.20	54.29

Table 2: Exact-match accuracy (%) of 14 LLMs across five planning methods on DisasterBench. **Bold** indicates the best result per model. The overall best result is <u>underlined</u>. Structural baselines do not distinguish planning methods and are reported as single EM scores.

regardless of how much structural information they receive, confirming that execution-consistent workflow generation requires language-grounded parameter binding beyond tool selection alone (Appendix K).

Implementation. All models are queried via official APIs with prompt-level JSON schema enforcement only; unparseable outputs are treated as incorrect. Full configurations are provided in Appendix B.

4.2 Main Results

Table 2 reports exact-match (EM) accuracy for all 14 models across five planning methods. The results reveal that planning strategy and model capacity interact in non-uniform ways: no single planning strategy is universally optimal, and method effectiveness depends systematically on model tier and workflow-grounding performance.

Finding 1: Model capacity is the primary bottleneck. The performance gap between frontier and efficiency-oriented models substantially exceeds the variation introduced by any individual planning strategy, suggesting that maintaining execution consistency across multi-step workflows is fundamentally constrained by model capacity rather than reasoning paradigm alone.

Finding 2: Method optimality is tier-dependent. Frontier models peak under search-based methods (Gemini under ToT, 73.39%; GPT-5.4 under RAP, 65.67%), while strong open-source models most often benefit from CoT. Efficiency-oriented models almost uniformly favor ReAct, with gains of up to +17 percentage points (pp) over CoT (Ministral-14B: 50.64% vs. 33.48%). This suggests that global search-based reasoning is reliably exploitable only by sufficiently capable models, while local interleaved reasoning is more robust when execution consistency becomes difficult to maintain across steps.

Finding 3: Search-based methods require reliable self-evaluation. RAP is competitive on frontier models (Gemini: 72.10%, GPT-5.4: 65.67%) but degrades sharply on weaker models (Llama-3.1-8B: 28.33%, Ministral-3B: 3.86%), even underperforming DP for several efficiency-tier models. This pattern suggests that weaker models struggle to reliably evaluate partially generated workflows, causing MCTS search to amplify rather than correct execution errors, a hypothesis we examine further in Section 4.5.

4.3 Semantic-Execution Gap

We define the **semantic-execution gap** as the per-task difference between Tool Accuracy (A) and exact-match accuracy (EM) on the same prediction. A positive gap indicates that the model selected the correct tool sequence but failed to produce a fully executable plan along at least one parameter-binding or dependency-propagation re-

Error Type	DP	CoT	ToT	RAP	ReAct
tool mismatch	50.4	54.6	52.3	32.2	58.2
Parameter Binding	34.6	31.7	31.0	50.1	28.7
Parameter	34.5	31.5	29.3	48.5	28.7
Dependency	0.0	0.2	1.7	1.2	0.0
Dep.-content	0.1	0.0	0.1	0.4	0.0
Structural	15.0	13.7	16.7	17.7	13.1
Halluc. steps	13.0	11.9	15.6	6.6	11.3
Early stop	1.9	1.8	1.1	11.1	1.6
Format	0.1	0.0	0.0	0.0	0.2
Failures (n)	1797	1584	1733	1918	1491

Table 3: FPoF error distribution by planning method (% of method failures). Fine-grained sub-categories shown in italics. Columns sum to 100% within each method.

quirement. Full results are reported in Table 17 (Appendix F.2); the gap is present for every model under every method, with average values ranging from ∼5 to ∼11pp across all model–method combinations, confirming that *semantically plausible tool selection is consistently easier than execution-consistent workflow grounding*.

The gap is more strongly associated with planning method than with model tier. Averaged across all models, the gap varies systematically with planning method (DP: 8.2pp, CoT: 6.7pp, ToT: 9.0pp, RAP: 7.9pp, ReAct: 6.8pp) but does not cleanly separate model tiers. For example, Gemini's average gap (9.3pp) is comparable to that of Ministral-3B (9.7pp). Methods with explicit step-by-step reasoning, such as CoT and ReAct, produce the smallest gaps, suggesting that articulated intermediate reasoning helps preserve local execution consistency even when global workflow generation remains error-prone. Conversely, single-pass generation and global search-based paradigms yield larger gaps, consistent with their greater susceptibility to execution-level failures.

Parameter binding is an independent execution bottleneck. The largest gap occurs for Llama-3.3-70B under DP, which selects correct tools on 60.94% of tasks but achieves only 38.63% EM, corresponding to a 22.31pp gap. Even when tool grounding succeeds, execution-consistent parameter binding and dependency propagation remain distinct failure sources, confirming that correct tool sequences are necessary but not sufficient for executable workflow grounding. The DeepSeek-V3.2 CoT collapse, by contrast, is not associated with a widening of this gap but with a uniform drop in both A and EM; we treat it separately as an instance of *instruction clash* in Section 4.6.

These results suggest that workflow grounding introduces execution-consistency requirements beyond semantic tool selection alone: Models must not only identify plausible tools, but also maintain valid parameter bindings and dependency propagation throughout the workflow.

4.4 Depth Analysis

Figure 2 plots exact-match accuracy as a function of workflow depth, micro-averaged across all models for each planning method. Performance degrades sharply as workflow depth increases across all methods and model tiers: even the strongest model, Gemini 3.1 Pro Preview, falls from near-perfect accuracy at depth 1 to roughly 53–61% at depth 4 across methods, confirming that deeper workflows introduce a difficulty dimension not eliminated by increased model capacity alone. Method differences also narrow with depth, suggesting that accumulated execution errors eventually overwhelm the benefits of individual reasoning strategies. This convergence is consistent with the view that deeper workflows impose compounding execution-consistency requirements rather than simply increased reasoning complexity. We examine this pattern further through failure attribution in Section 4.5.

4.5 Failure Mode Analysis

Table 3 reports the FPoF error distribution aggregated over all 70 model–method combinations and 8,523 total failures. Because FPoF records the earliest divergence from the ground-truth workflow, these distributions reflect likely root causes rather than downstream cascading symptoms. Tool mismatch and Parameter Binding Error together account for over 80% of failures under every method, confirming that executable workflow grounding failures dominate over low-level structural generation failures.

Tool mismatch is the dominant semantic grounding failure. For four of the five methods, tool mismatch is the most frequent first error (50.4%–58.2%), reflecting the difficulty of semantic discrimination among functionally overlapping but operationally distinct tools. RAP is a qualitative exception: under RAP, Parameter Binding Error (50.1%) overtakes tool mismatch (32.2%), suggesting that MCTS often reaches a plausible initial tool but fails to maintain execution-consistent parameter bindings and dependency propagation in subsequent steps.

RAP exhibits a distinctive early-termination

signature. RAP shows substantially elevated Early Stop failures (11.1% vs. 1–2% elsewhere) and suppressed Hallucinated Steps (6.6% vs. 11–16%), pointing to a search dynamic that truncates workflows prematurely. This is consistent with the view that weaker models cannot reliably evaluate partially generated workflows, causing local execution uncertainty to accumulate during search expansion and MCTS to terminate before reaching a complete executable workflow.

Structural errors reflect semantic over-generation, not formatting failures. Structural Errors account for 13–18% of failures but are dominated by Hallucinated Steps rather than format violations ($\leq$ 0.2% across methods), indicating that low-level formatting is rarely the primary failure source. Instead, models tend toward semantic over-generation: inserting semantically plausible but unwarranted tool calls beyond the ground-truth workflow. Per-model FPoF breakdowns are provided in Appendix I.

4.6 Reasoning-Optimized Models and Instruction Clash

Verbose intermediate reasoning does not uniformly improve executable workflow generation. In some configurations it helps; in others it actively disrupts structured output production, an effect that depends on both the planning paradigm and the model's reasoning style.

Instruction clash: when reasoning disrupts executable generation. The most striking case is DeepSeek-V3.2 under CoT, where EM collapses from 57.94% (DP) to 36.05%, a drop of 21.89pp. The FPoF distribution reveals a sharp redistribution rather than a uniform decline: under DP, only 32.7% of failures are Parameter Errors, but under CoT this share jumps to 76.5% (+43.8pp). This sharp redistribution, from tool-grounding failures under DP to parameter-binding failures under CoT, indicates that the verbose reasoning trace actively interferes with schema-consistent structured output generation rather than simply degrading tool selection. We term this **instruction clash**: CoT introduces competing generation objectives, namely free-form intermediate reasoning and schema-constrained executable workflow generation. For some models, the reasoning objective dominates, destabilizing parameter binding and dependency propagation in the final structured output.

The effect is paradigm-dependent. Reasoning-optimized variants help under CoT, where explicit reasoning aligns with the prompt's intent (Qwen3-Max-Thinking: 66.52% vs. Qwen3-Max: 62.66%), but underperform base models under DP (DeepSeek-R1: 54.08% vs. DeepSeek-V3.2: 57.94%), where extended reasoning does not improve executable grounding. Under RAP, the gap between base and reasoning variants remains small (within $\sim$2pp), suggesting that when an external search procedure orchestrates plan construction, the model's internal reasoning style has limited additional impact on execution consistency. An ablation externalizing format constraints via API-level JSON mode confirms that the collapse is caused by formatting conflict rather than degraded reasoning (Appendix J).

Implications. These results suggest that reasoning optimization does not uniformly transfer to executable workflow grounding. The conflict between verbose intermediate reasoning and schema-constrained structured generation emerges as a measurable and diagnosable failure mode, highlighting the importance of evaluating reasoning quality jointly with executable output controllability rather than through aggregate accuracy alone.

5 Conclusion

We present DisasterBench, a benchmark for evaluating executable workflow grounding in disaster-response multi-agent systems. By requiring models to generate execution-consistent plans over semantically similar but operationally distinct tools, DisasterBench exposes failure modes that aggregate metrics alone cannot reveal. We further propose First-Point-of-Failure (FPoF), a step-level diagnostic framework that separates root causes from cascaded downstream errors, enabling more precise failure attribution.

Across 14 LLMs and five planning paradigms, our evaluation reveals that performance degrades sharply with workflow depth, tool mismatch and parameter-binding failures dominate first errors, and verbose intermediate reasoning can disrupt structured output generation via instruction clash. Together, these findings suggest that strong semantic reasoning alone is insufficient for reliable execution-grounded planning, motivating future research on structured workflow coordination in safety-critical domains.

6 Limitations

DisasterBench currently evaluates workflow grounding under predefined agent specifications and benchmark settings. Future work may incorporate interactive execution feedback and more dynamic environments to study adaptive workflow planning under realistic deployment conditions. Extending the benchmark to multilingual settings is another important direction for future research.

Ethics Statement

DisasterBench is designed solely for research purposes to evaluate LLM-based planning and coordination capabilities in structured disaster-response workflows. The benchmark is constructed from publicly available research artifacts, including open-source models, published methodologies, and documented disaster-management pipelines.

The benchmark does not contain personally identifiable information (PII), private user data, or sensitive operational records. All workflow tasks and agent interfaces are abstracted at the system level and are intended for evaluation of planning consistency rather than real-world emergency deployment.

Although the benchmark focuses on disaster-response scenarios, the evaluated systems should not be interpreted as deployment-ready decision support tools. Incorrect workflow generation, parameter binding failures, or dependency propagation errors may lead to unsafe or misleading outputs in safety-critical environments. Therefore, benchmark results should be treated as diagnostic indicators of current model limitations rather than evidence of reliable autonomous disaster management capabilities.

We also acknowledge that advances in agentic planning systems may introduce broader societal risks, including over-reliance on automated decision-making and misuse in high-stakes settings without sufficient human oversight. We encourage future deployment-oriented research to incorporate domain experts, human-in-the-loop verification, and rigorous safety validation procedures.

References

Nilani Algiriyage, Raj Prasanna, Kristin Stock, Emma EH Doyle, and David Johnston. 2022. Multi-source multimodal data and deep learning for disaster response: a systematic review. *SN Computer Science*, 3(1):92.

Roberto Bentivoglio, Elvin Isufi, Sebastian Nicolaas Jonkman, and Riccardo Taormina. 2022. Deep learning methods for flood mapping: a review of existing applications and future research directions. *Hydrology and Earth System Sciences Discussions*, 2022:1–50.

Chao-Peng Chen, Jun-Wei Hsieh, Ping-Yang Chen, Yi-Kuan Hsieh, and Bor-Shiun Wang. 2023. Saras-net: Scale and relation aware siamese network for change detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14187–14195.

Zhitong Chen, Kai Yin, Xiangjue Dong, Chengkai Liu, Xiangpeng Li, Yiming Xiao, Bo Li, Junwei Ma, Ali Mostafavi, and James Caverlee. 2026. Disastqa: A comprehensive benchmark for evaluating question answering in disaster management. *arXiv preprint arXiv:2601.03670*.

Yezhen Cong, Samar Khanna, Chenlin Meng, Patrick Liu, Erik Rozi, Yutong He, Marshall Burke, David Lobell, and Stefano Ermon. 2022. Satmae: Pre-training transformers for temporal and multi-spectral satellite imagery. *Advances in Neural Information Processing Systems*, 35:197–211.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. 2021. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92.

Ritwik Gupta, Richard Hosfelt, Sandra Sajeev, Nirav Patel, Bryce Goodman, Jigar Doshi, Eric Heim, Howie Choset, and Matthew Gaston. 2019. xbd: A dataset for assessing building damage from satellite imagery. *arXiv preprint arXiv:1911.09296*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Zhi-Kai Huang, Wei-Ting Chen, Yuan-Chun Chiang, Sy-Yen Kuo, and Ming-Hsuan Yang. 2023. Counting crowds in bad weather. *arXiv preprint arXiv:2306.01209*.

Navjot Kaur, Cheng-Chun Lee, Ali Mostafavi, and Ali Mahdavi-Amiri. 2023. Large-scale building damage assessment using a novel hierarchical transformer architecture on satellite images. *Computer-Aided Civil and Infrastructure Engineering*, 38(15):2072–2091.

Arnold Kazadi, James Doss-Gollin, and Arlei Lopes Da Silva. 2024. Pluvial flood emulation with hydraulics-informed message passing. In *Forty-First International Conference on Machine Learning*.

Samar Khanna, Patrick Liu, Linqi Zhou, Chenlin Meng, Robin Rombach, Marshall Burke, David Lobell, and

Stefano Ermon. 2024. Diffusionsat: A generative foundation model for satellite imagery. In *International Conference on Learning Representations*, volume 2024, pages 5586–5604.

Kartik Kuckreja, Muhammad Sohail Danish, Muzammal Naseer, Abhijit Das, Salman Khan, and Fahad Shahbaz Khan. 2024. Geochat: Grounded large vision-language model for remote sensing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 27831–27840.

Jingtao Li, Xinyu Wang, Hengwei Zhao, Shaoyu Wang, and Yanfei Zhong. 2023a. Anomaly segmentation for high-resolution remote sensing images based on pixel descriptors. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4426–4434.

Zekun Li, Wenxuan Zhou, Yao-Yi Chiang, and Muhao Chen. 2023b. Geolm: Empowering language models for geospatially grounded language understanding. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 5227–5240.

Hanzhou Liu, Kai Yin, Zhitong Chen, Chenyue Liu, and Ali Mostafavi. 2025. Floodsql-bench: A retrieval-augmented benchmark for geospatially-grounded text-to-sql. *arXiv preprint arXiv:2512.12084*.

Wenyu Liu, Gaofeng Ren, Runsheng Yu, Shi Guo, Jianke Zhu, and Lei Zhang. 2022. Image-adaptive yolo for object detection in adverse weather conditions. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 1792–1800.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Prashant W Patil, Sunil Gupta, Santu Rana, Svetha Venkatesh, and Subrahmanyam Murala. 2023. Multi-weather image restoration via domain translation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 21696–21705.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 9895–9901.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen Jr, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 7699–7715.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022a. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zhaonan Wang, Renhe Jiang, Hao Xue, Flora D Salim, Xuan Song, and Ryosuke Shibasaki. 2022b. Event-aware multimodal mobility nowcasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4228–4236.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Peng Wu, Xuerong Zhou, Guansong Pang, Lingru Zhou, Qingsen Yan, Peng Wang, and Yanning Zhang. 2024. Vadclip: Adapting vision-language models for weakly supervised video anomaly detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 6074–6082.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Kai Yin, Jianjun Wu, Weiping Wang, Der-Horng Lee, and Yun Wei. 2023. An integrated resilience assessment model of urban transportation network: A case study of 40 cities in china. *Transportation Research Part A: Policy and Practice*, 173:103687.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Demin Yu, Xutao Li, Yunming Ye, Baoquan Zhang, Chuyao Luo, Kuai Dai, Rui Wang, and Xunlai Chen. 2024. Diffcast: A unified framework via residual diffusion for precipitation nowcasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27758–27767.

Zhuo Zheng, Yanfei Zhong, Junjue Wang, and Ailong Ma. 2020. Foreground-aware relation network for geospatial object segmentation in high spatial resolution remote sensing imagery. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4096–4105.

A Model and Tool Details

A.1 LLM Identifiers and Access Information

Table 4 lists the complete model identifiers, API providers, and access dates for all 14 LLMs evaluated in this work.

A.2 Tool Source References

Table 5 lists the source publication or repository for each of the 26 specialized tools in DisasterBench.

B Implementation Details

B.1 Shared Generation Parameters

All models are queried via their respective official APIs. The following generation parameters are shared across all planning methods unless otherwise noted (Table 6).

Temperature is overridden post-argument-parsing based on planning method: deterministic decoding (`temperature`$= 0.0$) is used for single-pass methods (DP, CoT, ReAct), while a small degree of stochasticity (`temperature`$= 0.5$) is applied to search-based methods (ToT, RAP) to enable exploration of diverse candidate trajectories.

B.2 Reproducibility and Statistical Considerations

For single-pass methods (DP, CoT, ReAct), we use deterministic decoding (`temperature`$= 0.0$), which produces identical outputs across runs for a given model and prompt. Results for these methods are therefore fully reproducible without variance. For search-based methods (ToT, RAP), stochastic sampling (`temperature`$= 0.5$) is required to enable candidate diversity; results for these methods may vary across runs. We report single-run results for all methods, which is standard practice for large-scale LLM evaluation given the substantial API cost of repeated runs across 14 models and 233 tasks. Observed performance differences between methods within the same model tier are generally large enough (often > 5pp) to be robust to run-to-run variance under these settings.

B.3 Tree-of-Thought (ToT) Configuration

We implement ToT as a two-step breadth-first search (BFS) with LLM-based pruning. The search horizon is fixed at **2 expansion steps**, which aligns with the average solution depth of the benchmark (2.53 steps). At each step, candidate continuations are generated in parallel from all active prefixes, scored by an LLM critic via majority voting, and pruned to retain only the top-scoring prefix before the next expansion (Table 7).

The effective behaviour is a *wide-expansion, narrow-retention* search: at each step, up to $7 \times |\text{active prefixes}|$ candidates are generated, each evaluated by 5 critic samples, and only the single highest-scoring prefix is carried forward. This design prioritises exploration at each step while controlling computational cost.

B.4 Reasoning via Planning (RAP) Configuration

RAP formulates plan generation as Monte Carlo Tree Search (MCTS) over the tool pool. We use UCT-based node selection with a prior, aggregating child rewards via mean and propagating them upward via max-child backpropagation. The combined reward at each node is $R = r_0^{\alpha} \cdot r_1^{1-\alpha}$, where r_0 is the usefulness reward and r_1 is the self-consistency reward estimated by repeated sampling (Table 8).

Per-model configurations. For computationally expensive frontier models (e.g., `google/gemini-3.1-pro-preview`), we reduced `num_rollouts` and sampling counts to manage API cost while preserving the qualitative search behaviour. Exact per-model configurations are recorded in `results/<model>/rap/config.json` and are included in the released codebase.

Note on `w_exp` vs. `mcts_exploration_weight`. Some `config.json` files contain a field named `mcts_exploration_weight` (e.g., 2.0); this field does *not* drive UCT exploration in our implementation—the effective exploration weight is controlled exclusively by the `w_exp` argument passed directly to the `MCTS` constructor.

B.5 Structured Output Enforcement

All planning methods are prompted to produce output in a fixed JSON schema specifying `step`, `agent`, `inputs`, `outputs`, `dependency`, and `dependency_content` fields. We deliberately do *not* apply API-level schema enforcement (e.g., OpenAI function-calling, strict JSON mode, or post-hoc repair) beyond the prompt-level specification. Outputs that cannot be parsed into the expected schema are treated as incorrect predictions.

This design choice is intentional: API-level enforcement constrains the generation space in ways

Model (Paper Name)	API Identifier	Provider	Accessed
Gemini 3.1 Pro Preview	`google/gemini-3.1-pro-preview`	Google	Apr. 2026
GPT-5.4	`gpt-5.4`	OpenAI	Apr. 2026
DeepSeek-V3.2	`deepseek-ai/deepseek-v3.2`	DeepSeek	Apr. 2026
DeepSeek-R1	`deepseek-ai/deepseek-r1`	DeepSeek	Apr. 2026
Qwen3-Max	`qwen/qwen3-max`	Alibaba	Apr. 2026
Qwen3-Max-Thinking	`qwen/qwen3-max-thinking`	Alibaba	Apr. 2026
Qwen3.5-27B	`qwen/qwen3.5-27b`	Alibaba	Apr. 2026
Gemma-4-31B	`google/gemma-4-31b`	Google	Apr. 2026
Llama-3.3-70B	`meta-llama/llama-3.3-70b`	Meta	Apr. 2026
Qwen3.5-9B	`qwen/qwen3.5-9b`	Alibaba	Mar. 2026
Llama-3.1-8B	`meta-llama/llama-3.1-8b`	Meta	Apr. 2026
Ministral-14B	`mistralai/ministral-14b`	Mistral	Apr. 2026
Ministral-8B	`mistralai/ministral-8b`	Mistral	Apr. 2026
Ministral-3B	`mistralai/ministral-3b`	Mistral	Apr. 2026

Table 4: Complete model identifiers, API providers, and access dates for all 14 evaluated LLMs. All models are accessed via OpenRouter.

that may artificially inflate performance, conflating scaffolding capability with intrinsic planning ability. Investigating the effect of schema enforcement on the instruction clash phenomenon—and whether it mitigates CoT-induced formatting failures—is a promising direction for future work.

B.6 Token Budget and Computational Cost

Our current implementation logs model outputs but does not instrument `completion.usage` fields (prompt/completion token counts) in stored prediction files. We acknowledge this as a limitation for strict cost-controlled comparisons across methods, and plan to include token logging in the public release.

As a qualitative guide, the five methods differ substantially in API call count per task (Table 9):

These figures reflect the default configurations; actual counts vary with per-model overrides and early termination in MCTS.

B.7 Code and Data Release

The full benchmark—including the agent DAG (`graph_desc.json`), typed edge schema, 233 task JSON files with ground-truth plans, evaluation code (including the FPoF analyzer and partial credit scorer), prompts for all five planning methods, and per-model `config.json` files—will be released publicly upon acceptance. Reproducibility scripts for all reported experiments will be included. We are committed to enabling full reproduction of every result reported in this paper.

C Complete Agent DAG Edge Table

Table 10 lists all 81 directed typed edges in the DisasterBench agent DAG, organized by subgraph family. To keep the table compact, edge types are abbreviated as M→D (`model_to_data`) and D→M (`data_to_model`). Fan-in edges list multiple source data nodes in brackets.

D Dataset Statistics

Table 14 summarizes the structural composition of the 233 benchmark tasks. Tasks are categorized into three types following the taxonomy of Shen et al. (2024): *Node* (single-step), *Chain* (sequential multi-step), and *Branching* (one-to-many dependency). Compositional difficulty increases across types: Node tasks test tool grounding in isolation, while Chain and Branching tasks require parameter propagation and dependency reasoning across multiple steps.

Tool coverage. The 35 Node tasks span 21 distinct tools. To assess robustness to surface-level variation independently of compositional reasoning, 12 tools are instantiated across 2–3 task variants with distinct natural-language descriptions and input parameterizations. Table 16 (Appendix E) reports the full per-tool invocation frequency across all 233 tasks.

Plan depth distribution. Figure 2 plots exact-match accuracy as a function of plan depth. The majority of tasks fall at depths 2–4 ($n = 167$), with three tasks at depths ≥ 5 (depths 5, 6, and 9). These deeper tasks are aggregated into a single bucket in the depth analysis due to limited per-bucket sample size.

Tool	Source
Temporal_High_Resolution_Image_Generation	Khanna et al. (2024)
High-Resolution_Image_Reconstructor	Khanna et al. (2024)
Metadata_and_Text_Prompt_Image_Generation	Khanna et al. (2024)
Weather_Degraded_Image_Restoration	Patil et al. (2023)
Temporal_Image_Sequence_Classifier	Cong et al. (2022)
RGB_GeoImage_Classifier	Cong et al. (2022)
Multi_Spectral_Classifier	Cong et al. (2022)
Anomaly_Detection_Forest	Li et al. (2023a)
Landslide_Segmentation	Li et al. (2023a)
Urban_Anomaly_Detection	Li et al. (2023a)
Change_Mapping_and_Detection	Chen et al. (2023)
Foggy_Scenario_Object_Detection	Liu et al. (2022)
Low-Light_Object_Detection	Liu et al. (2022)
Geospatial_Object_Segmentation	Zheng et al. (2020)
Crowd_Counting_in_Adverse_Weather	Huang et al. (2023)
Building_damage_assessment	Kaur et al. (2023)
Precipitation_Nowcasting	Yu et al. (2024)
Flood_depth_prediction	Kazadi et al. (2024)
depth_speed_model	Yin et al. (2023)
Toponym_Detection	Li et al. (2023b)
Event_detection	Li et al. (2023a)
Video_anomaly_detection	Wu et al. (2024)
Post_Disaster_Mobility_Recovery	Wang et al. (2022b)
Multimodal_mobility_prediction_under_events	Wang et al. (2022b)
GeoChat	Kuckreja et al. (2024)
precipitation_data_convert_tool	Yu et al. (2024)

Table 5: Source publications and repositories for all 26 tools in DisasterBench.

Parameter	Value	Notes
max_tokens	8,192	Per API call
top_p	0.95	Nucleus sampling
top_k	20	Where supported by API
temperature	0.0	DP, CoT, ReAct (deterministic)
temperature	0.5	ToT, RAP (stochastic search)

Table 6: Shared generation hyperparameters across all models and methods.

Parameter	Value	Description
Branching factor	7	Candidate continuations generated per active prefix per step
Critic samples	5	Number of LLM critic calls per candidate for majority-vote scoring
Beam width	1	Number of prefixes retained after pruning at each step
Search depth	2	Number of BFS expansion steps (hardcoded)
Temperature	0.5	Applied to both generation and critic sampling

Table 7: ToT hyperparameters (CLI defaults).

Parameter	Value	Description
Rollouts (n_{rollout})	5	Number of MCTS simulation rollouts
Max tree depth	5	Maximum expansion depth (steps)
Sub-question samples (n_{subq})	16	Candidate sub-steps sampled per node expansion
Confidence samples (n_{conf})	32	Repeated samples for self-consistency reward r_1
UCT exploration weight (w)	1.0	w in UCT score $\bar{r} + w\sqrt{\ln N_p / N}$
Reward exponent (α)	0.5	Balances usefulness and self-consistency rewards
Default r_1	1.0	Fallback self-consistency reward when sampling fails
Reward aggregation	mean	Child reward aggregation in backpropagation
Temperature	0.5	Applied to all MCTS generation and sampling calls

Table 8: RAP / MCTS hyperparameters (CLI defaults; per-model overrides recorded in results/<model>/rap/config.json).

Random baseline bound. As a sanity check on benchmark non-triviality, the probability of matching a ground-truth tool sequence by uniform random selection from $N = 26$ tools over a plan of length L is N^{-L}. At the benchmark's average depth $\bar{L} = 2.53$, this yields $26^{-2.53} \approx 2.6 \times 10^{-4}$, confirming that correct planning cannot be achieved by chance. All structural baselines empirically achieve 0% EM (Section 4.1).

D.1 Tool Semantic Overlap Analysis

To verify that benchmark difficulty arises from semantic ambiguity among tools rather than from pool size alone, we compute pairwise cosine similarity between tool descriptions using two sentence encoders: all-MiniLM-L6-v2 and intfloat/e5-small-v2. tool descriptions are the natural-language functional specifications provided to models during evaluation.

Table 15 reports similarity statistics across all $\binom{26}{2} = 325$ tool pairs. The two encoders reveal

Method	Approximate API calls per task
Direct Prompting (DP)	1
Chain-of-Thought (CoT)	1
ReAct	$\approx D$ (average $D = 2.53$ steps)
Tree-of-Thought (ToT)	$\approx 2 \times (n_{\text{gen}} + n_{\text{eval}}) = 2 \times (7 + 5) = 24$
Reasoning via Planning (RAP)	$\approx n_{\text{rollout}} \times n_{\text{subq}} + n_{\text{conf}} \approx 50$–$200$

Table 9: Approximate API call count per task for each planning method.

complementary perspectives: under MiniLM, 18.5% of pairs exceed cosine 0.50 and only 0.6% exceed 0.80, reflecting a conservative similarity distribution; under E5-small, all 325 pairs exceed 0.77 and 92.9% exceed 0.80, indicating that tool descriptions are uniformly close in the E5 embedding space. Both encoders agree that the most semantically similar pairs are operationally distinct tools: `Temporal_Image_Sequence_Classifier` ↔ `RGB_GeoImage_Classifier` (MiniLM: 0.855; E5: 0.965), `Anomaly_Detection_Forest` ↔ `Urban_Anomaly_Detection` (MiniLM: 0.804; E5: 0.933), and `Foggy_Scenario_Object_Detection` ↔ `Low-Light_Object_Detection` (MiniLM: 0.783; E5: 0.952). These pairs appear semantically interchangeable in natural language yet differ substantially in operational input requirements and downstream compatibility.

These results confirm that the 26-tool pool exhibits substantial description-level semantic overlap despite operational divergence. The uniformly high E5 similarities (≥ 0.77 for all pairs) further suggest that surface-level semantic matching alone is insufficient for correct tool grounding, models must reason about operational compatibility and downstream dependency constraints beyond description similarity.

E Tool Invocation Frequency

To verify balanced task coverage, we report the number of tasks in which each tool appears at least once in the ground-truth plan (Table 16). No single tool dominates the distribution: `GeoChat` is the most frequent at 54 tasks (23.2%), while 18 of the 26 tools each appear in fewer than 20% of tasks, confirming that the benchmark exercises the full tool pool.

F Evaluation Details

This appendix provides the formal definitions of the evaluation metrics introduced in Section 3.4, including the partial-credit diagnostic metrics, the First-Point-of-Failure (FPoF) error taxonomy, and the implication structure among metrics.

F.1 Partial-Credit Diagnostic Metrics

All four metrics are task-level booleans: each returns 1 if the corresponding condition holds for every step in the predicted plan, and 0 otherwise. Dataset-level scores are obtained by averaging over all 233 tasks (i.e., the fraction of tasks for which the condition holds).

Let $\hat{P} = (\hat{s}_1, \ldots, \hat{s}_{\hat{T}})$ denote the predicted plan and $P = (s_1, \ldots, s_T)$ the ground-truth plan. We first require $\hat{T} = T$ (plan lengths match); if not, all four metrics return 0. When $\hat{T} = T$, each step s_t is a tuple $(\texttt{step}_t, \texttt{agent}_t, \texttt{inputs}_t, \texttt{outputs}_t, \texttt{dep}_t, \texttt{dep_content}_t)$, and the metrics are defined as follows.

Tool Accuracy (A). All step indices and agent identities match:

$$\mathrm{A}(\hat{P}, P) = \mathbb{I}\big(\forall t : \hat{\texttt{step}}_t = \texttt{step}_t \ \wedge \ \hat{\texttt{agent}}_t = \texttt{agent}_t\big)$$

Parameter Accuracy. tool conditions hold, and inputs and outputs match at every step:

$$\mathrm{Param}(\hat{P}, P) = \mathbb{I}\Big(\mathrm{A} = 1 \ \wedge \ \forall t : \hat{\texttt{inputs}}_t = \texttt{inputs}_t \ \wedge \ \hat{\texttt{outputs}}_t = \texttt{outputs}_t\Big)$$

Dependency Accuracy. tool conditions hold, and dependency structure and dependency content match at every step:

$$\mathrm{Dep}(\hat{P}, P) = \mathbb{I}\Big(\mathrm{A} = 1 \ \wedge \ \forall t : \mathrm{sort}(\hat{\texttt{dep}}_t) = \mathrm{sort}(\texttt{dep}_t) \ \wedge \ \widehat{\texttt{dep_content}}_t \doteq \texttt{dep_content}_t\Big)$$

where $\mathrm{sort}(\cdot)$ normalizes dependency lists to sorted order (allowing -1 vs. $[-1]$ equivalence), and $\doteq$ denotes deep equality with both-null treated as equal.

Exact Match (EM). EM is computed by exact structural match over all step fields under the canonical step schema; equivalently, EM $= 1$ if and only if both Parameter Accuracy and Dependency Accuracy equal 1:

$$\mathrm{EM}(\hat{P}, P) = \mathbb{I}(\mathrm{Param} = 1 \ \wedge \ \mathrm{Dep} = 1)$$

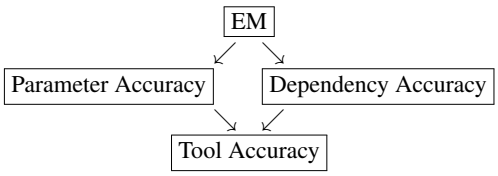
F.2 Semantic-Execution Gap: Full Results

Table 17 reports Tool Accuracy (A) and exact-match accuracy (EM) for all 14 models across five planning methods. The gap between A and EM—the **semantic-execution gap**—is present for every model under every method, confirming that

semantically plausible tool selection is consistently easier than execution-consistent workflow generation. These results support the analysis in Section 4.3; notable cases discussed in the main text include Llama-3.3-70B under DP (22.31pp gap) and DeepSeek-V3.2 under CoT, where both A and EM drop uniformly relative to its DP baseline.

F.3 Metric Implication Structure

The four metrics form a partial order under logical implication. An arrow $X \Rightarrow Y$ means that $X = 1$ logically guarantees $Y = 1$ for every task.



Formally:

- Param $= 1 \;\Rightarrow\;$ A $= 1$ (parameter check subsumes step and tool matching).
- Dep $= 1 \;\Rightarrow\;$ A $= 1$ (dependency check subsumes step and tool matching).
- EM $= 1 \;\Rightarrow\;$ Param $= 1 \wedge$ Dep $= 1 \wedge$ A $= 1$.

Crucially, Parameter Accuracy and Dependency Accuracy are *not* mutually comparable: a task can satisfy one without the other, because they check disjoint subsets of step attributes (inputs/outputs vs. dependency structure/content). This diamond structure enables independent diagnosis of parameter binding failures and dependency reasoning failures on the same dataset.

F.4 FPoF Error Taxonomy

The First-Point-of-Failure (FPoF) analyzer examines each failed prediction and identifies the earliest step at which the predicted plan diverges from the ground truth. The error at this step is classified into one of eight fine-grained types (left column of Table 18), which we aggregate into three top-level categories for the analysis in Section 4.5.

Classification procedure. For a failed prediction $\hat{P}$ with ground truth P:

1. **Structural pre-check.** If $\hat{P}$ is empty (no model output), return `empty_output`. If $\hat{P}$ cannot be parsed as a JSON list, return `format_error`.
2. **Step-level scan.** Let $L = \min(\hat{T}, T)$. For each aligned step $t = 0, \ldots, L-1$, check in order:
   (a) Step index mismatch ($\hat{\texttt{step}}_t \neq \texttt{step}_t$): return `parameter_error` at step t.
   (b) tool mismatch ($\hat{\texttt{agent}}_t \neq \texttt{agent}_t$): return `agent_mismatch` at step t.
   (c) Inputs or outputs mismatch: return `parameter_error` at step t.
   (d) Dependency list mismatch (after sorting): return `dependency_error` at step t.
   (e) Dependency content mismatch: return `dependency_content_error` at step t.
3. **Length mismatch (only if all aligned steps match).** If $\hat{T} < T$, return `early_stop`. If $\hat{T} > T$, return `hallucinated_extra_steps`. Otherwise ($\hat{T} = T$ and all steps match), the prediction is correct.

This priority order ensures that more fundamental errors (wrong agent) are reported before finer-grained ones (wrong parameters, wrong dependencies). Length mismatch is attributed only when all overlapping steps are correct—capturing plans that are structurally valid up to the point of premature termination or over-extension. By reporting only the *first* point of failure, FPoF isolates root causes from cascading downstream errors that typically follow once a plan has diverged from the ground-truth workflow.

Codebase correspondence. The fine-grained error types correspond directly to the return values of `Evaluator.analyze_error_propagation()` in `evaluators/evaluators.py`. The field `tools_correct` in the released prediction files corresponds to Tool Accuracy as defined above.

G Datasheet for DisasterBench

Following the datasheet framework of (Gebru et al., 2021), we provide the following documentation for DisasterBench.

Motivation. DisasterBench was created to evaluate the ability of LLMs to coordinate multi-step agent workflows in disaster management, a setting where incorrect parameter bindings or dependency propagation can invalidate downstream workflow execution. The benchmark is intended for academic research on workflow grounding and structured multi-agent planning.

Composition. The dataset consists of 233 planning tasks, each comprising a natural-language task description and a structured ground-truth plan in JSON format specifying tool calls, input parameters, dependency structure, and dependency content. Tasks are derived from four disaster-response workflow categories: temporal remote sensing analysis, image reconstruction, adverse-weather perception, and hydrological modeling. No personally identifiable information is included.

Collection process. Tasks were generated through a two-stage pipeline. First, GPT-4o (`gpt-4o`, accessed January–March 2025) was used to generate naturalistic task descriptions from sampled workflow structures. Second, all generated tasks were manually verified through two rounds of expert review by annotators with domain familiarity in remote sensing and disaster management. The workflow dependency structure was expert-curated to ensure semantic and technical validity of all agent dependencies.

Preprocessing / cleaning / labeling. Ground-truth workflows were verified for semantic consistency between task descriptions and target plans, as well as strict adherence to agent input/output schemas. Ambiguous cases were resolved through group discussion, and unresolved tasks were discarded.

Uses. DisasterBench is intended for evaluating workflow grounding and structured planning capabilities of LLMs under executable dependency constraints. The benchmark should not be used for training without careful consideration of potential benchmark contamination. The disaster-management setting is intended as a research testbed and does not constitute operational guidance for real-world disaster response.

Distribution. The dataset and evaluation code will be publicly released upon acceptance under an open-source license.

Maintenance. The dataset will be maintained by the authors. Issues, corrections, and future updates will be managed through the public project repository.

Known limitations. Most tasks in DisasterBench are linear workflows, which may under-represent highly parallel or data-merging coordination patterns. Evaluation is plan-level only and does not include execution-grounded runtime verification. In addition, the tool pool reflects a curated subset of disaster-response capabilities and may not cover all real-world workflow configurations.

H Prompt Templates

This appendix provides the prompt templates used throughout the benchmark construction and evaluation pipeline. Section H.1 covers the task generation prompt used during benchmark construction (Section 3.3), Section H.2 covers the agent filtering prompt (Section 3.1), and Sections H.3–H.8 cover the five planning method prompts used during evaluation (Section 4.1). All planning method prompts share the same agent description block (`{agents_desc}`), which contains the full input/output schema for all 26 tools. The workflow structure is *not* included in any prompt; models must infer valid compositions from agent specifications alone. For brevity, the shared JSON output schema is shown once (Section H.3) and omitted from subsequent templates. Complete prompt texts including all few-shot examples are provided in the released codebase.

H.1 Task Description Generation (Self-Instruct)

During benchmark construction (Section 3.3), we use GPT-4o to generate naturalistic task descriptions from sampled ground-truth plans via the following prompt:

```
You are an expert technical writer specializing in
     disaster
management scenarios. You will be given a structured
      agent
execution plan (in JSON format) that describes a
     sequence
of AI agents, their inputs, outputs, and inter-step
dependencies for a disaster-response workflow.

Your task is to write a realistic, goal-oriented
     task
description that a disaster management practitioner
     might
submit to an AI planning system. The description
     should:

1. Describe the user's high-level goal and the
     disaster
   scenario context (e.g., flood assessment, post-
       earthquake
   damage analysis, storm impact evaluation).
2. Mention the specific input data files that the
     user
   would provide (using the file paths from the plan
       's
   first step inputs).
3. Be written in natural language as a single
     coherent
   paragraph.

CRITICAL CONSTRAINTS:
- Do NOT mention any tool names, model names, or
     tool
  identifiers.
```

```
- Do NOT expose the number of steps, the plan
    structure,
  or the sequential ordering of operations.
- Do NOT reference intermediate data types, output
    keys,
  or dependency relationships.
- The description should read as if written by a
    domain
  practitioner who knows what outcome they want but
      not
  how to achieve it technically.

Input plan:
{ground_truth_plan_json}

Write the task description below:
```

All generated descriptions undergo two rounds of expert review to ensure that no structural cues are leaked, as described in Section 3.3.

H.2 Agent Filtering

During tool pool construction (Section 3.1), candidate models are screened for inclusion using the following prompt:

```
You are an expert in AI for disaster management and
    remote
sensing. You will be given information about a
    publicly
available AI model (title, abstract, and repository
    link).
Your task is to determine whether this model is
    suitable
for inclusion in a disaster-response agent benchmark
    .

A model is suitable if ALL of the following criteria
     are
met:

1. DOMAIN RELEVANCE: The model addresses a task
    directly
   applicable to disaster response workflows, such
       as:
   remote sensing image analysis, damage assessment,
   flood/precipitation modeling, anomaly detection,
       change
   detection, crowd monitoring, object detection
       under
   adverse conditions, geospatial reasoning, or
       disaster-
   related text/event processing.

2. TYPED INTERFACE: The model has clearly defined
    input and
   output types (e.g., takes a satellite image and
       outputs
   a segmentation map), enabling it to be composed
       with
   other models in a typed pipeline.

3. IMPLEMENTATION AVAILABILITY: The model has a
    publicly
   accessible codebase or pre-trained weights that
       can be
   used for inference.

For each candidate, respond with:
- Decision: INCLUDE or EXCLUDE
- Justification: One sentence explaining why.

Candidate model:
Title: {paper_title}
Abstract: {paper_abstract}
Repository: {repo_url}
```

Candidates passing the automated screen are further verified manually for implementation quality and interface compatibility, as described in Section 3.1.

H.3 Shared Output Schema (Planning Methods)

All five planning methods instruct the model to produce a JSON array conforming to the following schema. Each element represents one planning step:

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "agent":   {"type": "string"},
      "step":    {"type": "integer"},
      "dependence": {
        "type": "array",
        "items": {"type": "integer"}
      },
      "dependence_content": {
        "oneOf": [{"type": "null"},
                  {"type": "object"}]
      },
      "inputs":  {"type": "object"},
      "outputs": {"type": "array",
                  "items": {"type": "string"}}
    },
    "required": ["agent", "step", "dependence",
      "dependence_content", "inputs", "outputs"]
  }
}
```

Key conventions shared across all methods:

- `dependence`: `[-1]` if no dependency on prior steps; otherwise a single step index (e.g., `[0]`).
- `dependence_content`: maps the depended step index to a list of output keys consumed, or `null` if independent.
- Generated inputs use the placeholder format `<GENERATED>-{step}-<{output_key}>`.

H.4 Direct Prompting (DP)

DP uses no intermediate reasoning. The system prompt instructs the model to output *only* the structured plan with no explanation.

System prompt. The system prompt specifies the output format and agent usage rules, and appends the full agent description block:

```
You are a task planning expert agent specializing in
disaster response automation. Your goal is to
    transform a
high-level disaster management task into a
    structured JSON
plan. You MUST output ONLY the final structured plan
    , with
no reasoning and no extra text.

STRICT OUTPUT FORMAT (must follow):
- Output exactly one line that starts with:
  The structured task plan is:
- After that prefix, output a single valid JSON
    ARRAY only.
- Use double quotes for JSON keys and string values.
```

```
- Do not output any other text before or after the
    JSON.

IMPORTANT RULES:
- Choose the minimum set of agents required.
- Each step index is 0-based and sequential.
- Each step may depend on at most ONE previous step.
- Use dependence [-1] when all inputs come from the
    user.
- For generated inputs, use:
  "<GENERATED>-<STEP>-<OutputKey>"

Agents and Input Output Details:
{agents_desc}

[Shared output schema inserted here]
```

User prompt. The user prompt provides the task description and elicits the plan directly without intermediate reasoning:

```
Instruction: {task_desc}

Response:
```

No few-shot examples are provided for DP.

H.5 Chain-of-Thought (CoT)

CoT elicits intermediate reasoning before plan generation.

System prompt. The system prompt is identical to DP except: (1) the no-reasoning instruction is replaced with *"Add 'The structured task plan is:' then followed by a structured JSON at the end of your response"*, and (2) two few-shot examples are appended demonstrating step-by-step reasoning followed by a valid plan.

User prompt. The user prompt appends a CoT trigger suffix to elicit step-by-step reasoning before plan generation:

```
Instruction: {task_desc}

Response: Let's think step by step.
```

The suffix `Let's think step by step.` serves as the CoT trigger.

Few-shot examples. Two examples are provided: (1) a two-step chain (`Urban_Anomaly_Detection` → `GeoChat`) and (2) a four-step branching plan (`Metadata_and_Text_Prompt_Image_Generation` → `Weather_Degraded_Image_Restoration` → `Crowd_Counting` + `Low-Light_Object_Detection`). Each example includes a complete reasoning trace followed by the structured JSON output.

H.6 Tree-of-Thought (ToT)

ToT uses two separate prompts: a *generation prompt* for the actor LLM that proposes candidate plans, and a *vote prompt* for the critic LLM that selects the best candidate.

Generation prompt (actor). The generation prompt follows the same structure as CoT (system prompt with tool descriptions, shared output schema, and two few-shot examples). Multiple candidate plans are sampled at `temperature`$= 0.5$ (see Appendix B for the branching factor and search depth configuration).

Vote prompt (critic). The critic receives all candidate plans and selects the best one:

```
You are a task planning critic agent specializing in
rating the plans generated by an actor agent. Your
    task
is to choose the best planning based on the provided
choices.

You must check the following:
[Shared output schema and field definitions inserted
     here]

At the end, reply with exactly one line in this
    exact
format (no other text before or after):
The best choice is: <BEST_OPTION>
where <BEST_OPTION> is an integer from 1 to the
    number
of choices listed above.
```

H.7 Reasoning via Planning (RAP)

RAP uses two prompt types within the MCTS loop: a *sub-question/sub-answer* prompt for node expansion, and a *usefulness* prompt for reward estimation.

Sub-question / sub-answer prompt. This prompt instructs the model to decompose the task into sub-questions, answering each with a partial plan. When sufficient context has been accumulated, the model begins the final sub-question with `Now we can answer the question:` followed by the consolidated plan.

```
You are an expert disaster management AI agent
    tasked
with breaking down disaster management queries into
simpler, actionable questions. You will build the
    answer
in multiple steps, where in each step, you generate
    a
question based on the user question and previous
    answers.
When all required subquestions are answered, start
    the
next question with "Now we can answer the question:"
followed by the question.

[Shared output schema inserted here]

Agents description:
{agents_list}

[Three few-shot examples demonstrating sub-question
decomposition, each showing 2-3 sub-questions
    leading
to a final consolidated plan]
```

Usefulness prompt. The usefulness prompt evaluates whether a candidate sub-question is relevant to the original task. It receives the original question, prior sub-questions, and the candidate, then outputs `Yes` or `No` with a one-sentence justification. Three few-shot examples are provided demonstrating both useful and non-useful sub-questions.

H.8 ReAct

ReAct interleaves reasoning traces (`Thought`), action decisions (`Action`), and state observations (`Observation`) before producing the final plan.

System prompt. The system prompt specifies the Thought/Action/Observation reasoning format and appends the full agent description block:

```
You are a task planning expert agent specializing in
disaster response automation.

You MUST reason using the following format:

Thought: [analyze what tools are needed and why]
Action: [decide the next step: which agent to select
     ]
Observation: [what this step produces and what the
     next
             step needs]
... (repeat until all steps are planned)

Then output the final structured plan on ONE line:
The structured task plan is: [JSON array]

[Shared output schema and rules inserted here]

Agents and Input Output Details:
{agents_desc}
```

User prompt. The user prompt instructs the model to apply the reasoning format before producing the final structured plan:

```
Instruction: {task_desc}

Use Thought/Action/Observation to reason through the
      plan
step by step, then output the final structured task
     plan.
Response:
```

Few-shot examples. Four examples are provided covering: (1) a two-step chain (image restoration → object detection), (2) a three-step chain (precipitation nowcasting → data conversion → flood prediction), (3) a two-step chain (anomaly detection → contextual description), and (4) a five-step branching plan with fan-out (image generation → parallel forest and urban anomaly detection → two separate GeoChat descriptions). Each example demonstrates the full Thought/Action/Observation trace followed by the structured JSON output.

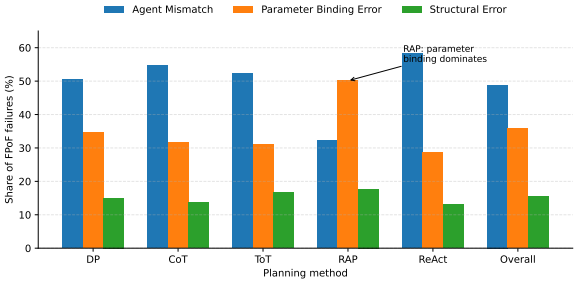


Figure 3: Distribution of First-Point-of-Failure (FPoF) error types across all 14 models and 5 planning methods, aggregated into the three top-level categories defined in Section 3.4: tool mismatch (48.8%), Parameter Binding Error (35.8%), and Structural Error (15.4%). RAP's distinctive early-termination signature (11.1% of RAP failures) is visible as an elevated Structural Error share.

I Per-Model FPoF Breakdowns

Table 19 reports the First-Point-of-Failure (FPoF) distribution for each of the 14 models under all five planning methods. For each model–method combination, we show the number of perfectly matched tasks (out of 233) and the top three most frequent FPoF error types with counts and percentages of total tasks. Error types follow the fine-grained taxonomy defined in Appendix F.4.

J Instruction Clash Ablation

To verify that the DeepSeek-V3.2 CoT collapse reflects instruction clash rather than degraded reasoning capability, we run CoT with API-level JSON mode enabled (n=1), which enforces structural validity externally without modifying the reasoning prompt. EM recovers from 36.05% to 62.23%, surpassing DP (57.94%), while the dominant failure mode shifts from parameter_error (48.9% of tasks) back to agent_mismatch (20.6%), closely mirroring the DP profile (13.7% parameter error). This confirms that the collapse is caused by competing generation objectives rather than insufficient reasoning: when the formatting burden is externalized, CoT's latent reasoning quality is preserved. The no-visible-reasoning condition is instantiated by the DP baseline by design, which outputs only the final structured plan with no intermediate reasoning.

K Structural Baseline Details

We include four non-LLM baselines to isolate the contribution of language-grounded workflow generation. *Random* selects tools uniformly at random (0% EM). *Lexical-Greedy* selects tools by TF-

IDF similarity to the task description (5.58% Tool Accuracy, 0% EM), confirming that surface-level matching is insufficient for executable workflow construction. *Shortest Path* is given the gold plan's start and end tools plus plan length, yet achieves only 51.07% Tool Accuracy and 0% EM, showing that structural hints alone cannot recover executable workflows. *Oracle-Random* receives the complete ground-truth tool sequence but assigns parameters randomly, achieving 100% Tool Accuracy yet 0% EM—isolating parameter binding as an independent execution bottleneck even when tool selection is perfect.

G.	Source	Type	Target
Subgraph I: Temporal High-Resolution Analysis (33 edges)			
I	MetadataPromptImageGeneration	M→D	ThreeImagesSameLocation
I	ThreeImagesSameLocation	D→M	TemporalHRImageGeneration
I	TemporalHRImageGeneration	M→D	HRImagesBeforeandAfterDisaster
I	HRImagesBeforeandAfterDisaster	D→M	AnomalyDetectionForest
I	AnomalyDetectionForest	M→D	AnomalyMapForest
I	AnomalyMapForest	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	LandslideSegmentation
I	LandslideSegmentation	M→D	LandslideMap
I	LandslideMap	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	UrbanAnomalyDetection
I	UrbanAnomalyDetection	M→D	AnomalyMapUrban
I	AnomalyMapUrban	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	ChangeMappingandDetection
I	ChangeMappingandDetection	M→D	ChangeMap
I	ChangeMap	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	BuildingDamageAssessment
I	BuildingDamageAssessment	M→D	DamageClassificationMap
I	DamageClassificationMap	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	TemporalImageSequenceClassifier
I	TemporalImageSequenceClassifier	M→D	TemporalClassificationResults
I	TemporalClassificationResults	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	GeospatialObjectSegmentation
I	GeospatialObjectSegmentation	M→D	SegmentationMap
I	SegmentationMap	D→M	GeoChat
I	HRImagesBeforeandAfterDisaster	D→M	RGBGeoImageClassifier
I	RGBGeoImageClassifier	M→D	PredictedCategory
I	PredictedCategory	D→M	GeoChat
I	GeoChat	M→D	ContextualDescription
I	[PredictedCategory + SegmentationMap]	D→M	GeoChat
I	[ChangeMap + DamageClassificationMap]	D→M	GeoChat
I	[DamageClassificationMap + TemporalClassificationResults]	D→M	GeoChat
I	[DamageClassificationMap + TemporalClassificationResults + ChangeMap]	D→M	GeoChat

Table 10: Typed edges in the DisasterBench DAG by subgraph.

G.	Source	Type	Target
Subgraph II: Image Reconstruction (23 edges)			
II	MetadataPromptImageGeneration	M→D	LowResolutionMulti-SpectralImage
II	LowResolutionMulti-SpectralImage	D→M	HighResolutionImageReconstructor
II	HighResolutionImageReconstructor	M→D	HighResolutionImage
II	HighResolutionImage	D→M	GeospatialObjectSegmentation
II	GeospatialObjectSegmentation	M→D	SegmentationMap
II	SegmentationMap	D→M	GeoChat
II	HighResolutionImage	D→M	Multi-SpectralClassifier
II	Multi-SpectralClassifier	M→D	ImageClassificationCategory
II	ImageClassificationCategory	D→M	GeoChat
II	HighResolutionImage	D→M	AnomalyDetectionForest
II	AnomalyDetectionForest	M→D	AnomalyMapForest
II	AnomalyMapForest	D→M	GeoChat
II	HighResolutionImage	D→M	LandslideSegmentation
II	LandslideSegmentation	M→D	LandslideMap
II	LandslideMap	D→M	GeoChat
II	HighResolutionImage	D→M	UrbanAnomalyDetection
II	UrbanAnomalyDetection	M→D	AnomalyMapUrban
II	AnomalyMapUrban	D→M	GeoChat
II	GeoChat	M→D	ContextualDescription
II	[AnomalyMapUrban + LandslideMap]	D→M	GeoChat
II	[LandslideMap + AnomalyMapForest]	D→M	GeoChat
II	[LandslideMap + AnomalyMapUrban]	D→M	GeoChat
II	[LandslideMap + AnomalyMapUrban + AnomalyMapForest]	D→M	GeoChat

Table 11: Typed edges in the DisasterBench DAG by subgraph (Part II).

G.	Source	Type	Target
Subgraph III: Adverse-Weather Perception (18 edges)			
III	MetadataPromptImageGeneration	M→D	HighResolutionImage
III	HighResolutionImage	D→M	FoggyScenarioObjectDetection
III	FoggyScenarioObjectDetection	M→D	DetectedObjectsFoggy
III	DetectedObjectsFoggy	D→M	GeoChat
III	GeoChat	M→D	ContextualDescription
III	HighResolutionImage	D→M	Low-LightObjectDetection
III	Low-LightObjectDetection	M→D	DetectedObjectsLowLight
III	HighResolutionImage	D→M	CrowdCountinginAdverseWeather
III	CrowdCountinginAdverseWeather	M→D	CountedCrowd
III	HighResolutionImage	D→M	WeatherDegradedImageRestoration
III	WeatherDegradedImageRestoration	M→D	RestoredImage
III	RestoredImage	D→M	FoggyScenarioObjectDetection
III	FoggyScenarioObjectDetection	M→D	DetectedObjectsFoggy
III	DetectedObjectsFoggy	D→M	GeoChat
III	RestoredImage	D→M	Low-LightObjectDetection
III	Low-LightObjectDetection	M→D	DetectedObjectsLowLight
III	RestoredImage	D→M	CrowdCountinginAdverseWeather
III	CrowdCountinginAdverseWeather	M→D	CountedCrowd

Table 12: Typed edges in the DisasterBench DAG by subgraph (Part III).

G.	Source	Type	Target
Subgraph IV: Hydrological Modeling (7 edges)			
IV	PrecipitationNowcasting	M→D	PredictedPrecipitation
IV	PredictedPrecipitation	D→M	PrecipitationDataConversionTool
IV	PrecipitationDataConversionTool	M→D	ConvertedPrecipitationData
IV	ConvertedPrecipitationData	D→M	FloodDepthPrediction
IV	FloodDepthPrediction	M→D	PredictedWaterDepths
IV	PredictedWaterDepths	D→M	Depth-SpeedModel
IV	Depth-SpeedModel	M→D	TrafficSpeed

Table 13: Typed edges in the DisasterBench DAG by subgraph (Part IV).

Type	Count	% of Total	Depth Range	Avg. Depth
Node	35	15.0%	1	1.00
Chain	166	71.2%	2–9	2.79
Branching	32	13.7%	2–5	2.75
Total	233	100%	1–9	2.53

Table 14: Structural composition of DisasterBench tasks.

Metric	MiniLM	E5-small
Mean cosine similarity	0.350	0.850
Median cosine similarity	—	0.847
Std	—	0.035
Min / Max	— / 0.855	0.772 / 0.965
Pairs ≥ 0.50	60 (18.5%)	325 (100.0%)
Pairs ≥ 0.70	6 (1.8%)	325 (100.0%)
Pairs ≥ 0.80	2 (0.6%)	302 (92.9%)
Pairs ≥ 0.90	—	23 (7.1%)
Pairs ≥ 0.95	—	2 (0.6%)

Table 15: Pairwise cosine similarity statistics across all 325 agent pairs under two sentence encoders. Both encoders confirm high semantic overlap among operationally distinct tools, supporting the claim that benchmark difficulty arises from fine-grained semantic disambiguation rather than pool size.

Tool	Tasks (#)	Tasks (%)
Urban_Anomaly_Detection	55	23.6
GeoChat	54	23.2
Anomaly_Detection_Forest	49	21.0
Geospatial_Object_Segmentation	47	20.2
Landslide_Segmentation	39	16.7
Crowd_Counting_in_Adverse_Weather	38	16.3
Weather_Degraded_Image_Restoration	32	13.7
Low-Light_Object_Detection	29	12.4
Foggy_Scenario_Object_Detection	28	12.0
Building_damage_assessment	27	11.6
Change_Mapping_and_Detection	24	10.3
Multi_Spectral_Classifier	19	8.2
High-Resolution_Image_Reconstructor	16	6.9
Metadata_and_Text_Prompt_Image_Generation	15	6.4
Temporal_High_Resolution_Image_Generation	14	6.0
Temporal_Image_Sequence_Classifier	12	5.2
Flood_depth_prediction	10	4.3
RGB_GeoImage_Classifier	9	3.9
Precipitation_Nowcasting	7	3.0
precipitation_data_convert_tool	7	3.0
depth_speed_model	3	1.3
Event_detection	2	0.9
Multimodal_mobility_prediction_under_events	2	0.9
Post_Disaster_Mobility_Recovery	2	0.9
Toponym_Detection	2	0.9
Video_anomaly_detection	2	0.9

Table 16: Tool Invocation Frequency across 233 benchmark tasks.

		DP		CoT		ToT		RAP		ReAct	
Category	**Model**	A	EM	A	EM	A	EM	A	EM	A	EM
Frontier	Gemini 3.1 Pro Preview	75.97	68.24	81.97	72.53	82.40	**73.39**	82.40	72.10	79.83	69.96
	GPT-5.4	60.09	54.08	69.96	62.23	69.96	62.23	77.25	**65.67**	70.82	64.38
Strong Open-source	DeepSeek-V3.2	68.24	57.94	41.20	36.05	56.65	50.21	40.77	34.33	67.81	**61.80**
	DeepSeek-R1	60.52	54.08	69.10	**62.66**	61.80	53.22	38.63	35.19	60.52	56.22
	Qwen3-Max	62.66	55.79	71.67	62.66	73.39	**63.95**	73.39	63.09	67.38	62.23
	Qwen3-Max-Thinking	64.38	57.51	73.82	**66.52**	68.67	62.23	71.24	61.37	68.24	61.37
	Qwen3.5-27B	62.23	58.37	67.38	**62.23**	66.09	59.23	66.09	58.80	64.38	59.23
	Gemma-4-31B	73.82	**66.52**	69.96	63.09	65.67	58.37	60.09	50.64	68.24	62.23
	Llama-3.3-70B	60.94	38.63	66.95	**58.80**	51.50	41.63	31.76	28.76	51.93	38.63
Efficiency-oriented	Qwen3.5-9B	53.65	45.92	67.38	61.37	66.52	59.23	47.21	42.92	74.25	**65.67**
	Llama-3.1-8B	24.03	15.88	32.62	26.18	29.18	16.31	42.06	**28.33**	31.76	26.61
	Ministral-14B	30.90	20.17	38.63	33.48	24.03	16.31	24.89	18.88	56.22	**50.64**
	Ministral-8B	28.76	25.32	34.76	31.33	27.04	21.03	18.88	12.88	51.07	**44.21**
	Ministral-3B	16.74	10.30	27.90	21.03	39.06	18.88	13.30	3.86	42.49	**36.91**

Table 17: Tool Accuracy (A) and exact-match accuracy (EM, %) on DisasterBench across all methods. The gap between A and EM reflects the **semantic-execution gap**: failures that occur after correct tool selection due to parameter binding or dependency propagation errors. Both metrics are task-level booleans averaged over all 233 tasks. **Bold** indicates the best EM per model.

Evaluator Error Type	Top-Level Category	Condition
agent_mismatch	tool mismatch	Predicted tool $\neq$ gold tool
parameter_error	Parameter Binding Error	Agent matches but inputs or outputs differ
dependency_error	Parameter Binding Error	Agent matches but dependency list differs
dependency_content_error	Parameter Binding Error	Agent matches but dependency content differs
hallucinated_extra_steps	Structural Error	Predicted plan is longer than gold (and all aligned steps match)
early_stop	Structural Error	Predicted plan is shorter than gold (and all aligned steps match)
empty_output	Structural Error	Model produces no parseable plan
format_error	Structural Error	Output is parseable but violates JSON schema

Table 18: Mapping from fine-grained evaluator error types to the three top-level FPoF categories used in the main text.

Model	Method	Perfect	Top-3 FPoF Error Types
Gemini 3.1 Pro Preview	DP	159	agent_mismatch: 33 (14.2%), parameter_error: 32 (13.7%), early_stop: 7 (3.0%)
	CoT	169	agent_mismatch: 30 (12.9%), parameter_error: 30 (12.9%), halluc. steps: 3 (1.3%)
	ToT	171	agent_mismatch: 34 (14.6%), parameter_error: 26 (11.2%), halluc. steps: 2 (0.9%)
	RAP	168	parameter_error: 33 (14.2%), agent_mismatch: 27 (11.6%), early_stop: 3 (1.3%)
	ReAct	163	agent_mismatch: 39 (16.7%), parameter_error: 30 (12.9%), halluc. steps: 1 (0.4%)
GPT-5.4	DP	126	agent_mismatch: 59 (25.3%), parameter_error: 31 (13.3%), halluc. steps: 17 (7.3%)
	CoT	145	agent_mismatch: 53 (22.7%), parameter_error: 31 (13.3%), halluc. steps: 4 (1.7%)
	ToT	145	agent_mismatch: 51 (21.9%), parameter_error: 30 (12.9%), halluc. steps: 5 (2.1%)
	RAP	153	parameter_error: 38 (16.3%), agent_mismatch: 29 (12.4%), early_stop: 9 (3.9%)
	ReAct	150	agent_mismatch: 43 (18.5%), parameter_error: 33 (14.2%), halluc. steps: 6 (2.6%)
DeepSeek-V3.2	DP	135	agent_mismatch: 55 (23.6%), parameter_error: 32 (13.7%), halluc. steps: 10 (4.3%)
	CoT	84	parameter_error: 114 (48.9%), agent_mismatch: 30 (12.9%), halluc. steps: 4 (1.7%)
	ToT	117	agent_mismatch: 73 (31.3%), parameter_error: 24 (10.3%), halluc. steps: 17 (7.3%)
	RAP	80	early_stop: 70 (30.0%), parameter_error: 53 (22.7%), agent_mismatch: 22 (9.4%)
	ReAct	144	agent_mismatch: 59 (25.3%), parameter_error: 24 (10.3%), early_stop: 3 (1.3%)
DeepSeek-R1	DP	126	agent_mismatch: 59 (25.3%), parameter_error: 37 (15.9%), early_stop: 7 (3.0%)
	CoT	146	agent_mismatch: 57 (24.5%), parameter_error: 24 (10.3%), halluc. steps: 4 (1.7%)
	ToT	124	agent_mismatch: 71 (30.5%), parameter_error: 29 (12.4%), halluc. steps: 9 (3.9%)
	RAP	82	parameter_error: 104 (44.6%), agent_mismatch: 32 (13.7%), early_stop: 13 (5.6%)
	ReAct	131	agent_mismatch: 74 (31.8%), parameter_error: 23 (9.9%), early_stop: 3 (1.3%)

Table 19: Per-model FPoF breakdown across all five planning methods. For each model–method pair, we report the number of perfect matches (out of 233 tasks) and the three most frequent first-error types. Percentages are relative to the 233 total tasks.

Model	Method	Perfect	Top-3 FPoF Error Types
Qwen3-Max	DP	130	agent_mismatch: 65 (27.9%), parameter_error: 24 (10.3%), halluc. steps: 13 (5.6%)
	CoT	146	agent_mismatch: 51 (21.9%), parameter_error: 27 (11.6%), halluc. steps: 5 (2.1%)
	ToT	149	agent_mismatch: 49 (21.0%), parameter_error: 28 (12.0%), halluc. steps: 4 (1.7%)
	RAP	147	agent_mismatch: 43 (18.5%), parameter_error: 28 (12.0%), early_stop: 7 (3.0%)
	ReAct	145	agent_mismatch: 52 (22.3%), parameter_error: 27 (11.6%), early_stop: 5 (2.1%)
Qwen3-Max-Thinking	DP	134	agent_mismatch: 61 (26.2%), parameter_error: 25 (10.7%), halluc. steps: 13 (5.6%)
	CoT	155	agent_mismatch: 46 (19.7%), parameter_error: 21 (9.0%), halluc. steps: 7 (3.0%)
	ToT	145	agent_mismatch: 54 (23.2%), parameter_error: 23 (9.9%), halluc. steps: 8 (3.4%)
	RAP	143	agent_mismatch: 45 (19.3%), parameter_error: 28 (12.0%), halluc. steps: 10 (4.3%)
	ReAct	143	agent_mismatch: 55 (23.6%), parameter_error: 28 (12.0%), early_stop: 5 (2.1%)
Qwen3.5-27B	DP	136	agent_mismatch: 56 (24.0%), parameter_error: 28 (12.0%), halluc. steps: 12 (5.2%)
	CoT	145	agent_mismatch: 47 (20.2%), parameter_error: 32 (13.7%), halluc. steps: 6 (2.6%)
	ToT	138	agent_mismatch: 51 (21.9%), parameter_error: 25 (10.7%), halluc. steps: 17 (7.3%)
	RAP	137	agent_mismatch: 41 (17.6%), parameter_error: 28 (12.0%), halluc. steps: 15 (6.4%)
	ReAct	138	agent_mismatch: 49 (21.0%), parameter_error: 26 (11.2%), halluc. steps: 19 (8.2%)
Gemma-4-31B	DP	155	agent_mismatch: 48 (20.6%), parameter_error: 27 (11.6%), halluc. steps: 3 (1.3%)
	CoT	147	agent_mismatch: 57 (24.5%), parameter_error: 25 (10.7%), early_stop: 2 (0.9%)
	ToT	136	agent_mismatch: 63 (27.0%), parameter_error: 27 (11.6%), halluc. steps: 5 (2.1%)
	RAP	118	parameter_error: 73 (31.3%), agent_mismatch: 31 (13.3%), halluc. steps: 6 (2.6%)
	ReAct	145	agent_mismatch: 56 (24.0%), parameter_error: 25 (10.7%), halluc. steps: 6 (2.6%)

Table 20: Per-model FPoF breakdown across all five planning methods (continued, part 2 of 4).

Model	Method	Perfect	Top-3 FPoF Error Types
Llama-3.3-70B	DP	90	parameter_error: 72 (30.9%), agent_mismatch: 56 (24.0%), halluc. steps: 15 (6.4%)
	CoT	137	agent_mismatch: 52 (22.3%), parameter_error: 31 (13.3%), halluc. steps: 9 (3.9%)
	ToT	97	agent_mismatch: 57 (24.5%), parameter_error: 41 (17.6%), halluc. steps: 34 (14.6%)
	RAP	67	parameter_error: 99 (42.5%), agent_mismatch: 39 (16.7%), early_stop: 28 (12.0%)
	ReAct	90	agent_mismatch: 70 (30.0%), parameter_error: 49 (21.0%), halluc. steps: 24 (10.3%)
Qwen3.5-9B	DP	107	parameter_error: 63 (27.0%), agent_mismatch: 37 (15.9%), early_stop: 18 (7.7%)
	CoT	143	parameter_error: 49 (21.0%), agent_mismatch: 27 (11.6%), early_stop: 9 (3.9%)
	ToT	138	agent_mismatch: 51 (21.9%), parameter_error: 24 (10.3%), halluc. steps: 18 (7.7%)
	RAP	100	parameter_error: 40 (17.2%), agent_mismatch: 39 (16.7%), early_stop: 39 (16.7%)
	ReAct	153	agent_mismatch: 41 (17.6%), parameter_error: 29 (12.4%), halluc. steps: 6 (2.6%)
Llama-3.1-8B	DP	37	agent_mismatch: 92 (39.5%), parameter_error: 87 (37.3%), halluc. steps: 17 (7.3%)
	CoT	61	agent_mismatch: 122 (52.4%), parameter_error: 29 (12.4%), halluc. steps: 21 (9.0%)
	ToT	38	agent_mismatch: 111 (47.6%), parameter_error: 43 (18.5%), halluc. steps: 28 (12.0%)
	RAP	66	agent_mismatch: 75 (32.2%), parameter_error: 57 (24.5%), early_stop: 15 (6.4%)
	ReAct	62	agent_mismatch: 95 (40.8%), halluc. steps: 37 (15.9%), parameter_error: 37 (15.9%)

Table 21: Per-model FPoF breakdown across all five planning methods (continued, part 3 of 4).

Model	Method	Perfect	Top-3 FPoF Error Types
Ministral-14B	DP	47	agent_mismatch: 67 (28.8%), parameter_error: 62 (26.6%), halluc. steps: 57 (24.5%)
	CoT	78	agent_mismatch: 61 (26.2%), halluc. steps: 59 (25.3%), parameter_error: 34 (14.6%)
	ToT	38	agent_mismatch: 82 (35.2%), parameter_error: 62 (26.6%), halluc. steps: 51 (21.9%)
	RAP	44	parameter_error: 105 (45.1%), agent_mismatch: 54 (23.2%), halluc. steps: 25 (10.7%)
	ReAct	118	agent_mismatch: 68 (29.2%), parameter_error: 31 (13.3%), halluc. steps: 16 (6.9%)
Ministral-8B	DP	59	agent_mismatch: 82 (35.2%), halluc. steps: 59 (25.3%), parameter_error: 33 (14.2%)
	CoT	73	agent_mismatch: 92 (39.5%), halluc. steps: 46 (19.7%), parameter_error: 21 (9.0%)
	ToT	49	agent_mismatch: 77 (33.0%), halluc. steps: 55 (23.6%), parameter_error: 51 (21.9%)
	RAP	30	parameter_error: 140 (60.1%), agent_mismatch: 38 (16.3%), halluc. steps: 20 (8.6%)
	ReAct	103	agent_mismatch: 74 (31.8%), parameter_error: 32 (13.7%), halluc. steps: 24 (10.3%)
Ministral-3B	DP	24	agent_mismatch: 136 (58.4%), parameter_error: 67 (28.8%), halluc. steps: 6 (2.6%)
	CoT	49	agent_mismatch: 140 (60.1%), parameter_error: 31 (13.3%), halluc. steps: 13 (5.6%)
	ToT	44	agent_mismatch: 82 (35.2%), parameter_error: 74 (31.8%), halluc. steps: 17 (7.3%)
	RAP	9	parameter_error: 104 (44.6%), agent_mismatch: 103 (44.2%), halluc. steps: 10 (4.3%)
	ReAct	86	agent_mismatch: 93 (39.9%), parameter_error: 34 (14.6%), halluc. steps: 18 (7.7%)

Table 22: Per-model FPoF breakdown across all five planning methods (continued, part 4 of 4).