When Language Models Fail as Execution Systems

Opus Experiment O-01: A Study of Exactness, Conformance, and Determinism

Nesean Crofford

Abstract

Large language models are increasingly proposed not merely as interfaces for generating text, but as systems capable of participating directly in operational workflows. In these settings, the relevant question is not whether a model can produce a plausible answer, but whether it can reliably produce the exact required output, under fixed constraints, in a form that downstream systems can safely consume.

This paper evaluates that question through a controlled study of 1,260 live executions across five frontier language models and one deterministic execution system, using twelve policy scenarios derived from four operational workflows: invoice discrepancy, refund policy, conditional credit, and insurance adjudication. Each scenario has a single unambiguous correct output and is evaluated using strict exact-match criteria.

Across all evaluated language models, exact-match reliability remains low, with no system exceeding a 25% pass rate. The dominant failure mode is not necessarily incorrect reasoning, but execution non-conformance: in a large share of cases, models produce outputs that are semantically reasonable or partially correct while still failing to emit valid structured artifacts. In addition, repeated executions on identical inputs yield non-identical outputs, indicating systematic non-determinism at the system boundary. By contrast, the deterministic reference system produces exact and stable outputs across all runs.

These findings identify a structural gap between reasoning capability and execution reliability. Language models may often solve the underlying task, but their probabilistic generation process introduces representation errors, variability, and conformance failures that prevent them from functioning as reliable execution systems. The results suggest that the limitations observed are not merely benchmark shortcomings or prompt engineering problems, but reflect a deeper mismatch between probabilistic generation and the requirements of operational software.

1 Introduction

The first generation of large language model applications was defined by conversation. Users supplied prompts, models returned text, and evaluation was typically qualitative: whether the answer was useful, persuasive, informative, or aligned with user intent. In that setting, approximate correctness was often sufficient.

A different standard applies when language models are placed inside operational systems. In financial workflows, claims processing, policy adjudication, or execution-oriented enterprise automation, outputs are no longer consumed only by humans. They are often consumed by downstream software systems that expect exact field values, valid enumerations, stable schemas, and reproducible behavior. In these environments, a semantically similar answer is not equivalent to a correct answer. A system that produces the "right idea" in the wrong representation is still a failing system.

This paper studies that distinction. The question under evaluation is not whether frontier language models are capable of reasoning about bounded policy tasks. The question is whether they can reliably serve as execution systems for those tasks.

Execution systems must satisfy at least three properties:

- **Exactness**: outputs must match a required specification precisely.
- **Conformance**: outputs must obey a fixed schema and controlled vocabulary.
- **Determinism**: identical inputs must produce identical outputs.

These are not unusually strict requirements. They are standard requirements for software systems whose outputs feed payments, claims systems, audit trails, or workflow engines. The central issue, therefore, is whether probabilistic language models can satisfy these requirements consistently enough to be trusted as operational infrastructure.

To evaluate this, we conduct a controlled study of 1,260 live executions across five frontier language models and one deterministic reference execution system. The experiment covers twelve scenarios derived from four structured workflows. Each scenario has a single unambiguous oracle output, and every run is evaluated using strict exact-match criteria rather than semantic equivalence.

The central finding of this paper is simple: language models can often solve these tasks, but cannot reliably execute them.

This claim does not depend on showing that models are unintelligent, nor on claiming that they fail every reasoning task. In fact, one of the strongest findings in the paper is that many failures occur in cases where models appear to understand the problem and even produce the correct decision and numeric value, but still fail to emit a valid execution artifact. This gap between solving and executing is the core result.

The contribution of this paper is therefore threefold:

1. an empirical evaluation of frontier language models under execution-oriented constraints,
2. a decomposition of failures into reasoning failures and execution fidelity failures, and
3. a comparison against a deterministic execution reference system that highlights the distinction between probabilistic generation and operational execution.

The aim of this work is deliberately narrow. It is not to prove that language models are useless, nor to claim that deterministic systems solve all future operational tasks. It is to establish that there is a meaningful architectural distinction between systems that generate plausible answers and systems that produce exact, deterministic execution artifacts. That distinction is foundational if AI is to move from conversation into reliable operational infrastructure.

2 Related Work

Existing language model evaluation has focused primarily on capability benchmarks. Datasets such as **MMLU** (Hendrycks et al., 2021) and **BIG-Bench** (Srivastava et al., 2022) measure performance

across a broad range of reasoning and knowledge tasks, while more recent work has emphasized code generation, mathematical reasoning, and chain-of-thought performance. These benchmarks are valuable for understanding model competence, but they typically assess outputs using semantic or task-level correctness rather than execution-level conformance.

Agent-oriented benchmarks such as **AgentBench** (Liu et al., 2023) and subsequent tool-use evaluations extend this line of work by measuring whether a model can complete a multi-step task using external tools or APIs. However, these evaluations still focus primarily on task success or broad outcome quality, rather than on whether the resulting artifacts satisfy the exact and deterministic requirements of operational systems.

A separate line of work examines structured outputs, including JSON mode, tool calling, and function invocation protocols. These interfaces improve output regularity, but most evaluations of them still emphasize syntactic validity, task completion, or coarse schema adherence rather than exact-match execution reliability across repeated runs.

This paper sits at a different layer of the stack. It is not primarily a capability benchmark, nor an agent benchmark, nor a prompt formatting study. It is an evaluation of *execution reliability*: whether a system can repeatedly produce the exact operational artifact required by a workflow.

This distinction matters because execution reliability imposes a stricter and different requirement than semantic correctness. A model may be able to reason about a policy, describe the correct rationale in prose, or even make the correct decision, while still failing to produce a valid artifact that a downstream system can consume. That failure is operational, not merely stylistic.

Figure 1: Deterministic execution vs probabilistic generation

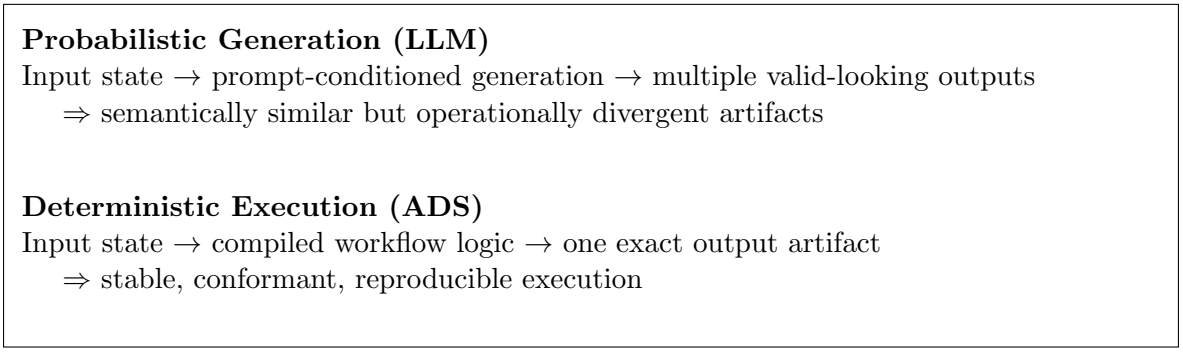

Figure 1: Conceptual distinction between probabilistic generation and deterministic execution. The core claim of this paper is that this architectural difference manifests directly in operational reliability.

The framing of this paper is therefore closer to runtime reliability than to general intelligence. As illustrated in Figure 1, the relevant comparison is not "which model is smartest," but "which system can produce operationally valid outputs reliably enough to function as software."

3 Experiment Design

3.1 Systems and Study Matrix

Six systems were evaluated:

- Claude Sonnet
- Claude Haiku
- GPT-4o
- GPT-4o-mini
- Gemini 2.5 Flash
- ADS (Alternative Deterministic System), a deterministic execution reference system

The experiment consists of 1,260 live runs across twelve frozen scenarios. These scenarios are distributed across four workflow families:

- invoice discrepancy
- multi-policy refund
- conditional credit
- insurance adjudication

Each workflow contains three variants, yielding twelve total scenarios. Runs are distributed across systems according to a fixed plan designed to provide repeated samples for determinism analysis and failure decomposition.

3.2 Operational Workflows

The four workflow families were chosen because they expose different operational pressures that matter for real systems:

- **Invoice discrepancy** introduces arithmetic, reconciliation, and threshold logic.
- **Refund policy** introduces policy ordering, exceptions, and amount computation.
- **Conditional credit** introduces rule priority, deferral logic, and branch selection.
- **Insurance adjudication** introduces structured denial conditions and controlled output semantics.

The scenarios are intentionally bounded. Each one has a deterministic oracle output and does not require open-ended reasoning or world knowledge. This is important: the experiment is not designed to stress maximal intelligence, but to isolate whether a system can reliably produce exact execution artifacts under constrained policy logic.

3.3 Evaluation Criteria

Every run is evaluated under strict exact-match criteria. A run passes only when the produced output matches the oracle exactly across all required fields.

This exact-match requirement is not a stricter benchmark for its own sake. It reflects the real requirement of systems in which outputs are consumed programmatically rather than interpreted by humans. In such settings, semantically similar outputs are not interchangeable.

Each run is evaluated along three dimensions:

1. **Correctness**: does the output exactly match the required oracle?
2. **Conformance**: does the output adhere to the required structured representation?
3. **Determinism**: does the system reproduce identical outputs under repeated runs?

Failures are decomposed into subtypes:

- `wrong_decision`
- `wrong_amount`
- `missed_defer`
- `wrong_reason_code`

The first three are treated as reasoning-related failures. The final one is treated as an execution fidelity failure: the model may solve the task but still fail to emit the required structured artifact.

Table 1: Experiment structure

Dimension	Value
Systems	5 frontier LLMs + 1 deterministic execution reference system
Workflow families	Invoice discrepancy, refund policy, conditional credit, insurance adjudication
Scenario count	12 frozen scenarios (3 per workflow)
Total runs	1,260
Evaluation standard	Strict exact match against deterministic oracle
Primary properties measured	Exactness, conformance, determinism
Failure decomposition	wrong_decision, wrong_amount, missed_defer, wrong_reason_code

Table 1: High-level structure of Experiment_01.

4 Results

4.1 System Reliability

Across the full study, no language model exceeds a 25% exact-match pass rate. By contrast, ADS produces exact outputs on every run, as shown in Figure 2 and Appendix Table 3.

This gap is categorical rather than incremental. Figure 2 makes clear that the relevant question is not whether one model is modestly better than another, but whether a system crosses the threshold required to be trusted as operational infrastructure.

Put differently, the experiment does not show a narrow quality gap between competing systems. It shows that one class of system reliably crosses the execution threshold and the other does not.

Table 2: System-level exact-match reliability

System	Pass Rate	Notes
ADS	100.0%	Exact and deterministic across all runs
Gemini 2.5 Flash	24.4%	Highest LLM performer, still low reliability
GPT-4o	18.3%	Exact-match reliability remains low
GPT-4o-mini	16.3%	Exact-match reliability remains low
Claude Sonnet	12.8%	Exact-match reliability remains low
Claude Haiku	11.7%	Exact-match reliability remains low

Table 2: System-level exact-match reliability. ADS achieves perfect exact-match reliability, while no evaluated language model exceeds a 25% pass rate.

Figure 2: Exact-match reliability by system

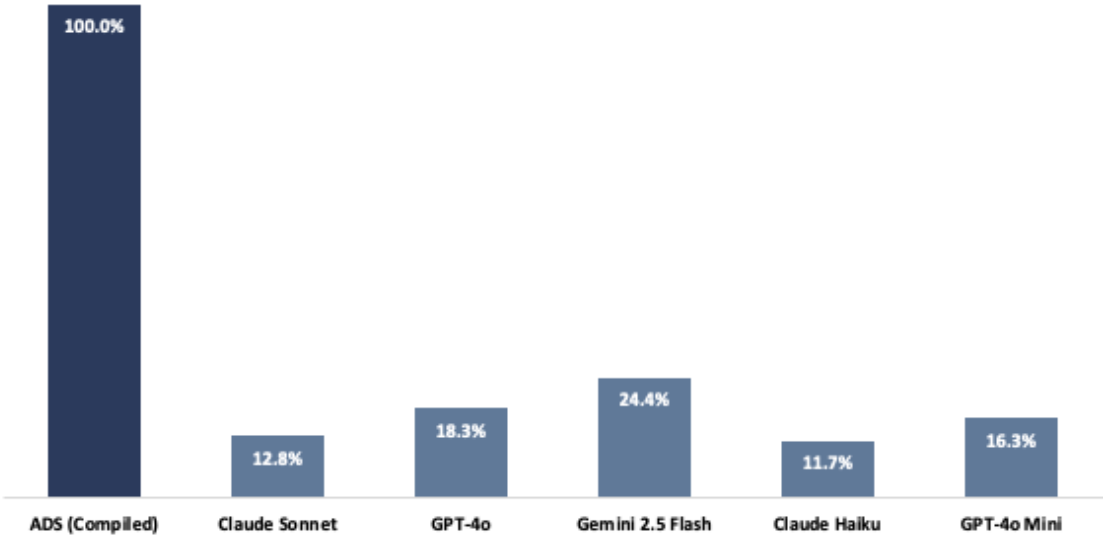

Figure 2: Exact-match pass rates by system. The central observation is not relative ranking among language models, but the categorical separation between deterministic execution and probabilistic generation under execution constraints.

4.2 The Execution Fidelity Gap

A substantial portion of failures occur in cases where the model correctly solves the underlying task but fails to produce a valid execution artifact.

More concretely, many outputs contain both:

- the correct decision, and
- the correct numeric value,

while still failing because the model emits the wrong `reason_code`, an invalid enumeration value, or a representation that does not conform to the required output specification.

This class of failure is central to the paper. It shows that correctness at the level of reasoning does not imply correctness at the level of execution.

From an execution perspective, these outputs are invalid. Downstream systems require exact values and controlled vocabularies, not semantically adjacent alternatives.

This gap between solving and executing is the dominant failure surface observed across all evaluated language models, and it is visible directly in the failure decomposition in Figure 3 and Appendix Table 4.

Figure 3: Reasoning vs execution fidelity failure split

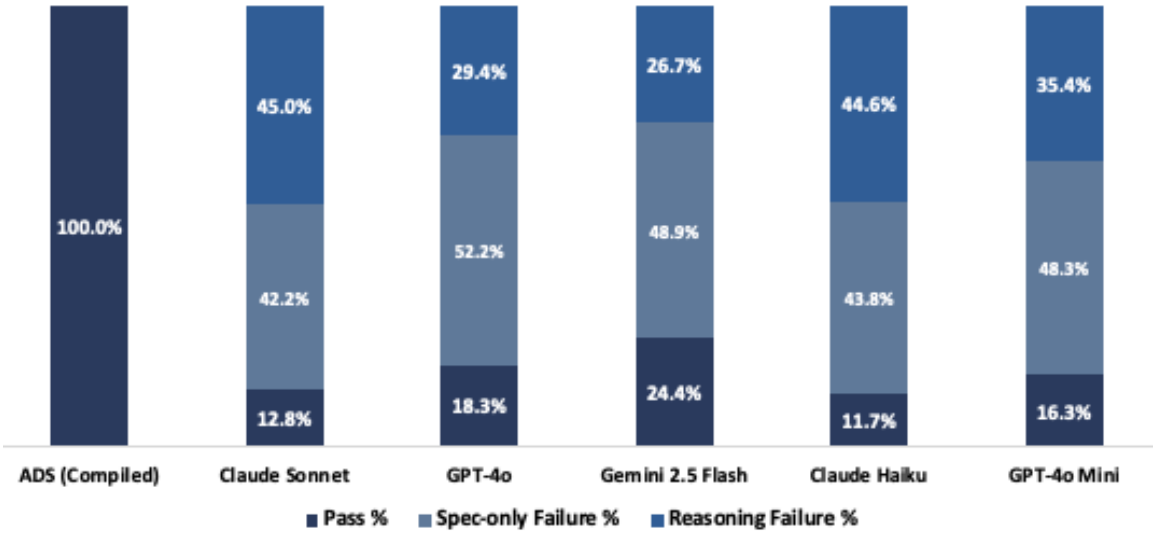

Figure 3: Failure decomposition by system. A substantial portion of failed runs are not wrong in substance, but wrong in representation. This distinction separates reasoning failures from execution fidelity failures.

4.3 Example: Correct Reasoning, Invalid Execution

The practical consequence of the execution fidelity gap can be illustrated with a simple example.

Expected output:

```
{ decision: "deny", reason_code: "invalid_signal", payout_amount: 0 }
```

Model output:

```
{ decision: "deny", reason_code: "override_not_applicable", payout_amount: 0 }
```

Under a semantic or human-evaluated standard, this output may appear acceptable. Under an execution standard, it is invalid.

This example matters because it captures the paper's central distinction. The model has effectively solved the task. It has not executed it correctly.

4.4 Determinism

Language models do not reliably produce identical outputs under repeated identical inputs. The instability is not limited to prose phrasing. It includes different enum values, different numeric outputs, and different structured artifacts for the same scenario.

In execution settings, this behavior is disqualifying. Determinism is not a secondary quality attribute; it is part of the operational contract.

The reference deterministic system exhibits perfect output stability across all runs. This is not a minor improvement. It is a qualitatively different execution profile, as shown in Figure 4 and Appendix Table 5.

Figure 4: Determinism by cohort

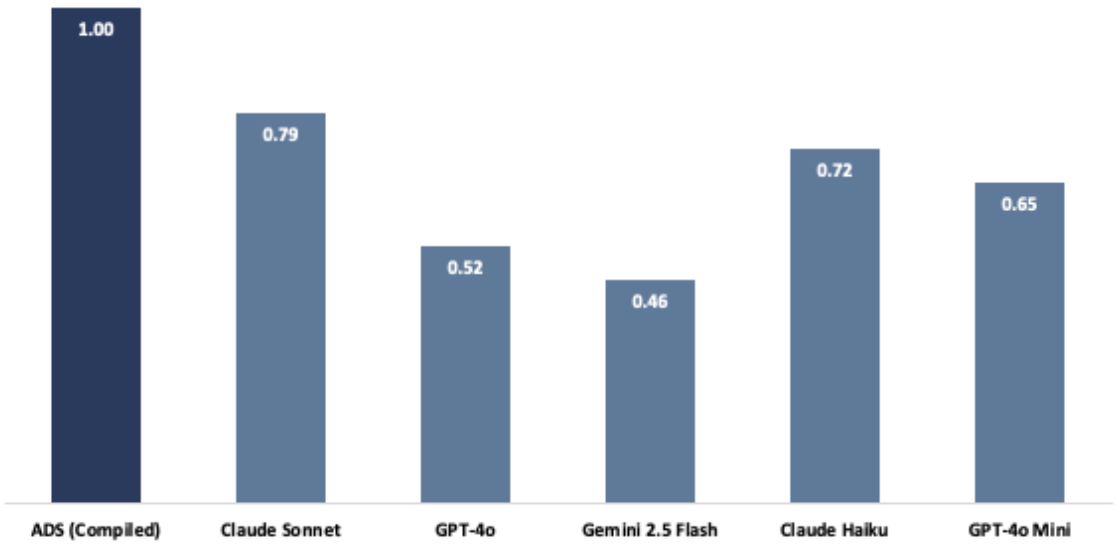

Figure 4: Output stability under repeated identical runs. Determinism separates execution systems from generative systems more clearly than average task performance alone.

4.5 Prompt Sensitivity

System behavior changes based on prompt phrasing, even when the underlying task and inputs are functionally the same.

This variability is not best understood as an optimization issue. It is evidence that system behavior is prompt-conditioned rather than state-determined.

Execution systems should behave consistently under equivalent task representations. Figure 5 shows that language model behavior shifts materially under prompt variation even when task structure remains fixed. This indicates that language models do not function as stable execution surfaces, but as probabilistic generators whose behavior depends materially on how the task is described.

Figure 5: Behavioral instability under prompt variation

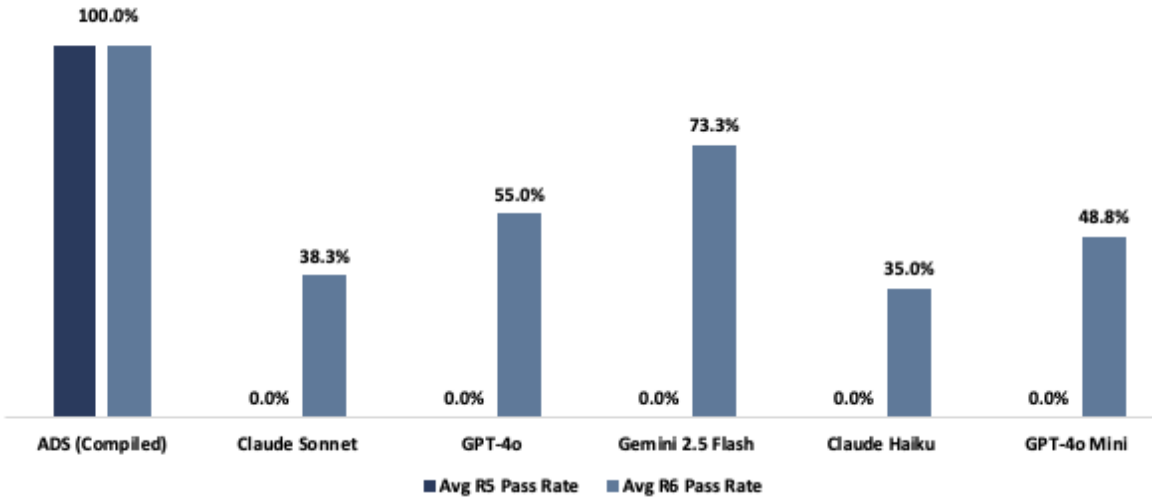

Figure 5: Behavioral instability under prompt variation. The issue is not merely lower performance under weaker prompts; it is that system behavior itself is contingent on phrasing even when task structure remains fixed.

4.6 The "Almost Correct" Failure Mode

A majority of LLM failures are not fully incorrect in a naive sense. Instead, they are structurally close to correct: correct decision, correct amount, invalid output artifact.

This is the most dangerous failure mode in operational systems. It creates outputs that appear plausible under human inspection while still failing system requirements.

In conversational settings, this behavior may be tolerable. In execution settings, it becomes silent operational risk.

"Almost correct" is not a sign that the system is close to reliable execution. As Figure 6 shows, a large share of outputs occupy this approximate-but-invalid region. This is evidence that the system is optimized for producing plausible representations rather than exact artifacts.

Figure 6: Exact, approximate, and invalid outputs

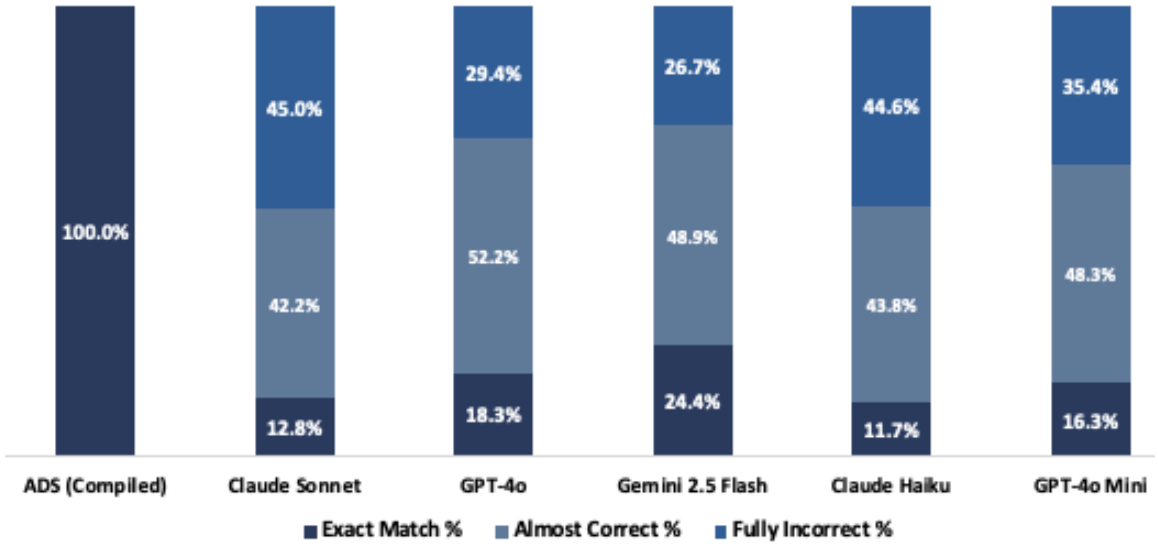

Figure 6: Distribution of exact, approximate, and invalid outputs. The operational risk lies in the large mass of outputs that appear close to correct while remaining unusable as execution artifacts.

5 Key Findings

The experiment yields five core findings.

1. Language models are unreliable execution systems.

No evaluated language model crosses a reliability threshold that would make it suitable as a direct execution system under strict exact-match constraints. This is not a narrow edge case or a single-model failure. It is a consistent pattern across providers and model families.

2. Intelligence is not the bottleneck.

A large share of failures occur after the model has effectively solved the underlying task at the level of decision substance. Decision and amount may both be correct, yet the artifact still fails due to an invalid label or representation. The model demonstrates correct reasoning, but still fails as a system.

3. Language models are non-deterministic at the execution boundary.

Identical inputs do not consistently produce identical outputs. For execution settings, this means that correctness is not reproducible. A system that is sometimes correct and differently wrong on subsequent runs cannot serve as trusted operational infrastructure.

4. Prompt variation does not restore execution reliability.

The experiment shows that behavior changes under prompt variation even when the oracle remains fixed. This does not, in this study, indicate a path to stable execution through prompt refinement alone. It indicates that the system is not stable with respect to representation.

5. Failure modes are orthogonal, not singular.

The observed issues are not reducible to one defect class. Models fail through wrong decisions, wrong amounts, missed deferrals, invalid controlled vocabulary, and non-deterministic variation. This matters because it suggests that the problem is architectural rather than a single fixable weakness.

6 Implications

6.1 Execution vs Generation

The results indicate a structural mismatch between probabilistic generation and execution system requirements.

Execution requires exactness, conformance, and reproducibility. Language models do not reliably satisfy all three simultaneously. This remains true even when they are sufficiently capable to understand the task.

The relevant distinction is therefore not between weak and strong intelligence, but between solving and executing. A system can be capable of solving a task while still being misaligned with the requirements of software execution.

6.2 Why "Close Enough" Fails in Operations

Many arguments for operational LLM deployment implicitly assume that semantically reasonable outputs can be post-processed into valid artifacts. The results here suggest that this view is too optimistic.

Repair layers, validation scaffolding, or post-hoc normalization may reduce some errors, but they do not eliminate the core problem. The execution surface remains probabilistic. Different incorrect outputs appear across identical runs, and conformance failures coexist with substantive reasoning failures. The engineering burden of converting a generative system into a reliable execution system is therefore not incidental; it is central.

6.3 ADS as a Reference Execution Model

The deterministic reference system in this paper is not introduced as a product claim, but as a category reference point.

ADS computes outputs from defined workflow logic rather than generating them through probabilistic text production. As a result, it does not exhibit the same failure surface:

- no conformance drift,
- no output instability,
- no prompt sensitivity,
- and no ambiguity at the artifact boundary.

The purpose of including ADS in the experiment is not to argue that one implementation solves all future operational tasks. It is to show that the failure surface observed in language models is not an unavoidable property of AI systems in general. It is a property of a specific computational abstraction: probabilistic generation used as execution.

7 Limitations

Several limitations should be noted.

First, the workflows are bounded and intentionally constrained. They are designed to isolate execution reliability rather than maximize open-ended complexity. This is a strength for causal interpretation, but it also means that the experiment should not be read as a complete evaluation of all operational AI use cases.

Second, the study uses exact-match evaluation. This is appropriate for execution contexts, but it will appear stricter than human-evaluated semantic scoring. That strictness is intentional; the goal is to evaluate operational validity, not interpretive quality.

Third, the deterministic reference system is used as a comparison point, not as proof of a fully general alternative system architecture. The experiment establishes that deterministic execution is meaningfully different from probabilistic generation in bounded workflow settings. It does not, by itself, prove the full scope of future execution architectures.

Fourth, this work does not evaluate more elaborate agentic scaffolding, multi-agent systems, or advanced output repair layers. Those comparisons are important, but they belong to future experiments rather than this initial study.

8 Future Work

Future work will include two follow up studies to this:

Experiment 02 will introduce compiled execution more explicitly as a distinct computational layer for operational workflows, moving beyond reference comparison and toward direct articulation of the execution architecture.

Experiment 03 will compare deterministic execution against agentic and multi-agent systems, including OpenClaw-style approaches, while taking care not to collapse these categories into a simplistic one-to-one comparison.

A broader research agenda will expand the surface area of evaluation:

- more complex workflows,
- stronger frontier models,
- agentic systems,
- multi-agent orchestration,
- and richer operational scenarios with deeper decision chains.

The aim of this agenda is not merely to show that language models fail today, but to clarify which aspects of those failures are temporary and which are architectural.

9 Conclusion

This study evaluates language models under execution-oriented constraints and finds that exact-output reliability, conformance, and determinism remain inconsistent across models, even in bounded and unambiguous policy tasks.

A substantial portion of failures occur in cases where models correctly solve the task but fail to produce valid execution artifacts. This indicates a structural gap between reasoning and execution.

By contrast, a deterministic execution reference system produces exact and stable outputs across all runs.

The results therefore do not merely suggest that language models need better prompting or more tuning. They suggest that probabilistic generation and execution reliability are governed by different requirements, and may therefore require different system abstractions.

Improving model intelligence alone is unlikely to resolve these issues, because the limitations arise from how outputs are produced, not just how decisions are made.

10 References

Hendrycks, D., Burns, C., Basart, S., et al. (2021).
Measuring Massive Multitask Language Understanding.
arXiv:2009.03300.

Srivastava, A., Rastogi, A., Rao, A., et al. (2022).
Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models.
arXiv:2206.04615.

Liu, X., Yu, Y., Zhang, J., et al. (2023).
AgentBench: Evaluating LLMs as Agents.
arXiv:2308.03688.

Leucker, M., & Schallhart, C. (2009).
A Brief Account of Runtime Verification.
Journal of Logic and Algebraic Programming.

A Appendix: Supplementary Result Tables

This appendix provides supplementary system-level tables referenced throughout the main text. These tables summarize the final aggregate results for exact-match reliability, failure subtype decomposition, and output stability. They are included to make the main claims auditable without overloading the core narrative.

Table 3 reports aggregate exact-match pass rates and the split between reasoning failures and

spec-only failures. Table 4 decomposes behavioral failures into the main subtype categories used throughout the paper. Table 5 reports output stability statistics by system.

Appendix Table 3: System-level exact-match and failure summary

System	Total Runs	Pass %	Reasoning Failure %	Spec-only Failure %
ADS (Compiled)	240	100.0%	0.0%	0.0%
Claude Sonnet	180	12.8%	45.0%	42.2%
GPT-4o	180	18.3%	29.4%	52.2%
Gemini 2.5 Flash	180	24.4%	26.7%	48.9%
Claude Haiku	240	11.7%	44.6%	43.8%
GPT-4o Mini	240	16.3%	35.4%	48.3%

Table 3: Aggregate exact-match reliability and failure-type summary by system. Reasoning failures include wrong-decision, wrong-amount, and missed-defer failures. Spec-only failures denote runs in which the decision and amount were correct but the structured artifact still failed exact-match evaluation.

Appendix Table 4: Failure subtype breakdown by system

System	Wrong Decision	Wrong Amount	Missed Defer	Wrong Reason Code
ADS (Compiled)	0.0%	0.0%	0.0%	0.0%
Claude Sonnet	45.0%	8.3%	8.3%	82.8%
GPT-4o	26.1%	14.4%	8.3%	78.3%
Gemini 2.5 Flash	26.1%	8.9%	8.3%	75.0%
Claude Haiku	37.1%	16.3%	8.3%	75.8%
GPT-4o Mini	34.2%	9.2%	16.7%	82.9%

Table 4: Behavioral failure subtype rates by system. Wrong reason code is the dominant failure surface across all evaluated language models, while wrong-decision remains the largest reasoning-related subtype overall.

Appendix Table 5: Output stability summary by system

System	Avg Stability	Min	Max
ADS (Compiled)	1.00	1.00	1.00
Claude Sonnet	0.79	0.07	1.00
GPT-4o	0.52	0.07	1.00
Gemini 2.5 Flash	0.46	0.07	1.00
Claude Haiku	0.72	0.10	1.00
GPT-4o Mini	0.65	0.05	1.00

Table 5: Output stability summary by system. ADS remains perfectly stable across all cohorts, while all language models exhibit sub-1.00 average stability, including very low minimum cohort stability values.

B Appendix: Representative Run Traces

This appendix provides concrete execution traces drawn from the full 1,260-run dataset. Each example shows an actual language model output alongside the oracle output produced by the deterministic execution system.

These traces are included to make the failure modes described in the main text directly observable. In each case, the task is unambiguous and the correct output is known in advance. The failures therefore reflect not ambiguity in the task, but instability or non-conformance in the execution system.

Trace A: Correct Reasoning, Invalid Execution Artifact

Scenario: Multi-Policy Refund — deny (over limit)
System: Claude Sonnet
Failure type: Spec-only (`wrong_reason_code`)

Oracle (Deterministic Execution)

```
{
  "decision": "deny",
  "refund_amount": 0,
  "reason_code": "defective"
}
```

Model Output

```
{
  "decision": "deny",
  "refund_amount": 0,
  "reason_code": "exceeds_policy_limit"
}
```

Interpretation. The system produces the correct decision and amount, but fails to emit a valid controlled vocabulary value. The generated reason code is semantically plausible but not part of the specification.

Implication. This class of failure is operationally dangerous because it produces outputs that appear correct under human inspection while violating system constraints. It represents a failure of execution fidelity rather than reasoning.

Trace B: Multi-Dimensional Failure

Scenario: Conditional Credit — full credit (defective item)
System: Claude Haiku
Failure type: `wrong_decision` + `wrong_amount` + `wrong_reason_code`

Oracle (Deterministic Execution)

```
{
  "decision": "full_credit",
  "credit_amount": 1500,
  "reason_code": "defective"
}
```

Model Output

```
{
  "decision": "defer",
  "credit_amount": 0,
  "reason_code": "evidence_missing"
}
```

Interpretation. All fields are incorrect. The system selects an invalid decision for the scenario, assigns an incorrect amount, and generates a justification inconsistent with the provided inputs.

Implication. While less subtle than Trace A, this failure illustrates that even bounded and unambiguous scenarios can produce complete breakdowns in reasoning and execution simultaneously.

Trace C: Cross-Model Consistency of Failure Mode

Scenario: Insurance Claim Adjudication — deny (invalid signal)
System: Gemini 2.5 Flash
Failure type: Spec-only (`wrong_reason_code`)

Oracle (Deterministic Execution)

```
{
  "decision": "deny",
  "payout_amount": 0,
  "reason_code": "all_evidence_sufficient"
}
```

Model Output

```
{
  "decision": "deny",
  "payout_amount": 0,
  "reason_code": "override_reason_not_approved"
}
```

Interpretation. The system again produces the correct decision and amount, but fails at the representation layer by emitting a non-spec reason code.

Implication. This failure pattern is not model-specific. The same behavior appears across providers and architectures, indicating that the issue arises from probabilistic generation rather than implementation details.

Synthesis

These traces illustrate three distinct but related failure classes:

- **Execution fidelity failure:** correct reasoning, invalid artifact (Trace A, Trace C)
- **Full reasoning failure:** incorrect decision and values (Trace B)
- **Cross-model consistency:** identical failure patterns across different systems

The most operationally significant failures are those in which the system appears correct while violating the specification. These failures are difficult to detect through manual inspection and cannot be reliably prevented through prompt design alone.

Taken together, these examples reinforce the central claim of the paper: systems that generate outputs probabilistically do not reliably satisfy the requirements of execution systems, even when they demonstrate the capacity to solve the underlying task.