FABLE: A Novel Data-Flow Analysis Benchmark on Procedural Text for Large Language Model Evaluation

Vishal Pallagani, Nitin Gupta, John A. Aydin, Biplav Srivastava

University of South Carolina, Columbia, SC, USA
{vishalp, niting, jaaydin, biplav.s}@email.sc.edu

Abstract

Data-flow reasoning, the ability to track how information is created, updated, and propagated across steps, is a core component of procedural language understanding, yet it is not well characterized by existing evaluations of LLMs. We introduce FABLE, a diagnostic benchmark for assessing data-flow reasoning over structured procedural text, consisting of 2,400 question–answer pairs spanning three real-world domains: cooking recipes, travel routes, and automated plans. FABLE is designed to probe fine-grained information propagation and intermediate state tracking behaviors inspired by classical program analysis. We evaluate reasoning-focused, general-purpose, and code-specific models primarily at a fixed parameter scale of 8B to enable controlled comparison across model types. At this scale, models perform above random baselines on average but remain far from human reliability, with the strongest reasoning-focused model reaching 82.63% accuracy compared to a 96% human majority-vote baseline, while many analyses and models remain near chance. General-purpose and code-specific models perform considerably worse, often near chance on several data-flow categories. Additional experiments across larger parameter sizes exhibit consistent qualitative trends. Overall, FABLE provides a controlled resource for evaluating data-flow reasoning in procedural text and helps clarify the gap between current model capabilities and human performance.

Introduction

Procedural texts such as recipes, travel routes, and plans (McDermott 1978) describe processes that unfold over time. Understanding such texts requires more than interpreting individual steps in isolation; it requires tracking how entities, variables, and their attributes are introduced, updated, and propagated across a procedure (Rosen 1979). This capability can be characterized as *data-flow reasoning*: reasoning about how information evolves as a process progresses (Khedker, Sanyal, and Sathe 2017).

In software engineering, data-flow analysis provides a principled framework for modeling how values flow through programs and across execution paths (Dwyer and Clarke 1994; Fosdick and Osterweil 1976; Kennedy 1979; Pollock and Soffa 2002). Although natural language procedures differ from source code in form, they exhibit closely related structural properties: entities are defined, modified, reused, and constrained over time, often with implicit dependencies between steps. Unlike programs, however, procedural texts are informal, underspecified, and rely on implicit world knowledge, making data-flow reasoning in natural language a distinct and challenging problem.

Despite this alignment, existing evaluations of large language models do not directly assess data-flow reasoning in procedural text. Prior benchmarks have examined individual aspects of procedural understanding, such as entity state tracking (Tandon et al. 2018b) or causal and temporal relations (Tandon et al. 2019; Ning et al. 2020). While informative, these datasets typically focus on a single reasoning dimension and short descriptions, whereas real-world procedures require jointly reasoning about temporal structure, causal effects, and evolving entity states over extended sequences.

In this work, we introduce FABLE, a benchmark designed to systematically evaluate data-flow reasoning in procedural text. FABLE adapts a diverse set of classical data-flow analyses to natural language procedures and instantiates them across three real-world domains: cooking recipes, travel routes, and automated plans. The resulting benchmark consists of 2,400 question–answer pairs with balanced coverage across domains and analysis types, enabling controlled and fine-grained evaluation.

We evaluate FABLE on reasoning-focused, general-purpose, and code-specific large language models at a fixed parameter scale of 8B. Our results show that while reasoning-focused models outperform other model classes, they still fall well short of human performance. Additional experiments across larger model sizes exhibit similar qualitative trends, indicating that data-flow reasoning remains a challenging and underexplored capability for current language models.

Background & Related Work

In this section, we situate FABLE within prior work on procedural text understanding and motivate the need for evaluating data-flow reasoning that jointly captures entity state, causal, and temporal dependencies.

Table 1: Comparison of representative procedural reasoning benchmarks. FABLE emphasizes integrated data-flow reasoning across entity state, causal, and temporal dimensions.

Benchmark	Year (Venue)	Primary Domain	Reasoning Focus	Scale
ProPara	2018 (NAACL)	Science	Entity state tracking (location, existence)	488 procedures, 81k state annotations
WIQA	2019 (EMNLP)	Science	Counterfactual causal reasoning	379 procedures, 40.7k QA pairs
OpenPI	2020 (EMNLP)	WikiHow	Open-vocabulary state changes	810 procedures, 29.9k state changes
TORQUE	2020 (EMNLP)	News	Temporal order reasoning	3.2k snippets, 21k questions
TRACIE	2021 (NAACL)	News, stories	Implicit temporal event inference	1.7k event pairs
CREPE	2023 (EACL)	Cooking, household	Step-level causal effects	183 procedures, 324 events
TRAC	2023 (ACL)	Blocksworld	Action/change reasoning over preconditions and effects	60k QA pairs
ActionReasoningBench	2025 (ICLR)	Plans	Fluent/state tracking, executability, effects, and composite RAC	152k QA pairs
ACPBench Hard	2026 (ICLR)	Plans	Generative reasoning about action, change, and planning	3,720 QA pairs
FABLE (ours)	**—**	**Recipes, travel, plans**	**Integrated data-flow reasoning**	**2,400 QA pairs across 3 domains**

Procedural Text and the Need for Data-Flow Understanding

We adopt and refine the definition of procedural text from prior work (Zhang et al. 2023). A *procedural text* can be formalized as a triple $\mathcal{P} = (G, S, \preceq)$, where G is a natural-language goal describing the intended outcome, $S = \{s_1, \ldots, s_n\}$ is a finite set of natural-language steps, and $\preceq$ specifies a partial or total order over these steps. Understanding a procedure requires reasoning about how entities and their attributes are introduced, modified, and reused across ordered steps, and how intermediate results contribute to achieving the goal (Tandon et al. 2018a). This process naturally calls for *data-flow reasoning*, a concept originating in program analysis, where variable definitions, updates, and dependencies are tracked across execution paths. Analogously, in natural language procedures, data-flow reasoning involves identifying where entities are defined, how they are transformed, and how later steps depend on previously established states. Unlike source code, however, procedural texts are informal, underspecified, and rely heavily on implicit world knowledge, making data-flow reasoning in natural language both distinct and challenging. Despite its central role in procedural understanding, data-flow reasoning is not directly evaluated by existing benchmarks for large language models, which tend to focus on isolated reasoning phenomena. This gap motivates the need for systematic evaluation of data-flow reasoning in procedural text.

Prior Relevant Benchmarks

Early work on procedural text understanding focused on tracking how entities change state throughout a process. ProPara (Tandon et al. 2018a) introduced entity state tracking in science paragraphs, requiring models to identify changes in properties such as location or existence at each step. OpenPI (Tandon et al. 2020) extended this setting to real-world how-to guides from WikiHow, asking models to produce structured representations of state changes across procedural steps.

Subsequent benchmarks emphasized causal reasoning in procedural contexts. WIQA (Tandon et al. 2019) introduced counterfactual questions grounded in influence graphs that encode causal relations between events, while CREPE (Zhang et al. 2023) focused on linking procedural steps to changes in event likelihood via intermediate entity states. Although these datasets expose important model limitations, models on WIQA and TORQUE continue to perform substantially below human levels, and the tasks primarily target localized or single-step causal effects rather than reasoning over the full procedural structure.

Procedural understanding also requires temporal reasoning, including both explicit event orderings and implicit events occurring between steps. Benchmarks such as TORQUE (Ning et al. 2020) and TRACIE (Zhou et al. 2021) evaluate temporal relations in narrative and news text, for example by inferring before–after relations or identifying unstated events. However, these benchmarks assess temporal, causal, and state-based reasoning largely in isolation.

In contrast, real-world procedures involve tightly coupled interactions among these dimensions. FABLE is designed to evaluate how entity states evolve over time, how causal effects propagate across steps, and how temporal structure constrains valid reasoning paths within a single unified framework inspired by classical data-flow analysis. Finally, most prior procedural benchmarks have been evaluated primarily using encoder-only architectures such as BERT and RoBERTa, leaving open questions about how modern autoregressive large language models handle procedural reasoning in more complex, generative settings. FABLE complements prior work by enabling systematic evaluation of data-flow reasoning across diverse model families under controlled conditions.

Another line of related work studies reasoning about actions and change (RAC). Benchmarks in this area similarly target temporal, causal, and state-change reasoning over action sequences, but they are typically grounded in classical planning domains. TRAC (He et al. 2023) probes RAC from

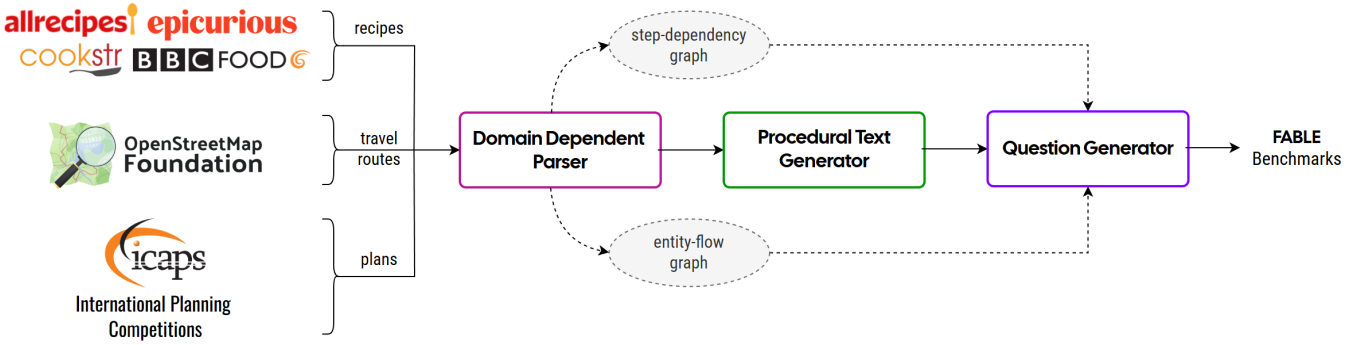

Figure 1: Overview of the FABLE dataset construction pipeline. Raw procedural data is collected from three domains: recipes (human-executed), travel routes (semi-automated), and plans (automated).

Table 2: Dataset statistics across domains in FABLE. **Procs** = procedures; **Avg** = average steps per procedure; **QA Gen** = total question–answer pairs generated; **QA Used** = balanced subset used in the benchmark.

Domain	Procs	Avg	Min	Max	QA Gen	QA Used
Recipes	1,382	4.15	3	17	1,600	800
Travel	1,500	17.26	4	56	800	800
Plans	1,378	6.77	1	15	11,024	800
Total	4,260	9.39	–	–	13,424	2,400

natural-language plans through tasks such as projection, plan verification, and goal recognition. ACPBench (Kokel et al. 2025), and later ACPBench-Hard (Kokel et al. 2026), evaluate related planning-oriented abilities, including plan validation, landmark identification, and next-action prediction. ActionReasoningBench (Handa et al. 2025) further expands this line by covering eight planning domains and testing models on action-centric reasoning tasks such as fluent tracking, state tracking, action executability, and reasoning over action effects.

Despite their differences, these benchmarks share an important limitation: they are restricted to classical planning settings rather than covering a broader range of procedural tasks. In contrast, FABLE is designed as an extensible framework for evaluating reasoning across diverse domains that can be represented as procedural text.

FABLE Dataset Construction

In this section, we describe the construction of the FABLE benchmark, covering domain selection, raw data collection, domain-specific parsing, procedural text generation, and the creation of question–answer instances grounded in classical data-flow analyses. An overview of the construction pipeline is shown in Figure 1.

Domain Selection and Data Sources

We select three distinct domains[1] to construct the FABLE benchmark. These domains are chosen to span a broad range of procedural complexity and automation levels: cooking recipes are fully human-executed, travel routes consist of semi-automated navigation instructions that may be followed by humans or autonomous systems, and automated plans used in robotics are fully generated and executed by planning systems. This diversity enables FABLE to evaluate data-flow reasoning across heterogeneous forms of procedural text encountered in real-world settings. Additional data source and preprocessing details are provided in Appendix A.5.

Recipes. Cooking recipes represent human-executed procedures with explicit temporal ordering, rich causal structure, and frequent entity transformations. We draw from the publicly available English Recipes Dataset (Vance 2017), which aggregates recipes scraped from online cooking resources. To ensure well-formed procedural structure, we standardize recipe formats and filter out instances containing fewer than three steps, yielding a base corpus of 74,031 recipes spanning diverse cuisines. To support reliable entity tracking, we perform entity frequency analysis to identify commonly occurring ingredients, tools, and intermediate products. Candidate entities are extracted using spaCy's `en_core_web_trf` model (Honnibal et al. 2020), followed by manual curation to remove noisy or uninformative terms such as measurement units and generic descriptors. We retain only recipes whose entities fall within this curated vocabulary, resulting in a high-quality subset of 1,382 recipes used throughout this work.

Travel Routes. The travel routes domain consists of structured navigation instructions generated by rule-based systems, representing semi-automated procedures. We construct this corpus using Valhalla's open-source turn-by-turn routing API (Valhalla), which generates natural-language driving directions from OpenStreetMap data. For each start–destination

[1]We use "domain" to denote application scenarios (recipes, travel routes, and plans), and qualify it with IPC when referring to its usage in the planning literature as problem categories (e.g., Hanoi and Gripper).

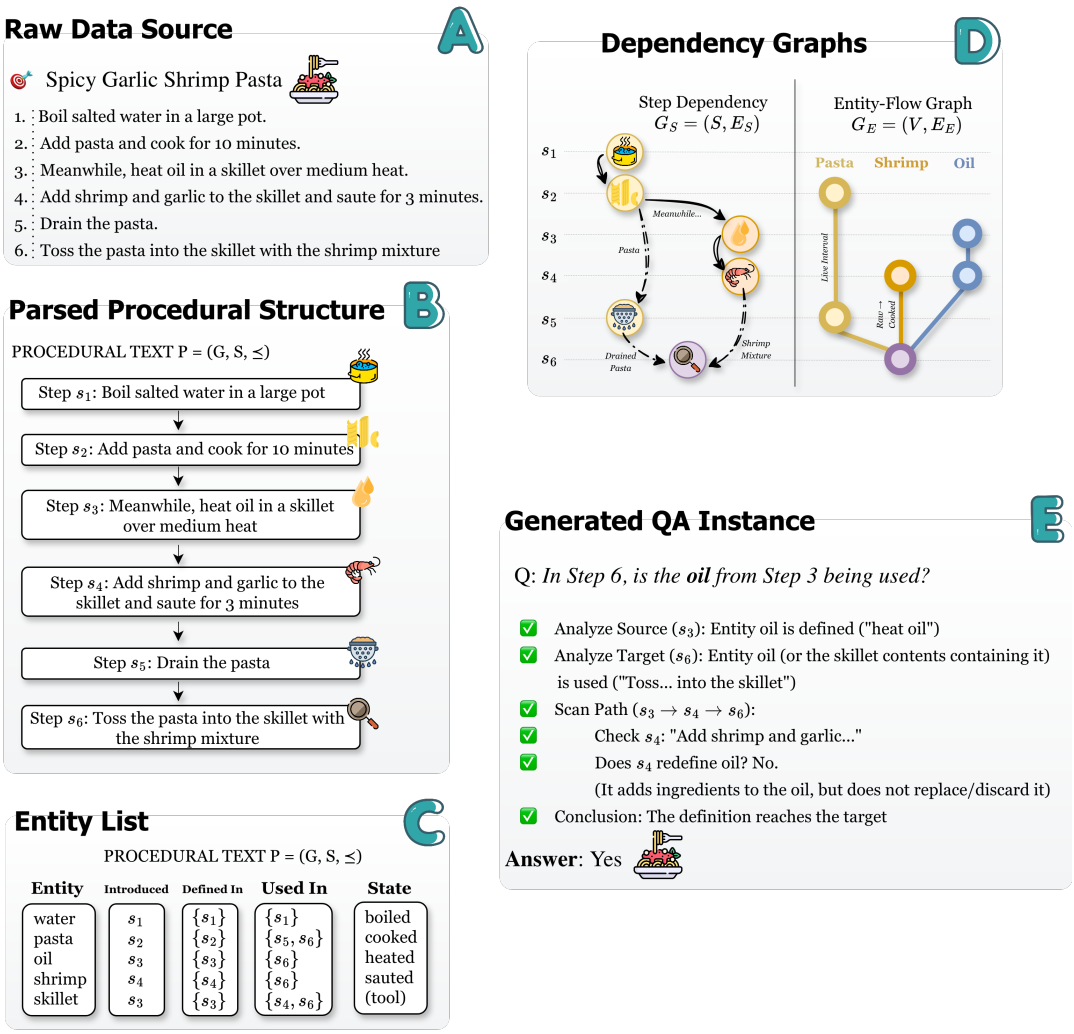

Figure 2: End-to-end walkthrough of FABLE construction for a recipe procedure. Panels illustrate (A) raw procedural text, (B) parsed procedural structure, (C) extracted entities and state tracking, (D) step-dependency and entity-flow graphs, and (E) a generated data-flow question–answer instance with deterministic reasoning shown to illustrate how ground-truth labels are generated.

pair, we query Valhalla under five distinct costing configurations (e.g., avoiding highways or penalizing turns) and generate multiple alternative routes.[2] To ensure procedural validity and reduce redundancy, we post-process routes to remove duplicate actions and normalize step formatting so that instructions can be followed as plain text. Locations are sampled from publicly available datasets of airports, historic sites, government buildings, hospitals, and landmarks within the contiguous United States. Each route is constrained to be intra-state and to span at most 150 kilometers, ensuring sufficient procedural complexity while maintaining interpretability. The resulting corpus reflects realistic, goal-directed navigation tasks composed of multi-step, temporally ordered instructions.

Plans. Automated planning concerns generating a sequence of actions that transforms an initial state into a desired goal state given a model of the environment (Ghallab, Nau, and Traverso 2004). Formally, a classical planning problem is defined as a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, G)$, where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ the transition function, $s_0 \in \mathcal{S}$ the initial state, and $G \subseteq \mathcal{S}$ the set of goal states (Russell and Norvig 2005). We use four classical domains from the International Planning Competition (IPC) (ICAPS 2022): Hanoi, Gripper, Ferry, and Driverlog (Community). These domains introduce distinct challenges including sequential dependencies, resource transportation, and object manipulation. We employ the Fast Downward planning system (Helmert 2006) to generate ground-truth plans from domain and problem specifications,

[2]In our implementation, we generate three distinct routes per start–destination pair by sampling from Valhalla outputs produced under five costing configurations.

resulting in 1,378 distinct plans. These plans form the procedural basis for generating structured texts and question–answer instances in the FABLE benchmark.

Domain-Dependent Parser

Given raw data from each domain, the first step in the FABLE construction pipeline converts procedural inputs into structured representations that explicitly encode step dependencies and entity flow. This transformation is performed by a *domain-dependent parser* customized for recipes, travel routes, and automated plans. For each procedure, the parser produces two outputs: a **step-dependency graph** $G_S = (S, E_S)$, where S is the set of steps and $E_S \subseteq S \times S$ encodes ordering constraints; and an **entity-flow graph** $G_E = (V, E_E)$, where V is the set of entities (or variables) and $E_E \subseteq S \times V \times S$ captures entity flow events across steps, including creation, modification, and consumption. Figure 2 illustrates the end-to-end construction process for a representative recipe, including procedural parsing, graph construction, and question generation.

Parsing strategies are adapted to domain-specific structure. For recipes, we apply NLP techniques to identify actions (verbs) and their arguments (ingredients and tools), and use temporal cues such as "after stirring" or "meanwhile" to infer ordering constraints in E_S. Entity flow in G_E is constructed by tracking ingredient and intermediate-product mentions across steps and associating them with transformation operations (e.g., mixing, heating) that modify entity states. For travel routes, structured metadata derived from OpenStreetMap is used to extract maneuvers, waypoints, and route segments, with step ordering determined by the sequential progression of travel instructions. For automated plans, grounded STRIPS-style action schemas specify preconditions and effects (McDermott 1978), allowing direct construction of G_S from the plan sequence and G_E by tracking predicate and object state changes across actions. To assess parser reliability, we manually audited 50 randomly sampled procedures per domain, verifying (i) step segmentation and ordering constraints and (ii) entity identification and flow across steps. We did not observe systematic errors that would affect downstream question generation.

Procedural Text Generator

The next stage in the FABLE construction pipeline generates coherent natural-language procedural texts while preserving their underlying procedural structure. The objective of this stage is to standardize representation across domains by extracting a common abstraction consisting of a goal, a set of steps, and ordering constraints, without altering the semantics of the original procedures. For recipes, procedures are already expressed in natural language. Each recipe includes an explicit goal G describing the intended dish and a sequence of steps $S = \{s_1, \ldots, s_n\}$ corresponding to cooking instructions. Temporal and causal cues in the text (e.g., "after mixing," "meanwhile") are used to infer a partial ordering $\preceq$ over steps. The parser extracts G, S, and $\preceq$ directly from the text, preserving the ordering information conveyed by the original recipe. For travel routes, the goal G is defined by a source–destination pair specified as geographic coordinates. The step set $S = \{s_1, \ldots, s_n\}$ consists of human-readable navigation instructions extracted from route metadata (e.g., "Turn left onto Airport Boulevard"). The ordering $\preceq$ is a total order, reflecting the strictly sequential execution of navigation instructions from origin to destination. For automated planning domains, the raw input consists of grounded action sequences specified in PDDL. The goal G corresponds to the desired final state defined in the planning problem, and the steps $S = \{s_1, \ldots, s_n\}$ are obtained from a valid plan generated by a planner. To render symbolic actions in natural language, we use AutoPlanBench (Stein et al. 2023), which maps each grounded action to a fluent English instruction while preserving its semantic role. As in travel routes, the resulting procedures follow a total order. By standardizing procedural texts across domains into a shared goal–step–ordering representation, FABLE enables unified evaluation of data-flow reasoning over diverse procedural inputs.

Question Generator

The final stage of FABLE construction generates diagnostic question–answer instances that probe data-flow reasoning over procedural texts. We instantiate classical data-flow analyses from software engineering over the structured graphs extracted during parsing and render them as natural-language questions grounded in each procedure. Table 3 summarizes the original software definitions, their procedural adaptations, and the reasoning dimensions each analysis targets.

Given a step-dependency graph $G_S = (S, E_S)$ and an entity-flow graph $G_E = (V, E_E)$, we employ a template-based generation approach to produce question instances. For each analysis, we define canonical templates that map properties of G_S and G_E to natural-language questions with deterministically computable answers. This design enables scalable and consistent generation across domains without manual annotation. All analyses except Interval Analysis yield binary *yes/no* questions, reflecting the boolean nature of classical data-flow properties while enabling controlled evaluation. Interval Analysis produces non-binary answers in the form of numeric ranges (e.g., temperature or time bounds) or step-indexed intervals (e.g., "between Step 4 and Step 7"). Prompt templates and examples are provided in Appendix A.11.

Certain domain–analysis combinations produce deterministic labels due to structural properties of the procedures. For example, in travel routes, Available Expressions and Taint Analysis evaluate to *Yes* under our formulation because routes monotonically accumulate reachable locations and do not model semantic contamination. Conversely, Very Busy Expressions, Type-State Analysis, and Concurrency Analysis evaluate to *No* because route instructions are strictly sequential and lack branching or reusable side effects. In recipes, Reaching Definitions and Available Expressions evaluate to *Yes* since introduced ingredients persist across steps, while Type-State and Concurrency analyses evaluate to *No* due to strict temporal dependencies. Taint Analysis evaluates to *Yes* in recipes because sanitization actions (e.g., heating) eliminate contamination under our operationalization.

These outcomes follow directly from the semantics encoded in G_S and G_E under the definitions in Table 3, rather

Table 3: Data-flow analyses, their procedural adaptations, and associated reasoning dimensions.

Data-Flow Analysis	Software Definition	Procedural Text Adaptation	Reasoning Type
Reaching Definitions	Tracks which variable definitions reach a program point without being overridden.	Determines whether an entity introduced in one step influences its use in later steps.	State
Very Busy Expressions	Identifies expressions guaranteed to be evaluated before operand modification.	Checks whether an entity produced in a step must be consumed along all future paths.	Causal
Available Expressions	Determines whether a computed expression remains valid.	Verifies whether an entity's computed state can be reused without recomputation.	State
Live Variable Analysis	Checks whether a variable will be used again before being discarded.	Determines whether an entity or resource remains required in future steps.	State
Interval Analysis	Computes numeric bounds of variables.	Tracks valid numeric ranges (e.g., time, temperature, quantity) across steps.	Temporal
Type-State Analysis	Enforces valid object state transitions.	Ensures entities follow permissible state transitions across steps.	State
Taint Analysis	Tracks propagation of contaminated data.	Traces how undesirable entities influence downstream steps under domain-specific rules.	Causal
Concurrency Analysis	Identifies operations that can execute safely in parallel.	Detects whether steps can be reordered or executed concurrently without violating dependencies.	Temporal

Table 4: Human performance on a balanced evaluation subset of FABLE (96 questions), reported as average individual accuracy, majority-vote accuracy across annotators, and inter-annotator agreement (Cohen's κ).

Domain	Acc. (%)	Maj. Acc. (%)	κ
Recipes	96.9	98.4	0.82
Travel Routes	95.8	97.9	0.79
Plans	95.6	97.3	0.77
Overall	**96.1**	**97.9**	**0.79**

than from annotation artifacts. While the full generation process yields 13,424 question–answer pairs, we construct a balanced benchmark subset of 2,400 instances by sampling exactly 100 questions per analysis per domain (800 per domain total), enabling fair comparison across analyses and domains (Table 2). Additional clarifications addressing common questions about analysis formalization, deterministic labels, question design, and evaluation assumptions are provided in Appendix A.4.

Experimental Setup & Results

Our experimental evaluation addresses two research questions: **(1)** Is FABLE a challenging benchmark for current large language models? **(2)** How does model performance vary across data-flow analyses? We evaluate models on FABLE and analyze performance across data-flow analyses and procedural domains.

Baseline Models

We evaluate three representative model *types*: a reasoning-oriented model (`deepseek-r1:8b`), a general-purpose instruction-tuned model (`llama3.1:8b`), and a code-specialized instruction-tuned model (`granite-code:8b`). All models are approximately 8B parameters in size. This scale is commonly treated as an upper bound for practical deployment under moderate resource constraints (e.g., single-GPU or edge-class settings), while remaining large enough to exhibit non-trivial reasoning behavior.

Evaluation Setup

Each model is evaluated on the full FABLE benchmark, spanning all domains and data-flow analyses. For each question instance, we generate five independent completions using greedy decoding, with randomized sampling seeds held fixed across models. Each model is given the procedural text, goal, a brief description of the data-flow analysis associated with the questions, and the benchmark question. Final predictions are obtained via majority voting over the sampled completions to reduce stochastic variability. For binary questions, we report accuracy with respect to the gold labels. For interval-based questions, a prediction is considered correct if the extracted value falls within the expected numeric range or correctly identifies the relevant step interval specified by the procedural structure. All inference is performed using the Ollama Python API (Marcondes et al. 2025) under default decoding settings, without manual constraints on temperature, generation length, or sampling strategy. Prompts are processed sequentially, without batching or caching, to ensure consistent inference behavior across models. Hardware and execution details for all experiments are provided in Appendix A.3.

Human Baseline Evaluation

To contextualize model performance, we conduct a controlled human evaluation on a representative subset of FABLE to assess whether its data-flow reasoning tasks are reliably solvable by humans under the adopted definitions. We sample a balanced subset of 96 question–answer instances, covering all combinations of three domains and eight data-flow analyses, with four questions per domain–analysis pair. Four graduate-level annotators with backgrounds in computer science independently answer each question, using only the procedural text and question prompt, without access to gold answers or underlying graph representations. The task format exactly mirrors that used in model evaluation. Table 4 reports

human performance. Per-analysis human results are reported in Appendix A.7. Annotators achieve an overall accuracy of 96.1% with substantial inter-annotator agreement (Fleiss' $\kappa = 0.79$), indicating that the questions are well-specified and admit largely unambiguous answers.

Results and Analysis

We organize our analysis around the two research questions introduced earlier.

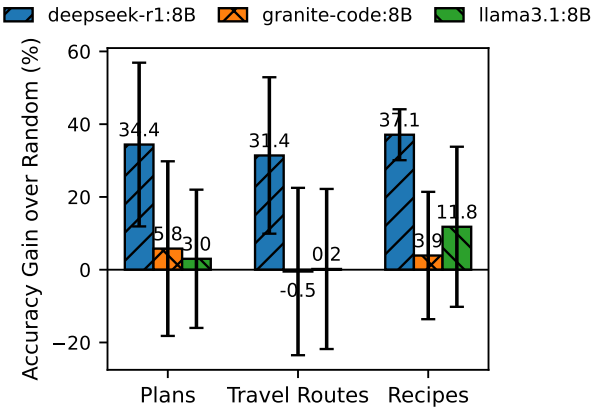

Figure 3: Domain-wise accuracy gains over structured random baselines, averaged across data-flow analyses.

Is FABLE a challenging benchmark for current models? To assess whether FABLE poses a meaningful challenge for large language models, we compare model performance against structured random baselines tailored to each domain and data-flow analysis. For binary questions, random predictions are sampled uniformly from *Yes/No*. For interval-based questions, random answers are generated from domain-specific numeric ranges or step-indexed intervals inferred from the gold answer distributions. Figure 3 reports, for each model and domain, accuracy gains relative to these baselines. Full domain-wise results and significance tests are in Appendix A.8. The reasoning-focused model (`deepseek-r1:8b`) consistently exceeds random performance across all domains, with gains of 34.4% on plans, 31.4% on travel routes, and 37.1% on recipes. In contrast, the code-specific (`granite-code:8b`) and general-purpose (`llama3.1:8b`) models exhibit only marginal improvements. `granite-code:8b` improves by 5.8% on plans, performs slightly below random on travel routes (-0.5%), and improves by 3.9% on recipes. `llama3.1:8b` shows similarly small gains of 3.0%, 0.2%, and 11.8% across the three domains, indicating near–chance-level behavior. Across models, accuracy gains exhibit substantial variability, with standard deviations frequently exceeding ± 20 percentage points when aggregated over data-flow analyses. This suggests that benchmark difficulty is uneven across question instances. In comparison, human annotators achieve over 96% accuracy on a balanced subset of FABLE (Section), indicating that the benchmark is well-defined and that the observed gaps primarily reflect current model limitations.

How does model performance vary across data-flow analyses? Model performance varies substantially across data-flow analyses and domains. Figure 4 shows that `deepseek-r1:8b` consistently outperforms the other models, but still exhibits pronounced weaknesses on specific analyses. The most severe gap occurs in Interval Analysis: while human accuracy remains high across domains (above 90%), `deepseek-r1:8b` achieves only 31% on plans and 39% on travel routes, with `granite-code:8b` and `llama3.1:8b` near chance. This contrast indicates that numeric and range-based reasoning over procedural structure is difficult for current models despite being reliably solvable by humans. Models perform comparatively better on analyses with localized dependencies, such as Reaching Definitions and Available Expressions, where human performance is near ceiling and model accuracy is higher, particularly for the code-specialized model. In contrast, analyses requiring implicit state persistence or temporal and causal reasoning, including Live Variable, Taint, and Concurrency Analysis, show larger and less stable gaps from human-level performance, even for the strongest model. Overall, these results indicate that baseline performance depends strongly on the type of data-flow reasoning required: models handle local, surface-aligned dependencies more effectively than analyses that require composing latent procedural state across multiple steps.

Inference Efficiency

Table 5 reports average inference time per 100 prompts. While `deepseek-r1:8b` achieves superior accuracy, it incurs over a $20\times$ slowdown relative to the fastest model in each domain. This highlights a clear trade-off between reasoning capability and deployment efficiency, particularly relevant for real-time or resource-constrained settings. Additional inference-time measurements and breakdowns are in Appendix A.10.

Table 5: Average inference time per 100 prompts (seconds). **Slow.** denotes slowdown relative to the fastest model per domain.

Model	Plans		Travel		Recipes	
	Time	Slow.	Time	Slow.	Time	Slow.
`deepseek-r1`	1136	29.6×	1060	37.6×	604	21.4×
`granite-code`	38	1.0×	46	1.6×	46	1.6×
`llama3.1`	38	1.0×	28	1.0×	28	1.0×

Additional Model Evaluation

To assess robustness beyond the 8B setting, we evaluate additional models spanning a wider range of parameter scales (14B–72B) and training objectives. Figure 5 reports overall accuracy on FABLE. Performance varies across models but shows no monotonic relationship with parameter count: several larger models do not consistently outperform

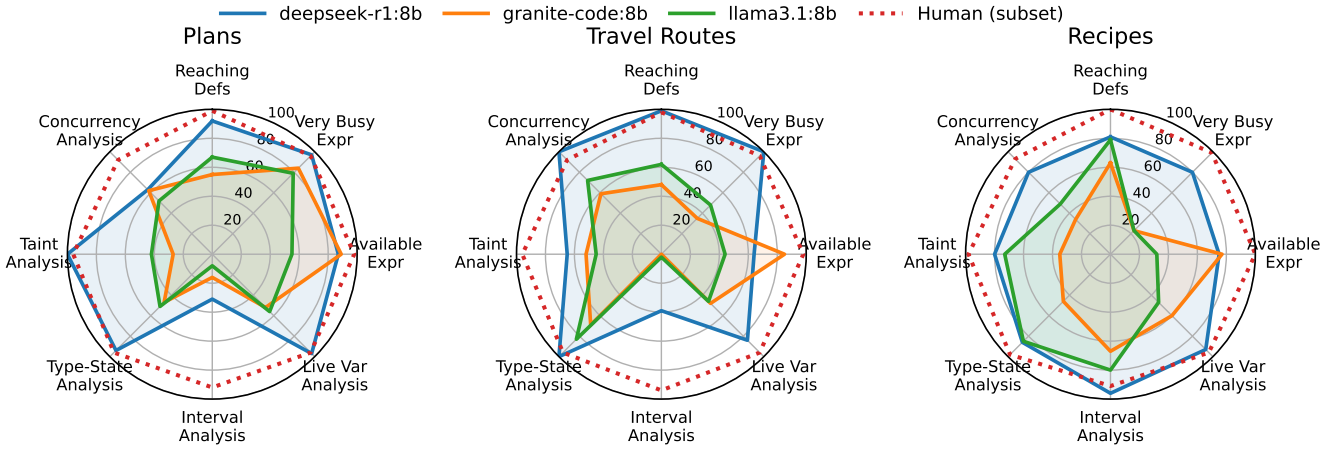

Figure 4: Per-analysis accuracy (%) across domains. The dotted curve shows human performance measured on a balanced 96-question subset, while model curves are computed over the full benchmark.

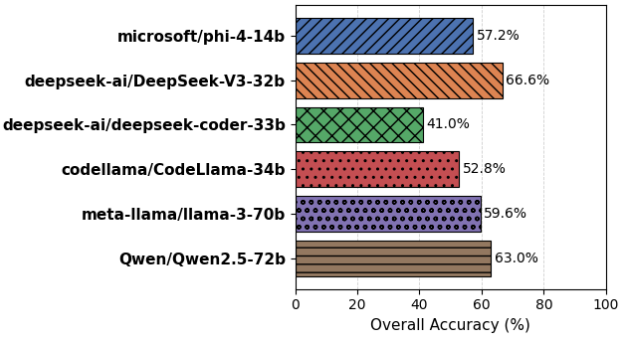

Figure 5: Overall accuracy (%) on FABLE for additional models aggregated over domains and analyses.

smaller ones, and all remain well below human-level accuracy. Across models, interval-based questions are consistently the most challenging, mirroring trends observed in the 8B setting. This suggests that the difficulties captured by FABLE are not artifacts of model size, but persist across architectures and scales. Per-analysis results for the additional models are in Appendix A.9.

Conclusion

We introduce FABLE, a benchmark for evaluating data-flow reasoning in procedural text by adapting classical data-flow analyses from software engineering to natural-language procedures. FABLE enables systematic assessment of how models track entities, dependencies, and constraints across multi-step processes in domains such as automated plans, travel routes, and recipes. Our evaluation shows that although performance varies across models and analyses, all evaluated LLMs exhibit substantial variability across reasoning types and fall well below human performance. In particular, analyses requiring numeric reasoning, temporal constraints, or long-range dependency tracking, such as Interval and Concurrency Analysis, remain challenging across architectures and parameter scales, indicating that increased model size alone is insufficient to address these limitations. FABLE complements existing procedural reasoning benchmarks by providing analysis-driven diagnostics and we hope this benchmark will support future work on representations, and training objectives aimed at more reliable procedural reasoning in LLMs.A discussion of limitations, future directions, and broader impacts is provided in Appendix A.1 and Appendix A.2.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2454027.

References

California Office of Emergency Services (Cal OES). 2021. Major State Government Buildings. Accessed: 2025-03-20.

Community, A.-P. ???? pddl-generators: A Collection of PDDL Problem Generators. Accessed: 2025-05-11.

Dwyer, M. B.; and Clarke, L. A. 1994. Data flow analysis for verifying properties of concurrent programs. *ACM SIGSOFT Software Engineering Notes*, 19(5): 62–75.

Fosdick, L. D.; and Osterweil, L. J. 1976. Data flow analysis in software reliability. *ACM Computing Surveys (CSUR)*, 8(3): 305–330.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Handa, D.; Dolin, P.; Kumbhar, S.; Son, T. C.; and Baral, C. 2025. ActionReasoningBench: Reasoning about Actions with and without Ramification Constraints. *arXiv preprint arXiv:2406.04046*.

He, W.; Huang, C.; Xiao, Z.; and Liu, Y. 2023. Exploring the Capacity of Pretrained Language Models for Reasoning about Actions and Change. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *Proceedings of the 61st Annual Meeting*

of the Association for Computational Linguistics (Volume 1: Long Papers), 4629–4643. Toronto, Canada: Association for Computational Linguistics.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

(HIFLD), H. I. F.-L. D. 2019. USA Hospitals. Accessed: 2025-03-20.

Honnibal, M.; Montani, I.; Van Landeghem, S.; and Boyd, A. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

ICAPS. 2022. International Planning Competitions at International Conference on Automated Planning and Scheduling (ICAPS). In *https://www.icaps-conference.org/competitions/*.

Kennedy, K. 1979. *A survey of data flow analysis techniques*. IBM Thomas J. Watson Research Division.

Khedker, U.; Sanyal, A.; and Sathe, B. 2017. *Data flow analysis: theory and practice*. CRC Press.

Kokel, H.; Katz, M.; Srinivas, K.; and Sohrabi, S. 2025. ACPBench: reasoning about action, change, and planning. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press. ISBN 978-1-57735-897-8.

Kokel, H.; Katz, M.; Srinivas, K.; and Sohrabi, S. 2026. ACPBench Hard: Unrestrained Reasoning about Action, Change, and Planning. arXiv:2503.24378.

Marcondes, F. S.; Gala, A.; Magalhães, R.; Perez de Britto, F.; Durães, D.; and Novais, P. 2025. Using Ollama. In *Natural Language Analytics with Generative Large-Language Models: A Practical Approach with Ollama and Open-Source LLMs*, 23–35. Springer.

McDermott, D. 1978. Planning and acting. *Cognitive Science*, 2(2): 71–109.

National Park Service. 2025. National Register of Historic Places. Accessed: 2025-03-20.

Ning, Q.; Wu, H.; Han, R.; Peng, N.; Gardner, M.; and Roth, D. 2020. TORQUE: A Reading Comprehension Dataset of Temporal Ordering Questions. In Webber, B.; Cohn, T.; He, Y.; and Liu, Y., eds., *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1158–1172. Online: Association for Computational Linguistics.

OpenStreetMap contributors. 2025. OpenStreetMap US Extract. https://www.openstreetmap.org. Data retrieved from https://www.geofabrik.de/.

OurAirports. 2025. OurAirports USA. Accessed: 2025-03-20.

Pollock, L. L.; and Soffa, M. L. 2002. An incremental version of iterative data flow analysis. *IEEE Transactions on Software Engineering*, 15(12): 1537–1549.

Rosen, B. K. 1979. Data flow analysis for procedural languages. *Journal of the ACM (JACM)*, 26(2): 322–344.

Russell, S.; and Norvig, P. 2005. AI a modern approach. *Learning*, 2(3): 4.

Stein, K.; Fišer, D.; Hoffmann, J.; and Koller, A. 2023. AutoPlanBench: Automatically generating benchmarks for LLM planners from PDDL. *arXiv preprint arXiv:2311.09830*.

Tandon, N.; Dalvi, B.; Grus, J.; Yih, W.-t.; Bosselut, A.; and Clark, P. 2018a. Reasoning about Actions and State Changes by Injecting Commonsense Knowledge. In Riloff, E.; Chiang, D.; Hockenmaier, J.; and Tsujii, J., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 57–66. Brussels, Belgium: Association for Computational Linguistics.

Tandon, N.; Mishra, B. D.; Grus, J.; Yih, W.-t.; Bosselut, A.; and Clark, P. 2018b. Reasoning about actions and state changes by injecting commonsense knowledge. *arXiv preprint arXiv:1808.10012*.

Tandon, N.; Mishra, B. D.; Sakaguchi, K.; Bosselut, A.; and Clark, P. 2019. Wiqa: A dataset for" what if..." reasoning over procedural text. *arXiv preprint arXiv:1909.04739*.

Tandon, N.; Sakaguchi, K.; Dalvi, B.; Rajagopal, D.; Clark, P.; Guerquin, M.; Richardson, K.; and Hovy, E. 2020. A Dataset for Tracking Entities in Open Domain Procedural Text. In Webber, B.; Cohn, T.; He, Y.; and Liu, Y., eds., *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6408–6417. Online: Association for Computational Linguistics.

U.S. Bureau of Transportation Statistics. 2025. Airports – Citizen Connect. Accessed: 2025-03-20.

Valhalla, V. ???? Open Source Routing Engine for OpenStreetMap.

Vance, Z. 2017. 140,000 English recipes in computer-readable form with photos and crawl.

Zhang, L.; Xu, H.; Yang, Y.; Zhou, S.; You, W.; Arora, M.; and Callison-Burch, C. 2023. Causal Reasoning of Entities and Events in Procedural Texts. In Vlachos, A.; and Augenstein, I., eds., *Findings of the Association for Computational Linguistics: EACL 2023*, 415–431. Dubrovnik, Croatia: Association for Computational Linguistics.

Zhou, B.; Richardson, K.; Ning, Q.; Khot, T.; Sabharwal, A.; and Roth, D. 2021. Temporal Reasoning on Implicit Events from Distant Supervision. In Toutanova, K.; Rumshisky, A.; Zettlemoyer, L.; Hakkani-Tur, D.; Beltagy, I.; Bethard, S.; Cotterell, R.; Chakraborty, T.; and Zhou, Y., eds., *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1361–1371. Online: Association for Computational Linguistics.

A Appendix

A.1 Limitations & Future Work

While FABLE provides a principled framework for evaluating data-flow reasoning in procedural text, following limitations remain. **(1) Analysis formalization:** Each data-flow analysis is instantiated using a single formal adaptation. Alternative definitions or relaxed variants could capture different reasoning behaviors, but are not explored in the current benchmark.

(2) Prompting strategy: Our evaluation uses a fixed, template-based prompting format. Different prompting strategies, such as few-shot examples or explicit rationales, may affect model performance and are left for future investigation. **(3) Answer format:** The predominance of binary question formats limits visibility into partial correctness or intermediate reasoning errors. Richer response formats could provide deeper diagnostic insight into model failures.

Future Work. Future work will extend FABLE along both breadth and depth. On the breadth side, we plan to incorporate additional procedural domains, such as dialog workflows, scientific protocols, and design processes, to assess generalization across diverse structural and linguistic settings. On the depth side, we aim to increase the number of question–answer instances per domain and analysis to enable finer-grained comparisons and stronger statistical analysis. We also plan to support multiple data-flow formalizations per analysis, enabling systematic study of how alternative reasoning abstractions interact with model behavior. Beyond dataset expansion, future work will explore richer evaluation formats, including free-form explanations, step-level annotations, and interaction-based settings that capture how models revise reasoning under feedback. Together, these directions aim to further position FABLE as a flexible diagnostic framework for studying procedural reasoning in large language models.

A.2 Broader Impacts

The FABLE benchmark aims to advance research on procedural understanding, reasoning diagnostics, and systematic evaluation of large language models. By adapting classical data-flow analyses from software engineering to natural language, FABLE connects techniques from program analysis and NLP, enabling principled evaluation of compositional reasoning, entity tracking, and state evolution in procedural text.

From an applied perspective, improved evaluation of procedural reasoning has implications for domains where correctness and consistency are critical, including healthcare protocols, robotic task execution, and instructional or educational systems. By exposing systematic weaknesses in handling temporal, causal, and state-based dependencies, FABLE can help guide the development of models and evaluation practices that better reflect real-world procedural demands.

The benchmark poses minimal risk of misuse. All data are derived from publicly available sources or procedurally generated artifacts and contain no personal, sensitive, or proprietary information. However, progress in procedural reasoning may increase the deployment of language models in high-stakes settings. To mitigate potential risks from incorrect or overconfident model outputs, future applications should incorporate safeguards such as verification mechanisms, human oversight, or structured validation steps. We hope FABLE encourages more transparent, diagnostic-driven evaluation practices and supports responsible development of procedural reasoning systems.

A.3 Hardware and Execution Details

Experiments for the three 8B models evaluated in the main paper are conducted locally on a single machine equipped with an AMD Ryzen 7 7800X3D CPU, 32 GB RAM, and an NVIDIA RTX 4080 SUPER GPU with 16 GB VRAM. All inference is performed sequentially on a single GPU without batching or caching.

Larger models beyond the 8B setting are evaluated using the IBM Research Inference Tuning Service (RITS) API, which provides access to hosted inference endpoints. For these models, we use the same prompting templates, decoding settings, and majority-vote aggregation protocol as in the local evaluation to ensure comparability. Differences in absolute latency across execution environments are therefore not interpreted as indicative of model efficiency, and inference time analyses in Appendix A.10 focus only on relative trends within the same execution setting.

This separation of execution environments allows controlled evaluation at a fixed scale while enabling broader robustness analysis across larger models without conflating architectural differences with hardware effects.

A.4 Clarifications and Frequently Asked Questions

Q1: Why focus on data-flow analyses rather than end-to-end procedural understanding? FABLE is intended as a diagnostic benchmark rather than an end-to-end generation task. By grounding evaluation in well-defined data-flow analyses, we isolate specific reasoning capabilities such as state persistence, causal dependency, and constraint propagation. This enables fine-grained analysis of model behavior that would be difficult to disentangle in open-ended procedural generation.

Q2: Do deterministic answers in some domain–analysis combinations reduce benchmark difficulty? Some domain–analysis combinations yield deterministic labels due to structural properties of the procedures under the adopted formalization. These cases are retained intentionally, as they test whether models can infer global procedural constraints rather than rely on local lexical cues. Human performance remains consistently high in these settings, while model performance varies substantially, indicating that determinism does not trivialize the task. Deterministic labels represent structural constraints inherent to the domain and are therefore included for benchmark coverage. The inclusion of these deterministic labels reveal models' inability to reason about domain constraints rather than

Q3: Why are most questions binary rather than open-ended? Binary questions reflect the boolean nature of classical data-flow properties and enable controlled comparison across analyses and domains. Despite their binary format, many questions require multi-step reasoning over entity flow, temporal order, and procedural constraints. Richer response formats are a promising extension but are beyond the scope of the current benchmark.

Q4: How is Interval Analysis different from standard numeric reasoning benchmarks? Interval Analysis in FABLE requires inferring numeric ranges grounded in procedural context rather than performing direct arithmetic. Correct answers depend on identifying when constraints are introduced, how long they persist, and which steps modify or invalidate them. This distinguishes the task from standalone numeric reasoning benchmarks.

Q5: Are the observed model failures primarily due to insufficient scale? Evaluation across models ranging from 14B to 72B parameters shows no monotonic relationship between parameter count and performance. While absolute accuracy varies, the relative difficulty of specific analyses—particularly interval-based and temporally grounded reasoning—persists across scales, suggesting that scaling alone does not resolve the challenges captured by FABLE.

Q6: Why include automated planning domains alongside natural language procedures? Automated planning domains provide procedurally precise settings with explicit state transitions, serving as a controlled counterpart to human-authored text. Including both formal and informal procedures allows FABLE to assess whether models can generalize data-flow reasoning across different representational regimes.

Q7: How reliable is the parsing and graph construction process? We manually audited 50 randomly sampled procedures per domain to verify step segmentation, ordering constraints, and entity-flow construction. No systematic errors affecting downstream question generation were observed. Remaining noise reflects inherent ambiguity in natural language rather than parser instability.

A.5 More Information on Data Sources

Recipes To procure 800 recipe QA pairs for the FABLE benchmark, we begin with the English Recipes Dataset (Vance 2017), a publicly released English-language recipe archive collected in June–July 2017. This snapshot aggregates data from four major culinary websites: *Allrecipes.com* ($\approx$ 91,000 recipes), *Epicurious.com* ($\approx$ 34,000), *BBC.co.uk*'s Food section ($\approx$ 10,000), and *Cookstr.com* ($\approx$ 8,000). Each website's data included photographs, raw HTML or JSON payloads, and the scripts used to crawl them. However, since our downstream parsing and question-generation pipeline requires fully structured ingredient lists and coherent step sequences, we elected to restrict our analysis to the Allrecipes.com subset and re-crawled every URL supplied in the subset. This resulted in recipe records that included, among other irrelevant components, a title, an (optional) ingredient list, and a free-form 'instructions list' of natural language steps. To guarantee a minimally informative procedural structure, we discarded any recipe whose instructions contained fewer than three steps, yielding 74,031 recipes. Each instruction was treated as a discrete step s_i, preserving its original ordering and any temporal or causal connectives (e.g. 'meanwhile', 'after').

We then applied our domain-dependent parser (see Section) with spaCy's `en_core_web_trf` model (Honnibal et al. 2020) to extract candidate entities from each step. Noun-chunk extraction was followed by a sequence of rule-based filters: removal of measurement units (e.g. 'cup', 'tsp'), container and tool names when uninformative (e.g. 'bowl', 'spoon'), generic descriptors (e.g. 'mixture', 'amount'), numeric or dimensional patterns (e.g. '9×13', '10–15'), and verb-derived phrases. We also applied derived-entity heuristics to link suffix forms (e.g. 'batter', 'sauce') back to their base entity when appropriate.

We performed a frequency analysis on this initial entity pool to focus our question generation machinery on the most salient domain concepts. We ranked entities by occurrence count and iteratively selected the smallest subset that, under a simple text-mention heuristic, would yield at least 100 questions per data-flow category. A subsequent manual review, consisting of removing residual noisy terms (e.g. 'ingredients', 'rest') and overly generic labels, produced a final vocabulary of domain-relevant entities, encompassing core ingredients (e.g. 'eggs', 'flour'), intermediate preparations (e.g. 'dough', 'glaze') and essential tools or appliances (e.g. 'oven', 'mixer').

Finally, we filtered the 74,031-recipe corpus to retain only those recipes whose extracted entities lie entirely within our curated vocabulary. This ensures that, for every retained recipe, all entities referenced in questions and answers can be unambiguously grounded in the underlying text. The resulting high-quality subset consists of 1,382 recipes, which serve as the basis for all subsequent question-answer generation in the recipes domain.

From the curated set of 1,382 recipes, we instantiated our eight data-flow analysis templates on each procedure, producing an initial, larger pool of 1,600 question–answer instances. We then applied a strict relevance filter, retaining only those questions that explicitly mention at least one entity from our manually vetted vocabulary. The remaining candidates were grouped by analysis type and sorted to maximize diversity of source recipes; from each group, we selected exactly 100 questions, yielding a balanced set of 800 recipe QA pairs.

Each of the 800 selected question–answer pairs underwent a final two-phase curation. In the first phase, two co-authors reviewed every answer for correctness, marking any that appeared inconsistent with the underlying procedural representation. In the second phase, for each flagged item, we algorithmically generated alternative candidate questions that were re-verified until the 100-per-type quota was restored. This ensured that every analysis category in the recipes domain consists of exactly 100 fully validated QA instances.

Travel Routes We construct the travel routes portion of FABLE by assembling 1,500 turn-by-turn navigation procedures, yielding the foundation for 800 question–answer pairs. Specifically, we generate three distinct routes for each of 500 start–destination pairs. Candidate endpoints are drawn from five publicly available datasets: National Park Service historic landmarks (National Park Service 2025), government buildings from the San Diego Supercomputer Center (California Office of Emergency Services (Cal OES) 2021), hospitals from the HIFLD "USA Hospitals" dataset (HIFLD), airports from the Bureau of Transportation Statistics (U.S. Bureau

of Transportation Statistics 2025), and a community-curated airport registry (OurAirports 2025). Location pairs are sampled such that both endpoints lie within the same U.S. state (contiguous U.S.) and the straight-line distance does not exceed 150 km. Pairs that fail to yield three distinct routes are discarded and resampled.

Routes are generated using Valhalla's routing API (Valhalla) over OpenStreetMap data (OpenStreetMap contributors 2025), selected for its fine-grained natural-language maneuver descriptions that include both action types and street names. Each start–destination pair is queried under five cost configurations: default time-optimized routing, highway avoidance, severe maneuver penalties (1,000 s per maneuver), distance-only optimization, and residential-street preference. From the resulting candidates, we retain three routes with distinct textual instructions; pairs with fewer than three unique routes are resampled to ensure uniformity.

To ensure procedural clarity and sequential coherence, we perform both manual auditing and automated normalization. We manually inspect 2% of routes (30 routes across 10 pairs) to verify maneuver correctness under varying conditions (e.g., named vs. unnamed streets, turn variants). We then apply a two-stage normalization process. First, we flag underspecified steps lacking spatial anchors or distance qualifiers and augment them with distances and durations derived from structured route metadata. Second, we merge redundant roundabout instructions into single consolidated maneuvers, preserving timing and road information while eliminating duplication. The final routes are stored in parallel natural-language and structured representations.

Question–answer generation proceeds via domain-specific templates grounded entirely in route text and metadata. For each analysis type, we identify step patterns matching the template constraints and compute gold answers deterministically from the structured representation, without relying on external geographic knowledge. From the full set of generated instances, we select 800 QA pairs (100 per analysis), prioritizing route diversity and question difficulty. As a final quality check, 5% of QA pairs are manually validated for correctness and clarity.

A.6 Running Example: End-to-End Dataset Construction

Figure 2 presents a complete end-to-end walkthrough of the FABLE construction pipeline using a representative recipe procedure. The example illustrates how raw procedural text is transformed into structured graph representations and subsequently used to generate diagnostic data-flow question–answer instances.

Raw procedural text (Panel A). The input consists of a natural-language recipe describing the preparation of *Spicy Garlic Shrimp Pasta*. The procedure specifies an implicit goal (producing the final dish) and a sequence of six ordered steps, including parallel actions (e.g., heating oil while pasta cooks) and entity transformations (e.g., pasta being drained and combined with shrimp). As is typical for human-authored procedures, dependencies between steps are partially implicit and expressed through linguistic cues such as temporal markers (e.g., "meanwhile") rather than explicit control flow.

Parsed procedural structure (Panel B). The domain-dependent parser extracts a structured representation of the procedural text as a tuple $P = (G, S, \preceq)$, where G denotes the goal, $S = \{s_1, \ldots, s_6\}$ is the set of steps, and $\preceq$ encodes ordering constraints. Each step is preserved verbatim in natural language, while the ordering relation captures both sequential execution and concurrency implied by the text. This representation serves as the foundation for subsequent graph construction.

Entity inventory and state tracking (Panel C). From the parsed steps, the parser identifies entities (e.g., water, pasta, oil, shrimp, skillet) and records where each entity is introduced, defined, and used across the procedure. In addition, entity states are tracked as they evolve through transformations such as boiling, cooking, heating, and sautéing. This explicit accounting enables reasoning about whether entities persist, are overwritten, or are reused across steps, which is central to multiple data-flow analyses.

Dependency graph construction (Panel D). Two complementary graphs are constructed from the parsed representation. The step-dependency graph $G_S = (S, E_S)$ captures temporal and causal relationships between steps, including sequential dependencies and parallel execution. The entity-flow graph $G_E = (V, E_E)$ tracks how entities flow across steps, encoding creation, modification, and usage events. For example, the entity *oil* is introduced and heated in Step s_3, combined with shrimp in Step s_4, and subsequently reused when pasta is tossed into the skillet in Step s_6.

Generated question–answer instance (Panel E). Classical data-flow analyses are instantiated over G_S and G_E to generate diagnostic questions. The example shown corresponds to a Reaching Definitions query: *"In Step 6, is the oil from Step 3 being used?"* The answer is computed deterministically by analyzing whether the definition of *oil* at s_3 reaches s_6 without being overwritten along the path $s_3 \rightarrow s_4 \rightarrow s_6$. Since no intervening step redefines or discards the oil, the definition reaches the target, yielding the answer *Yes*. The checklist shown in the figure makes explicit the reasoning steps implied by the analysis, although models are not provided with this intermediate structure during evaluation.

This running example highlights three key properties of FABLE. First, all question–answer pairs are derived from explicit graph semantics rather than manual annotation, ensuring consistency and scalability. Second, multiple data-flow analyses can be applied to the same procedural representation, enabling fine-grained evaluation of different reasoning dimensions. Finally, although this example is drawn from the recipe domain, the same construction pipeline applies uniformly to travel routes and automated plans, differing only in the domain-specific parsing stage.

A.7 Human Performance by Data-Flow Analysis

In addition to the aggregate human performance reported in the main paper (Section), we analyze human accuracy

at the level of individual data-flow analyses to better understand task-specific difficulty. This breakdown provides finer-grained evidence that errors made by models are not attributable to ambiguity in particular analyses.

Table 6 reports human accuracy for each data-flow analysis across the three domains. Performance remains consistently high across all analyses, with no analysis exhibiting systematic difficulty for humans. Notably, analyses that pose the greatest challenge for models—such as *Interval Analysis* and *Concurrency Analysis*—still achieve high human accuracy (91–94% and 92–93%, respectively). This confirms that these analyses require precise procedural reasoning but do not suffer from unclear question formulation.

Overall, the low variance in human performance across analyses indicates that the benchmark definitions are stable and interpretable. Consequently, the large gaps observed between human and model performance in these analyses can be attributed to limitations in model reasoning, rather than artifacts of question generation or domain-specific ambiguity.

Table 6: Human accuracy (%) per data-flow analysis across domains, computed on the same balanced subset used for aggregate human evaluation.

Analysis	**Recipes**	**Travel**	**Plans**
Reaching Definitions	100	98	99
Very Busy Expressions	99	96	96
Available Expressions	100	98	98
Live Variable Analysis	96	96	96
Interval Analysis	91	94	92
Type-State Analysis	98	96	96
Taint Analysis	98	96	96
Concurrency Analysis	93	92	92
Domain Avg.	**96.9**	**95.8**	**95.6**

A.8 Domain-wise Results and Statistical Significance for Baseline Models

Table 7 presents the detailed accuracy scores of each evaluated model across the eight data-flow analyses in the three domains of FABLE. This format allows for fine-grained comparison within and across domains, highlighting how each model handles different procedural reasoning challenges.

To determine whether performance differences across domains are statistically meaningful, we conduct pairwise statistical significance tests for each model, comparing accuracy distributions across the three domains in FABLE. Specifically, we apply the two-tailed Welch's t-test on the per-analysis accuracy values, which accounts for unequal variance and small sample sizes.

Let the domain-wise mean accuracy for a model M be denoted by μ_d, and standard deviation by σ_d, where $d \in \{\text{Plans}, \text{Travel}, \text{Recipes}\}$. Each accuracy vector has length $n = 8$, one value per data-flow analysis.

We compute the t-statistic for each pair of domains (d_1, d_2) using:

$$t = \frac{\mu_{d_1} - \mu_{d_2}}{\sqrt{\frac{\sigma_{d_1}^2}{n} + \frac{\sigma_{d_2}^2}{n}}},$$

with degrees of freedom ν estimated via the Welch–Satterthwaite equation.

For all models, the p-values across domain comparisons are well above the standard threshold of $\alpha = 0.05$, indicating that no statistically significant differences exist in mean accuracy across domains. This result holds despite moderate differences in raw averages (e.g., `granite-code:8b` shows weaker performance on Recipes). However, the absence of statistical significance should not be interpreted as robustness. Rather, it reflects that performance variation across domains is comparable in magnitude to intra-domain variation across analyses. In particular, the high standard deviations (e.g., $\sigma = 25.6$ for `deepseek-r1:8b` in Travel) suggest considerable inconsistency in model behavior across different types of reasoning tasks.

These statistical results support the claim that LLMs tested on FABLE do not exhibit domain-specific reasoning strength or weakness in a statistically robust manner. More importantly, the consistently large variance across analyses highlights the diagnostic value of FABLE in surfacing fine-grained reasoning failures. Rather than punishing models for cross-domain shifts, the benchmark focuses attention on compositional, temporal, and causal generalization—challenges that remain open across all domains.

A.9 Additional Models: Full Per-Analysis Results

This section reports detailed performance for additional large language models evaluated on FABLE, extending the main-paper analysis beyond the 8B parameter setting. The models span a wide range of parameter scales (14B–72B) and training objectives, including general-purpose instruction-tuned models and code-specialized variants.

Table 9 presents overall accuracy as well as accuracy broken down by each data-flow analysis. While overall performance varies across models, the per-analysis breakdown reveals consistent structural patterns. In particular, *Interval Analysis* remains the most challenging category across all models, regardless of scale or architecture. Analyses that rely on explicit state persistence, such as *Reaching Definitions* and *Type-State Analysis*, are comparatively easier, but still exhibit substantial variance across models.

Importantly, larger models do not uniformly outperform smaller ones across analyses, indicating that increased parameter count alone does not guarantee improved procedural data-flow reasoning. These results reinforce the conclusions drawn in the main paper that FABLE captures persistent reasoning challenges that are not resolved through scaling alone.

A.10 Inference Time Analysis

To complement accuracy-based evaluation, we analyze inference latency to assess computational efficiency across model classes. Figure 6, 7, and 8 presents heatmaps of average generation time per 100 prompts for each data-flow analysis and domain, using a logarithmic color scale to accommodate large variance.

Across all domains, `deepseek-r1:8b` incurs significantly higher latency—often exceeding 700 seconds and peaking at over 3300 seconds for *Interval Analysis*

Table 7: Accuracy (%) across eight FABLE data-flow analyses, grouped by domain.

Domain	Analysis	deepseek-r1:8B	granite-code:8B	llama3.1:8B
Plans	Reaching Definitions	92	55	67
	Very Busy Expressions	97	84	79
	Available Expressions	87	89	55
	Live Variable Analysis	97	52	56
	Interval Analysis	31	16	8
	Type-State Analysis	94	47	51
	Taint Analysis	100	27	42
	Concurrency Analysis	63	62	52
Travel Routes	Reaching Definitions	99	48	62
	Very Busy Expressions	100	35	48
	Available Expressions	64	85	44
	Live Variable Analysis	84	48	46
	Interval Analysis	39	0	2
	Type-State Analysis	100	69	83
	Taint Analysis	65	52	45
	Concurrency Analysis	100	59	72
Recipes	Reaching Definitions	81	63	79
	Very Busy Expressions	80	23	23
	Available Expressions	75	77	32
	Live Variable Analysis	93	60	47
	Interval Analysis	96	67	80
	Type-State Analysis	86	46	85
	Taint Analysis	80	35	73
	Concurrency Analysis	80	34	49

Table 8: Welch's t-tests across domains (two-tailed).

Model	P–T	P–R	T–R
deepseek-r1:8b	t=0.13, p=0.897	t=−0.14, p=0.890	t=−0.26, p=0.799
granite-code:8b	t=0.51, p=0.623	t=1.08, p=0.299	t=1.67, p=0.123
llama3.1:8b	t=0.33, p=0.746	t=−0.43, p=0.678	t=−0.67, p=0.514

P=Plans, T=Travel, R=Recipes.

in Travel Routes—highlighting the computational overhead of its reasoning-focused architecture. In contrast, granite-code:8b and llama3.1:8b remain substantially faster, with most analyses completing within 20–75 seconds per 100 prompts. Notably, *Interval Analysis* is consistently the most time-consuming across all models, reflecting the longer output spans and complex step-level parsing required for numeric-range and temporal inference.

These findings emphasize a critical tradeoff: higher-performing models like deepseek-r1:8b offer better accuracy but at an order-of-magnitude increase in compute time. Such latency bottlenecks raise important considerations for real-time deployment and call for future research into efficiency-aware model design and task-specific distillation strategies.

A.11 Prompt Examples for Data-Flow Analyses across Domains

Figures 9–16 present the prompt examples for all eight data-flow analyses in the travel routes domain. Figures 17–24 show the corresponding prompts for the automated plans domain. Finally, Figures 25–32 illustrate the prompts for the recipes domain. Each prompt encapsulates a unique benchmark analysis scenario and is designed to test the model's procedural reasoning capabilities under the constraints of a specific data-flow formalism.

Table 9: Per-analysis accuracy on FABLE for additional models evaluated beyond the 8B setting.

Model	Avail.	Concur.	Interval	Live Var.	Reach. Def.	Taint	Type-State	Very Busy
DeepSeek-V3 (32B)	0.48	0.69	0.45	0.73	0.86	0.65	0.83	0.63
Qwen2.5-72B-Instruct	0.25	0.69	0.39	0.78	0.85	0.66	0.79	0.64
LLaMA-3-70B-Instruct	0.33	0.70	0.34	0.69	0.86	0.55	0.82	0.48
Phi-4 (14B)	0.29	0.66	0.42	0.60	0.82	0.53	0.80	0.45
CodeLLaMA-34B-Instruct	0.12	0.69	0.31	0.58	0.68	0.44	0.78	0.62
DeepSeek-Coder-33B-Instruct	0.14	0.54	0.29	0.36	0.40	0.42	0.58	0.56

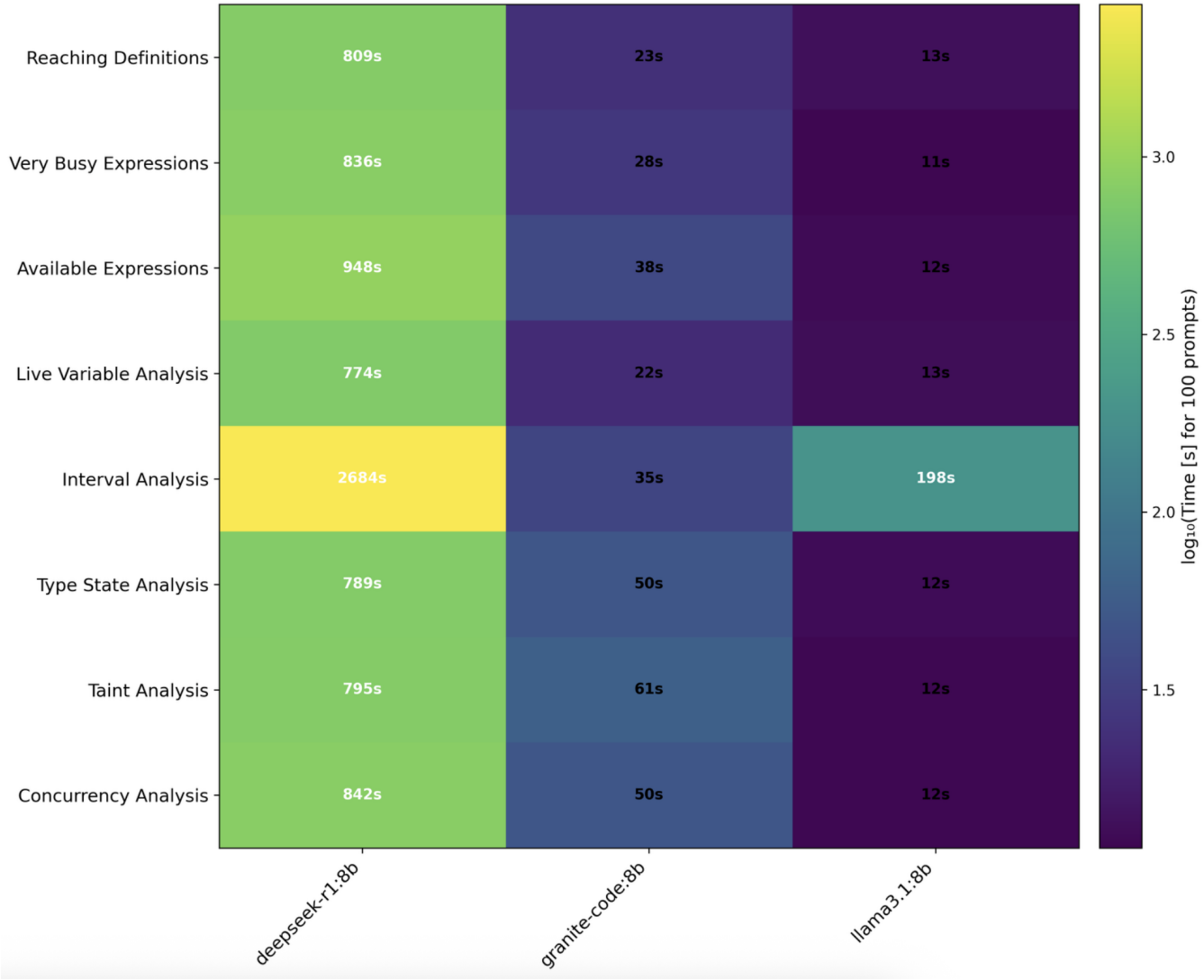

Figure 6: Inference latency heatmap for **Plans** domain. Color intensity indicates log-scale inference time (in seconds per 100 prompts) for each model–analysis pair.

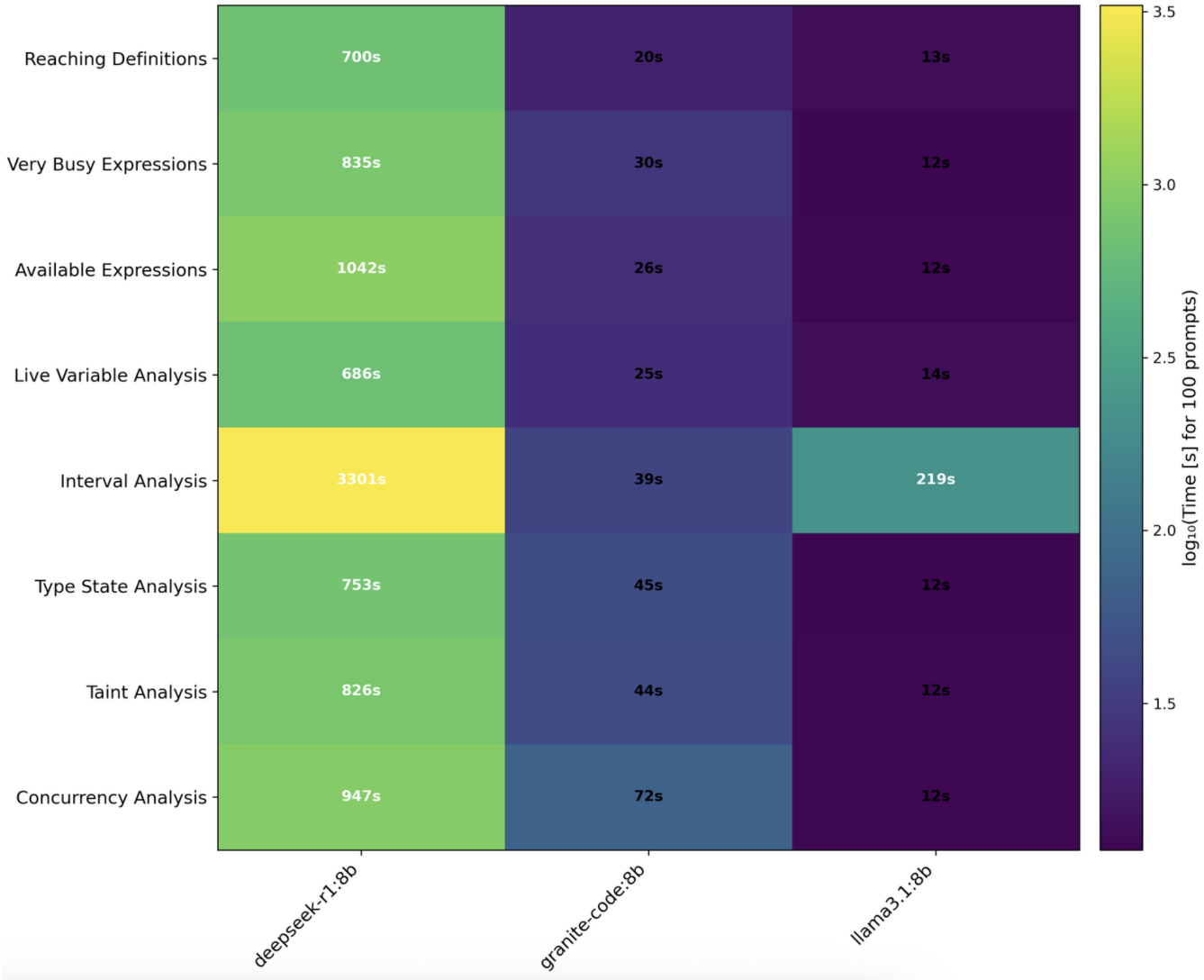

Figure 7: Inference latency heatmap for **Travel Routes** domain. Deep models like `deepseek-r1:8b` incur significantly higher latency, especially on Interval Analysis.

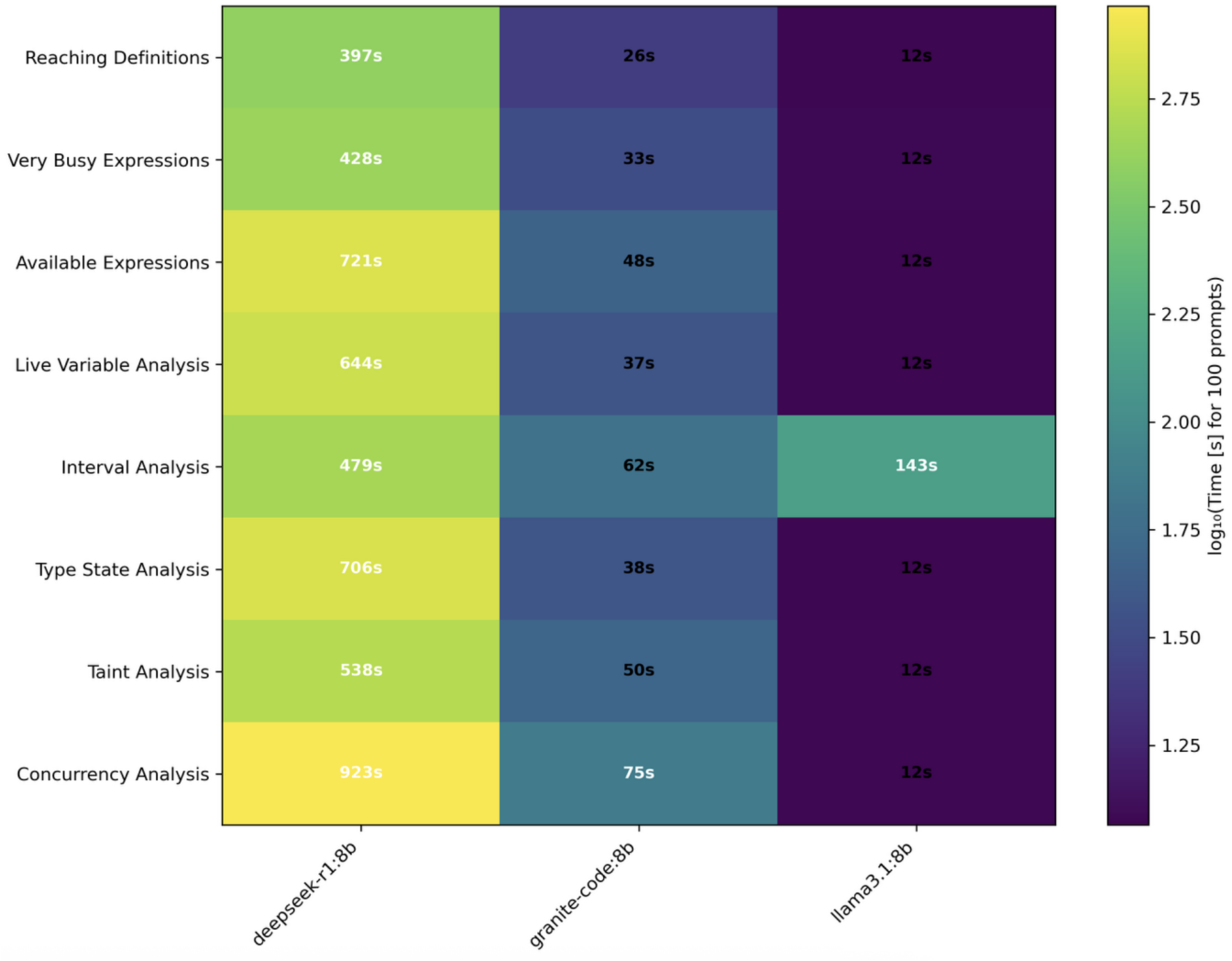

Figure 8: Inference latency heatmap for **Recipes** domain. All models show increased latency for Interval Analysis due to longer and more complex completions.

Travel Routes: Reaching Definitions

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Reaching Definitions
Definition: Tracks which definitions (assignments) of variables can reach a particular program point without being overridden.

Given the following travel planning scenario:
Goal: Coit Memorial Tower (San Francisco, CA) to QUEEN OF THE VALLEY HOSPITAL - NAPA (NAPA, CA)
Route steps:

- Drive north on Telegraph Hill Boulevard.
- Bear left onto Lombard Street.
- Turn left onto Columbus Avenue.
- Turn right onto Broadway.
- Turn right onto US 101 North/Van Ness Avenue.
- Turn left onto US 101 North/Lombard Street. Continue on US 101 North.
- Keep right to take CA 37 toward Napa.
- Take exit 19 on the right onto CA 29 toward Napa.
- Bear left onto CA 29/Sonoma Boulevard. Continue on CA 29.
- Keep left to stay on CA 29.
- Take exit 19 on the right toward Trancas Street/Redwood Road.
- Turn right onto Trancas Street.
- Turn left after 937 meters or 77–94 seconds.
- Turn right after 146 meters or 20–24 seconds.
- Continue for 21 meters or 8–9 seconds.
- Your destination is on the right.

Question: After Step 13, is the driver still on ‘CA 29‘ from Step 9?
Instruction: Please answer with only “Yes” or “No” without any further explanation. Adhere to this rule strictly.

Figure 9: Prompt example for the Reaching Definitions analysis in the Travel Routes domain.

Travel Routes: Very Busy Expressions

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Very Busy Expressions
Definition: An expression is very busy if on every path forward from that point, the expression will be evaluated before any operand changes.

Given the following travel planning scenario:
Goal: McDonald, J. D., House (Fremont, NE) to NORFOLK REGIONAL CENTER (NORFOLK, NE)
Route steps:

- Drive west on East Military Avenue.
- Turn right onto North Broad Street.
- Keep left to take Highway 275.
- Turn right onto Channel Road/NE 35/US 275 Business. Continue on NE 35/US 275 Business.
- Enter the roundabout and take the 2nd exit onto North Victory Road.
- Turn right after 1946 meters or 99–121 seconds.
- Turn right after 207 meters or 29–35 seconds.
- Turn left after 187 meters or 29–36 seconds.
- Turn left after 53 meters or 7–9 seconds.
- Continue for 48 meters or 8–9 seconds.
- You have arrived at your destination.

Question: Is Step 1 a very busy expression?
Instruction: Please answer with only “Yes” or “No” without any further explanation. Adhere to this rule strictly.

Figure 10: Prompt example for the Very Busy Expressions analysis in the Travel Routes domain.

Travel Routes: Available Expressions

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Available Expressions
Definition: An expression is available if it has already been computed and its operands have not been redefined since.

Given the following travel planning scenario:
Goal: VA MEDICAL CENTER - WEST ROXBURY DIVISION (WEST ROXBURY, MA) to Noyes, J.A., House (Cambridge, MA)
Route steps:

- Drive south.
- Turn left after 117 meters or 19–23 seconds.
- Turn right onto Spring Street.
- Turn right onto Veterans of Foreign Wars Parkway.
- Bear left onto Baker Street.
- Bear left onto Dedham Street.
- Turn right onto Parker Street.
- Keep left to take Centre Street.
- Turn left onto Brattle Street.
- Turn left onto Fayerweather Street.
- Turn right onto Reservoir Street.
- Turn right onto Highland Street.
- Continue for 400 meters or 44–54 seconds.
- Your destination is on the left.

Question: Is Step 11 available in Step 12?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 11: Prompt example for the Available Expressions analysis in the Travel Routes domain.

Travel Routes: Type-State Analysis

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Type-State Analysis
Definition: Ensures an object is used only in valid states (e.g., a file must be opened before reading).

Given the following travel planning scenario:
Goal: Zanesville Municipal Airport (Zanesville, OH) to The Ohio State University Airport - Don Scott Field (Columbus, OH)
Route steps:

- Drive northeast on Ritchey Parkway.
- Turn left onto Air Park Drive.
- Turn right onto Airport Road.
- Turn left to take the I 70 ramp.
- Keep right to take exit 108 onto I 270 North.
- Keep right to take I 270 North.
- Keep left to take I 270 North toward Main Street/Whitehall.
- Keep left to take I 270 North/Jack Nicklaus Freeway.
- Keep left to stay on Jack Nicklaus Freeway.
- Take exit 20 on the right toward Sawmill Road.
- Keep left to take Sawmill Road/CR 70 toward Sawmill Road South.
- Turn left after 3426 meters or 178–217 seconds.
- Continue for 1238 meters or 400–489 seconds.
- Your destination is on the left.

Question: Can Step 5 be performed without completing Step 4?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 12: Prompt example for the Type-State Analysis in the Travel Routes domain.

Travel Routes: Live Variable Analysis

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Live Variable Analysis
Definition: A variable is live if its value will be used again in the future (i.e., before being overwritten or discarded).

Given the following travel planning scenario:
Goal: Hartford Club (Hartford, CT) to Igor I Sikorsky Memorial Airport (Bridgeport, CT)
Route steps:

- Drive east on Prospect Street.
- Turn right onto US 5/Main Street.
- Turn right onto US 5/East River Drive.
- Turn left to take the US 5 ramp.
- Take exit 86 on the right onto I 91 South toward New Haven/New York City.
- Take exit 17 on the right onto CT 15 South/Wilbur Cross Parkway.
- Keep left to take CT 15 South/Wilbur Cross Parkway.
- Take exit 37 on the right toward I 95/US 1/Milford.
- Take exit 2B on the right onto I 95 South toward Bridgeport.
- Take exit 30 on the right onto CT 113 toward Surf Avenue.
- Turn left onto Surf Avenue.
- Turn left onto Lordship Boulevard/CT 113.
- Turn right to stay on Lordship Boulevard/CT 113.
- Turn left onto Great Meadow Road.
- Continue for 622 meters or 59–73 seconds.
- Your destination is on the right.

Question: Is the effect from Step 2, the driver getting on US 5, live at any point after Step 2? Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 13: Prompt example for the Live Variable Analysis in the Travel Routes domain.

Travel Routes: Interval Analysis

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Interval Analysis
Definition: Determines the possible numeric ranges (e.g., bounds on variables) at each point in a program.

Given the following travel planning scenario:
Goal: HENRY FORD WEST BLOOMFIELD HOSPITAL (W Bloomfield, MI) to Willow Run Airport (Detroit, MI)
Route steps:

- Drive east.
- Turn right after 221 meters or 29–35 seconds.
- Turn left after 24 meters or 8–10 seconds.
- Turn left onto West Maple Road.
- Turn left onto Haggerty Road.
- Turn left onto 8 Mile Road.
- Take the I 275 South ramp on the right toward Detroit/Toledo.
- Take exit 20 on the right onto Ecorse Road toward Romulus/Willow Run Airport.
- Turn right onto Ecorse Road.
- Turn left after 7078 meters or 283–346 seconds.
- Turn left after 78 meters or 39–48 seconds.
- Turn right after 3165 meters or 412–504 seconds.
- Continue for 806 meters or 109–134 seconds.
- Your destination is on the right.

Question: What is the time interval between performing Step 9 and Step 13?
Instruction: Please provide the numeric interval in the format [x, y] without any further explanation.

Figure 14: Prompt example for the Interval Analysis in the Travel Routes domain.

Travel Routes: Taint Analysis

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Taint Analysis
Definition: Tracks tainted or untrusted data to ensure it does not reach sensitive areas without sanitization.

Given the following travel planning scenario:
Goal: SAINT FRANCIS HOSPITAL (Roslyn, NY) to NYACK HOSPITAL (Nyack, NY)
Route steps:

- Drive north.
- Turn right after 13 meters or 3 seconds.
- Turn right onto Port Washington Boulevard/NY 101.
- Turn right onto North Service Road.
- Take the I 495 West ramp toward New York.
- Take exit 31N-S on the right toward Whitestone Bridge.
- Keep right to take exit 31N onto CI/Cross Island Parkway/100th Infantry Division Parkway toward Whitestone Bridge.
- Keep right to take exit 33 onto I 295 North toward Throgs Neck Bridge/Bronx/New England.
- Keep left to take Cross Bronx Expressway.
- Keep left toward Upper Level.
- Take exit 74 on the right toward Palisades Parkway.
- Take exit 4 on the right onto US 9W.
- Turn left onto Palisades Boulevard/US 9W. Continue on US 9W.
- Turn left to stay on US 9W.
- Turn right onto Hillside Avenue/US 9W. Continue on US 9W.
- Turn right after 1623 meters or 79–97 seconds.
- Continue for 48 meters or 11–14 seconds.
- You have arrived at your destination.

Question: Does turning left in Step 16 taint the preconditions for the rest of the directions?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 15: Prompt example for the Taint Analysis in the Travel Routes domain.

Travel Routes: Concurrency Analysis

You are an AI analyst being evaluated on data-flow analyses for travel routes.
Benchmark: Concurrency Analysis
Definition: Examines parallel (concurrent) execution paths to detect data races or synchronization issues.

Given the following travel planning scenario:
Goal: House at 220 Walnut Street (Nogales, AZ) to BENSON HOSPITAL (Benson, AZ)
Route steps:

- Drive east on West Oak Street.
- Turn left onto North Grand Avenue/I 19 Bus.
- Make a sharp left to stay on North Grand Avenue/I 19 Bus.
- Turn right onto AZ 82.
- Turn left onto AZ 90/State Route 90. Continue on AZ 90.
- Turn right to take the ramp toward I 10/Benson/Douglas.
- Take exit 303 on the right onto I 10 Business toward AZ 80 East/Tombstone/Douglas.
- Turn right onto South Ocotillo Avenue.
- Turn right after 569 meters or 42–52 seconds.
- Continue for 71 meters or 14–18 seconds.
- Your destination is on the right.

Question: Can Steps 10 and 6 be performed concurrently?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 16: Prompt example for the Concurrency Analysis in the Travel Routes domain.

Table 10: Mean accuracy (%) ± std. dev. across the 8 analyses, by domain.

Model	Plans	Travel	Recipes
`deepseek-r1:8b`	82.6 ± 23.4	81.4 ± 25.6	83.9 ± 7.3
`granite-code:8b`	54.0 ± 22.6	50.8 ± 18.1	43.6 ± 17.6
`llama3.1:8b`	55.0 ± 18.2	50.3 ± 25.0	59.1 ± 22.4

Plans: Reaching Definitions

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Reaching Definitions
Definition: Tracks which definitions (assignments) of variables can reach a particular program point without being overridden.

Given the following planning task:
Goal: `(and (at c0 l3) (at c1 l1) (at c2 l2))`
Plan steps:

1: `board(c2, l1)`
2: `sail(l1, l2)`
3: `debark(c2, l2)`
4: `sail(l2, l3)`
5: `board(c1, l3)`
6: `sail(l3, l1)`
7: `debark(c1, l1)`

Question: In Step 3 (`debark`), is the predicate `('on', 'c2')` potentially from the effect of Step 1 (`board`) being used?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 17: Prompt example for the Reaching Definitions analysis in the Plans domain.

Plans: Very Busy Expressions

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Very Busy Expressions
Definition: An expression is very busy if on every path forward from that point, the expression will be evaluated before any operand changes.

Given the following planning task:
Goal: `(and (at c0 l1) (at c1 l3) (at c2 l2))`
Plan steps:

1: `board(c2, l1)`
2: `sail(l1, l2)`
3: `debark(c2, l2)`
4: `board(c1, l2)`
5: `sail(l2, l3)`
6: `debark(c1, l3)`

Question: Is the action in Step 2 (`sail(l1, l2)`) *very busy* in the sense that its effect `('at-ferry', 'l2')` is used by the next step, Step 3 (`debark(c2, l2)`)?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 18: Prompt example for the Very Busy Expressions analysis in the Plans domain.

Plans: Available Expressions

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Available Expressions
Definition: An expression is available if it has already been computed and its operands have not been redefined since.

Given the following planning task:
Goal: `(and (at package1 s1) (at package2 s3))`
Plan steps:

1: `board-truck(driver2, truck1, s1)`
2: `load-truck(package2, truck1, s1)`
3: `drive-truck(truck1, s1, s2, driver2)`
4: `load-truck(package1, truck1, s2)`
5: `drive-truck(truck1, s2, s1, driver2)`
6: `unload-truck(package1, truck1, s1)`
7: `drive-truck(truck1, s1, s3, driver2)`
8: `unload-truck(package2, truck1, s3)`

Question: Is the effect `('driving', 'driver2', 'truck1')` from Step 1 (`board-truck`) still available for Step 3 (`drive-truck`)?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 19: Prompt example for the Available Expressions analysis in the Plans domain.

Plans: Live Variable Analysis

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Live Variable Analysis
Definition: A variable is live if its value will be used again in the future (i.e., before being overwritten or discarded).

Given the following planning task:
Goal: `(and (at package1 s1) (at package2 s3))`
Plan steps:

1: `board-truck(driver2, truck1, s1)`
2: `load-truck(package2, truck1, s1)`
3: `drive-truck(truck1, s1, s2, driver2)`
4: `load-truck(package1, truck1, s2)`
5: `drive-truck(truck1, s2, s1, driver2)`
6: `unload-truck(package1, truck1, s1)`
7: `drive-truck(truck1, s1, s3, driver2)`
8: `unload-truck(package2, truck1, s3)`

Question: After Step 1 (`board-truck`), is the effect `('driving', 'driver2', 'truck1')` *live* (i.e., needed by a future step like Step 3)?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 20: Prompt example for the Live Variable Analysis in the Plans domain.

Plans: Interval Analysis

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Interval Analysis
Definition: Determines the possible numeric ranges (e.g., bounds on variables) at each point in a program.

Given the following planning task:
Goal: `(and (at c0 l1) (at c1 l3) (at c2 l2))`
Plan steps:

1: `board(c2, l1)`
2: `sail(l1, l2)`
3: `debark(c2, l2)`
4: `board(c1, l2)`
5: `sail(l2, l3)`
6: `debark(c1, l3)`

Question: Based on immediate dependencies in this linear plan, what constraints determine when Step 6 (`debark(c1, l3)`) must occur?
Instruction: Please answer *before*, *after*, or *between which steps* without any further explanation.

Figure 21: Prompt example for the Interval Analysis in the Plans domain.

Plans: Type-State Analysis

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Type-State Analysis
Definition: Ensures an object is used only in valid states (e.g., a file must be opened before reading).

Given the following planning task:
Goal: `(and (on d1 d2) (clear d1) (on d2 peg1) (clear peg2) (clear peg3))`
Plan steps:

1: `move(d1, d2, peg3)`
2: `move(d2, peg2, peg1)`
3: `move(d1, peg3, d2)`

Question: If Step 1 (`move`) were skipped, would Step 2 (`move`) become invalid?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 22: Prompt example for the Type-State Analysis in the Plans domain.

Plans: Taint Analysis

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Taint Analysis
Definition: Tracks tainted or untrusted data to ensure it does not reach sensitive areas without sanitization.

Given the following planning task:
Goal: `(and (at c0 l1) (at c1 l3) (at c2 l2))`
Plan steps:

1: `board(c2, l1)`
2: `sail(l1, l2)`
3: `debark(c2, l2)`
4: `board(c1, l2)`
5: `sail(l2, l3)`
6: `debark(c1, l3)`

Question: Does any step in this plan interfere with (taint) the preconditions of the immediately following step?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 23: Prompt example for the Taint Analysis in the Plans domain.

Plans: Concurrency Analysis

You are an AI analyst being evaluated on data-flow analyses for planning tasks.
Benchmark: Concurrency Analysis
Definition: Examines parallel (concurrent) execution paths to detect data races or synchronization issues.

Given the following planning task:
Goal: `(and (at c0 l1) (at c1 l3) (at c2 l2))`
Plan steps:

1: `board(c2, l1)`
2: `sail(l1, l2)`
3: `debark(c2, l2)`
4: `board(c1, l2)`
5: `sail(l2, l3)`
6: `debark(c1, l3)`

Question: Based on simple precondition/delete analysis, can Step 4 (`board`) and Step 3 (`debark`) run concurrently?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 24: Prompt example for the Concurrency Analysis in the Plans domain.

Recipes: Reaching Definitions

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Reaching Definitions
Definition: Tracks which definitions (assignments) of variables can reach a particular program point without being overridden.

Given the following recipe scenario:
Goal: Amish Friendship Cake
Steps:

1: Preheat the oven to 350 degrees F (175 degrees C). Grease two 9-inch loaf pans.
2: Mix flour, sourdough starter, sugar, oil, eggs, baking powder, vanilla, and salt together in a bowl. Stir in chopped walnuts. Stir in baking soda and pour batter into the prepared pans.
3: Bake in the preheated oven until a toothpick inserted into the center comes out clean, 45 minutes to 1 hour.

Question: In Step 3, is the oven from Step 1 being used?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 25: Prompt example for the Reaching Definitions analysis in the Recipes domain.

Recipes: Very Busy Expressions

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Very Busy Expressions
Definition: An expression is very busy if on every path forward from that point, the expression will be evaluated before any operand changes.

Given the following recipe scenario:
Goal: Bulgur Wheat Pilaf
Steps:

1: Place the eggs in a saucepan of cold water, bring to a boil, and boil gently for 10 minutes. Drain the eggs and cool under cold running water. Set aside.
2: While the eggs are cooking, heat the oil in a large saucepan over medium-high heat. Add the onion and garlic, and saute for 3 minutes, stirring occasionally. Stir in the ground coriander, cinnamon, turmeric, and chilies (if using). Stir for 1 minute.
3: Add the pinto beans or chickpeas and apricots, and stir to coat them with the spices. Stir in the bulgur and green beans, then pour in enough water to cover by about 1/2 inch. Bring to a boil, then reduce the heat to low. Cover and simmer for 20 minutes, or until all of the liquid has been absorbed.
4: While the bulgur is cooking, shell and slice the eggs. Fluff the bulgur with a fork and season lightly with salt and pepper. Serve hot, garnished with the egg slices and sprinkled with cilantro leaves.

Question: Is `saucepan` from Step 1 used in multiple future steps without being redefined?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 26: Prompt example for the Very Busy Expressions analysis in the Recipes domain.

Recipes: Available Expressions

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Available Expressions
Definition: An expression is available if it has already been computed and its operands have not been redefined since.

Given the following recipe scenario:
Goal: Dark Chocolate Cookies (Gluten Free, Vegan)
Steps:

1: Preheat oven to 350 degrees F (175 degrees C). Line baking sheets with parchment paper.
2: Combine 1 1/2 cups hot water, chia seeds, and flax seeds in a blender; let soak, about 10 minutes. Blend mixture on high until combined, about 1 minute.
3: Combine millet flour, raw sugar, cacao powder, tofu, 1/2 cup warm water, agave syrup, 1/2 cup cacao nibs, almond meal, vanilla extract, baking soda, and salt in a large bowl. Fold in raspberries. Pour blended mixture over flour mixture; mix until batter is combined.
4: Scoop batter out onto baking sheets. Top each scoop with a few cacao nibs.
5: Bake in the preheated oven until firm, about 12 minutes. Transfer to wire racks to cool.

Question: Is oven from Step 1 still available in Step 2?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 27: Prompt example for the Available Expressions analysis in the Recipes domain.

Recipes: Live Variable Analysis

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Live Variable Analysis
Definition: A variable is live if its value will be used again in the future (i.e., before being overwritten or discarded).

Given the following recipe scenario:
Goal: Crab and Cheddar Quiche
Steps:

1: Preheat the oven to 350 degrees F (175 degrees C).
2: Beat eggs, mayonnaise, milk, flour, and 1 teaspoon Old Bay seasoning together in a bowl until smooth and creamy; stir in Cheddar cheese and parsley. Gently fold in crabmeat.
3: Pour crabmeat filling into pie crust; sprinkle 1 pinch Old Bay seasoning over quiche.
4: Bake in the preheated oven until a knife inserted into center comes out clean and top is lightly browned, 40 to 45 minutes.

Question: Is eggs live after Step 3?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 28: Prompt example for the Live Variable Analysis in the Recipes domain.

Recipes: Interval Analysis

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Interval Analysis
Definition: Determines the possible numeric ranges (e.g., bounds on variables) at each point in a program.

Given the following recipe scenario:
Goal: Cornmeal Cinnamon Rolls
Steps:

1: In a saucepan, combine milk and cornmeal. Bring to a boil over medium heat, stirring constantly. Add the sugar, butter and salt; cool to 110 degrees F–115 degrees F. In a mixing bowl, dissolve yeast in warm water. Add the eggs, cornmeal mixture and 2 cups flour; beat until smooth. Stir in enough remaining flour to form a soft dough. Turn onto a floured surface; knead until smooth and elastic, about 6–8 minutes. Place in a greased bowl, turning once to grease top. Cover and let rise in a warm place until doubled, about 1 hour.

2: Meanwhile, place raisins in a saucepan and cover with water. Bring to a boil; remove from heat. Cover and let stand for 15 minutes. Drain; set aside. Punch dough down. Turn onto a lightly floured surface; divide in half. Roll each into a 12-in. x 10-in. rectangle; brush with melted butter. Combine sugar, cinnamon and nutmeg if desired; sprinkle over dough to within 1/2 in. of edges. Sprinkle with raisins. Roll up, jelly-roll style, starting with a long side; pinch seam to seal. Cut each into 12 slices. Place, cut side down, in two greased 13-in. x 9-in. x 2-in. baking pans. Cover and let rise until doubled, about 1 hour. Bake at 375 degrees F for 20–25 minutes or until golden brown. Cool in pans on wire racks.

3: For frosting, combine sugar, butter, cream cheese, extract and enough milk to achieve desired consistency. Spread or drizzle over rolls. Refrigerate leftovers.

Question: What is the last time interval specified in Step 1?
Instruction: Please provide the numeric interval exactly as it appears (e.g., "1/2 hour") without any further explanation.

Figure 29: Prompt example for the Interval Analysis in the Recipes domain.

Recipes: Type-State Analysis

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Type-State Analysis
Definition: Ensures an object is used only in valid states (e.g., a file must be opened before reading).

Given the following recipe scenario:
Goal: Brazil Nut Fruitcake
Steps:

1: Preheat oven to 350 degrees F (175 degrees C). Grease 3 - 8x4 inch loaf pans and line them with parchment or waxed paper.

2: Beat eggs, salt and vanilla together until very light and lemon colored. Stir in sugar, 1 cup flour and baking powder.

3: Place cherries, nuts, and dates into a large bowl. Dust with the remaining 1/2 cup flour. Then stir in sugar mixture. There is very little batter which makes this a very stiff mixture. Mix with hands.

4: Press batter into prepared loaf pans. Bake for 1 hour.

Question: If we skip Step 2, is it still valid to use the `sugar` in Step 3?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 30: Prompt example for the Type-State Analysis in the Recipes domain.

Recipes: Taint Analysis

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Taint Analysis
Definition: Tracks tainted or untrusted data to ensure it does not reach sensitive areas without sanitization.

Given the following recipe scenario:
Goal: Carrot Cake
Steps:

1: Preheat oven to 350 degrees F (175 degrees C). Grease and flour a 9x13 inch pan. Sift together the flour, baking soda, salt and cinnamon. Set aside.
2: In a large bowl, beat eggs, sugar and oil until smooth. Beat in flour mixture. Stir in pureed carrots, pineapple, walnuts and coconut. Pour batter into prepared pan.
3: Bake in the preheated oven for 30 to 40 minutes, or until a toothpick inserted into the center of the cake comes out clean. Allow to cool.

Question: Does using `eggs` in Step 2 introduce a potential safety concern to the recipe?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 31: Prompt example for the Taint Analysis in the Recipes domain.

Recipes: Concurrency Analysis

You are an AI analyst being evaluated on data-flow analyses for cooking recipes.
Benchmark: Concurrency Analysis
Definition: Examines parallel (concurrent) execution paths to detect data races or synchronization issues.

Given the following recipe scenario:
Goal: Classic Pizza Margherita
Steps:

1: Preheat oven to 450 F. Place pizza crust on baking sheet.
2: Spread pesto over pizza crust. Arrange tomatoes over pesto. Sprinkle with cheese and crushed red pepper.
3: Bake for 10 to 12 minutes or until cheese is melted and crust is golden brown. Cut into wedges.

Question: Can we `preheat` (Step 1) and `bake` (Step 3) at the same time?
Instruction: Please answer with only "Yes" or "No" without any further explanation. Adhere to this rule strictly.

Figure 32: Prompt example for the Concurrency Analysis in the Recipes domain.