Diagnosing Decision-Boundary Induction in Few-Shot In-Context Learning

Anonymous ACL submission

Abstract

Few-shot in-context learning is usually evaluated by final accuracy, but correct predictions do not guarantee recovery of the intended task boundary. We formulate few-shot ICL as decision-boundary induction and decompose performance into boundary induction and rule execution. Using controlled binary verification tasks with decomposable boundaries, we diagnose which necessary components are behaviorally enforced and, under chain-of-thought, explicitly recognized. Across three tasks and multiple model families, we show that few-shot ICL often induces component-biased verifiers rather than the full target boundary. Stronger models are more robust to this bias, while chain-of-thought reshapes rather than uniformly improves boundary recovery. An explicit-rule intervention further shows that improvements come from both recovering missing boundary components and executing recognized ones more reliably. These results suggest that evaluating few-shot ICL should go beyond aggregate accuracy and directly test boundary recovery. Code is available at anonymous repository

1 Introduction

Few-shot in-context learning (ICL) enables a model to adapt to a task from a small set of labeled demonstrations without parameter updates (Brown et al., 2020). Although ICL is often evaluated by final prediction accuracy, successful adaptation requires more than assigning correct labels to individual queries. The model must infer the underlying decision rule: which input distinctions determine whether an instance should receive one label rather than another. In this sense, few-shot prompting induces a *behavioral decision boundary* under the prompted task.

Recovering this boundary is difficult because realistic task criteria are often only partially specified and hard to fully describe in natural language. For example, if a safety filter is shown negative examples mostly involving explicit violence, it may induce a narrow verifier that rejects violent content while overlooking other unsafe cases not represented in the demonstrations. More generally, a small set of demonstrations provides only partial evidence about the intended boundary, allowing the model to produce labels that are locally consistent with the examples while relying on an incomplete or shortcut boundary.

Final correctness alone therefore does not reveal whether the intended boundary has been recovered. Let A denote the event that the final prediction is correct, and let B denote the event that the model has recovered the intended boundary. Then aggregate accuracy mixes two sources of success: $P(A) = P(A \mid B)P(B) + P(A \mid \neg B)P(\neg B)$. The second term captures cases where the model is correct despite following a different boundary, such as a shortcut or locally sufficient pattern. Thus, treating every correct prediction as evidence of successful task induction can overestimate what the model has actually learned.

Prior work has shown that ICL behavior is highly sensitive to how demonstrations are constructed, including example selection, ordering, label words, calibration, prompt formatting, and input–label mappings (Zhao et al., 2021; Liu et al., 2022; Lu et al., 2022; Min et al., 2022; Dong et al., 2024). A related line of work studies how to retrieve, select, or annotate demonstrations for better ICL performance (Rubin et al., 2022; Su et al., 2023; Qin et al., 2024; Zhang et al., 2025; Wang et al., 2026). Recent surveys further emphasize that demonstration data quality can substantially affect ICL behavior (Schoch and Ji, 2025).

Other studies interpret ICL as task inference, latent concept learning, or implicit algorithmic learning (Xie et al., 2022; Garg et al., 2022; Akyürek et al., 2023; von Oswald et al., 2023). More recent work further analyzes ICL through context-

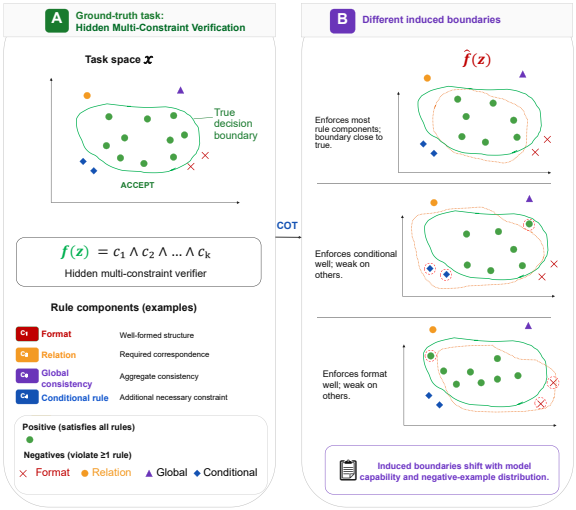

Figure 1: Overview of decision-boundary induction in few-shot ICL. The ground-truth task is defined by a hidden multi-constraint verifier, where an instance is accepted only if all necessary components are satisfied. After observing few-shot demonstrations, the model may induce different partial decision boundaries rather than the full target boundary. These induced boundaries can shift with model capability and the distribution of negative demonstrations.

dependent input–label relationships, latent demonstration variables, structured task hypotheses, and mixtures of different in-context learners (Kossen et al., 2024; Wang et al., 2023a; Li et al., 2024a; Hong et al., 2025). While these works demonstrate that demonstrations strongly affect model behavior, most analyses still rely on final predictions or aggregate performance. They therefore leave underexplored a structural question: how do demonstrations shape which parts of the intended decision boundary are recovered, enforced, weakened, or omitted?

A central difficulty is that the model's internal boundary is not directly observable. We therefore study *observable boundary recovery*. When chain-of-thought (CoT) reasoning is available, we use the reasoning trace as evidence of the verifier explicitly articulated during prediction, without assuming that CoT faithfully reveals the model's internal causal mechanism (Wei et al., 2022; Turpin et al., 2023). This allows us to distinguish two failure modes conflated by final accuracy: boundary omission, where a necessary component is absent from the expressed verifier, and execution failure, where the component is recognized but checked incorrectly. To make this diagnosis possible, we construct controlled binary decision tasks with known and decomposable ground-truth boundaries. We focus on binary decisions because they provide the simplest setting for studying boundary induction, while many multi-class or structured decisions can be viewed as compositions of binary distinctions. Each task contains multiple necessary boundary components: a valid instance must satisfy all components, whereas each negative probe violates one target component while keeping the others valid whenever possible. This design lets us attribute failures to specific components and examine whether demonstrations lead models to enforce some parts of the boundary while ignoring others.

Using this framework, we compare two diagnostic settings. In the implicit few-shot setting, the intended boundary is not stated explicitly and must be inferred from demonstrations. In the explicit-rule setting, all boundary components are directly specified in the instruction. This idealized intervention reduces the need for boundary induction and helps identify whether remaining errors arise from boundary omission or unreliable rule execution. Figure 1 provides an overview of our diagnostic framework. This paper makes three contributions.

1. We formulate boundary recovery in few-shot ICL as a two-stage process of boundary induction and rule execution, enabling component-level diagnosis of which parts of the boundary are recovered, enforced, weakened, or omitted.

2. We show that few-shot ICL can induce component-biased verifiers. Models do not recover all necessary components uniformly: stronger models are more robust to this bias, while CoT reshapes rather than uniformly improves boundary recovery.

3. We show that explicit rule specification improves both boundary recognition and recognition-conditioned execution. By comparing implicit few-shot prompting with an explicit-rule intervention, we show that performance gains come from both better boundary recovery and more reliable execution.

2 Related Work

ICL and demonstration sensitivity. In-context learning enables large language models to adapt from a small number of demonstrations (Brown et al., 2020). Prior work shows that ICL is sensitive to demonstration selection, ordering, label

words, calibration, prompt format, and input–label mappings (Zhao et al., 2021; Liu et al., 2022; Lu et al., 2022; Min et al., 2022; Dong et al., 2024). Other studies interpret ICL as task inference, latent concept learning, implicit algorithmic learning, or context-dependent use of input–label relationships (Xie et al., 2022; Garg et al., 2022; Akyürek et al., 2023; von Oswald et al., 2023; Kossen et al., 2024; Wang et al., 2023a; Li et al., 2024a). These works show that demonstrations strongly affect how models infer and execute tasks in context.

Underspecification, shortcuts, and decision boundaries. Our work is closest to studies showing that few-shot demonstrations can underdetermine the intended task. Si et al. (2023) show that underspecified demonstrations induce systematic feature biases, while other work studies shortcuts, ambiguity, spurious correlations, biased task interpretations, and conflicting rule evidence (Geirhos et al., 2020; Webson and Pavlick, 2022; Jang et al., 2024; Li et al., 2024b; Jiao et al., 2026). Closest in terminology, Zhao et al. (2024) probe decision boundaries induced by ICL and find that they can be irregular and non-smooth. In contrast, we study the component coverage of the induced boundary: which necessary components of a multi-component target boundary are recovered, weakened, or ignored.

Behavioral diagnosis and reasoning traces. Our component-level evaluation relates to behavioral testing and contrast-set evaluation, which use targeted examples to reveal model behavior beyond aggregate accuracy (Ribeiro et al., 2020; Gardner et al., 2020). However, our perturbations diagnose the boundary induced during few-shot ICL itself, rather than serving only as post-hoc robustness tests. We also analyze chain-of-thought traces as observable expressions of the articulated decision rule, without treating them as faithful access to the model's internal mechanism (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2023b; Turpin et al., 2023). Overall, we reframe demonstration sensitivity as boundary induction: demonstrations may reshape not only final predictions, but also which parts of the intended boundary become decision-critical.

3 Diagnostic Framework

3.1 Few-shot Verification with Componentized Boundaries

We study binary decision-making in few-shot in-context learning, where a model predicts the label of a query from a small set of labeled demonstrations (Brown et al., 2020; Dong et al., 2024). We focus on binary decisions because they provide the simplest and most controlled setting for analyzing decision-boundary induction. Since multi-class decisions can often be decomposed into multiple binary distinctions, this setting captures a basic unit of broader ICL behavior.

Formally, the prompt contains a small set of labeled demonstrations

$$\mathcal{S} = \{(z_i, f(z_i))\}_{i=1}^{n},$$

and the model is asked to predict the label of a new query instance z. Here, $f(z) \in \{0, 1\}$, where $f(z) = 1$ means that the instance should be accepted and $f(z) = 0$ means that it should be rejected.

To make the task boundary observable, we formulate verification as a multi-component decision problem. Let

$$\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$$

denote a set of controlled and interpretable task components, where each component is a binary predicate $c_j(z) \in \{0, 1\}$. An instance is valid only when all necessary components are satisfied:

$$f(z) = \bigwedge_{j=1}^{M} c_j(z).$$

Thus, the ground-truth boundary is conjunctive: violating any necessary component should make the instance negative. This structure lets us ask whether the model recovers the full set of necessary conditions, rather than merely learning a high-accuracy shortcut over probes.

This formulation does not assume that real-world tasks are always reducible to a conjunction of symbolic rules. Instead, we use componentized boundaries as a diagnostic tool. By designing tasks in which the true boundary is known and decomposable, we can test whether the model enforces each necessary condition separately. This is similar in spirit to behavioral testing and contrast-set

evaluation, which use targeted perturbations to reveal model behavior beyond aggregate accuracy (Ribeiro et al., 2020; Gardner et al., 2020). In our setting, this design allows us to compare the ground-truth verifier with the model's final predictions, and, when CoT is available, with the verifier explicitly described in the reasoning trace.

3.2 Observable Boundary Recovery

The componentized formulation defines the intended verifier f through a set of necessary components $\mathcal{C}$. However, after observing only a few demonstrations, a model may not recover the same verifier. It may enforce all components, omit some necessary components, or rely on a locally sufficient shortcut.

Since the model's internal decision boundary is not directly observable, we study *observable boundary recovery*. That is, we ask whether the model's observable outputs indicate that the intended boundary components are treated as necessary. We use two types of evidence. The first is final prediction behavior: a model should accept probes that satisfy all components and reject probes that violate any necessary component. This behavioral evidence is available in both direct and CoT settings.

The second type of evidence is the reasoning trace, which is available only in the CoT setting. We do not treat CoT as a faithful readout of the model's hidden causal mechanism. Instead, we use it as an observable expression of the verifier articulated during prediction. This allows us to ask not only whether the final label is correct, but also whether the model explicitly treats the violated component as a necessary condition.

This distinction is important because final accuracy can conflate different sources of success and failure. A model may reject a negative probe because it recognizes the target violation, or it may reject it for an unrelated reason. Conversely, it may mention the correct component but still execute the check incorrectly. We therefore separate final behavioral success from explicit component recognition in the diagnostic definitions below.

3.3 Diagnosing Boundary Recovery

We diagnose boundary recovery using two observable signals: final prediction correctness and, in CoT settings, explicit component recognition.

For each probe z, let

$$A(z) = \mathbf{1}[\hat{y}(z) = y(z)]$$

denote whether the model makes the correct final prediction. For valid probes, $A(z) = 1$ means that the model correctly accepts the candidate. For violation probes, $A(z) = 1$ means that the model correctly rejects the candidate.

For a negative probe whose primary violation corresponds to component c_j, we further define a trace-based recognition variable:

$$B_j(z) = J(\tau(z), c_j),$$

where $\tau(z)$ is the model's reasoning trace and J is an LLM-based judge (Zheng et al., 2023). Here, $B_j(z) = 1$ means that the trace explicitly treats component c_j as a necessary condition for rejection. Merely noticing an anomaly is not sufficient. For example, if the model observes an abnormal delimiter but treats it as an acceptable typo, we do not count this as recognizing the format boundary. This trace-based signal is used only in CoT settings; in direct settings, we report behavioral metrics only.

Recognition–execution decomposition. These two signals allow us to distinguish boundary recognition from rule execution. For a negative probe targeting component c_j, $P(B_j)$ measures whether the model explicitly recovers c_j as part of the expressed verifier. Conditioned on recognition, $P(A \mid B_j)$ measures whether the model correctly executes the corresponding check. After excluding unattributable correct rejections, we define target-attributed success as

$$P(A_j) = P(B_j) \cdot P(A \mid B_j).$$

This decomposition separates two failure modes. If $B_j(z) = 0$, the model omits the violated component from the expressed boundary. If $B_j(z) = 1$ but $A(z) = 0$, the model recognizes the component but fails to execute the check correctly. Thus, final behavior measures whether a component is enforced, while CoT traces help determine whether failures arise from boundary omission or execution failure.

Judge validation and attribution. Because trace-based recognition depends on annotation, and LLM-based evaluators can deviate from human judgments in some settings (Shen et al., 2023; Zheng et al., 2023), we validate the LLM judge

against human labels. For each CoT trace, the judge is given the task description, the target violation component, and the model's reasoning trace, and is asked whether the trace explicitly treats the target component as a necessary condition for rejection. The judge only annotates boundary recognition; final prediction correctness is computed directly from the ground-truth label. The LLM judge was instantiated with GPT-5.4.

We validate the judge on a stratified set of 500 human-annotated traces covering all three tasks, multiple models, components, and correctness patterns. The judge achieves over 97% agreement with human annotations. We also manually audit the two most important non-trivial cases: failures without component recognition and failures after component recognition.

We exclude unattributable correct rejections from trace-based recognition analysis. When a negative probe is correctly rejected for reasons unrelated to the target violation, we keep it in behavioral evaluation but exclude it from recognition analysis, because the trace does not show that the target component is treated as necessary. These annotations are used only for aggregate diagnostics, not as claims about the model's hidden causal mechanism.

3.4 Task Suite

We construct three controlled verification tasks that cover different forms of boundary components. Detailed task illustrations are shown in Figure 4 in Appendix A. The core design principle is *single-component controllability*: each negative probe is generated by perturbing one target component while keeping the other components valid whenever they are independently executable.

Task 1: Prefix-preserving digit-wise transformation. This task verifies whether a candidate string preserves the prefix and applies a digit-wise $+1$ transformation to the numeric body. It contains three components: format validity, prefix copying, and body transformation.

Task 2: Hidden list-function verification. This task requires the candidate annotation to select elements from hidden positions in an input list, preserve the required order, and insert a fixed token. Its components include format validity, content selection, insertion position, and sequence order.

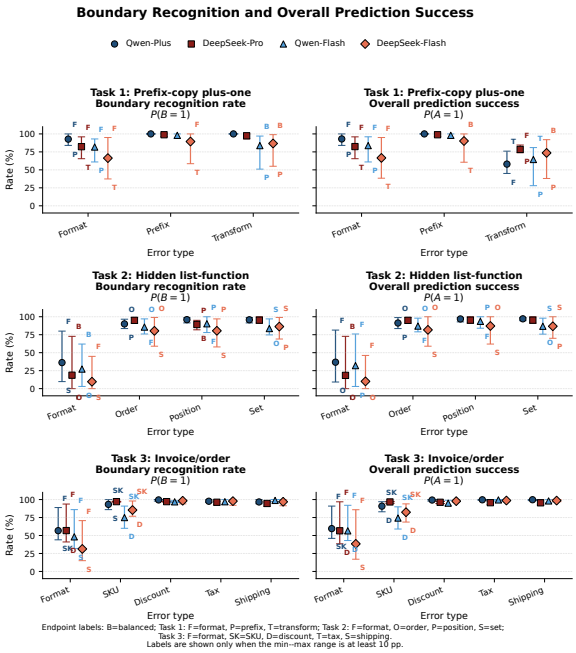

Figure 2: Boundary recognition and prediction success across negative-demonstration distributions. Points show means and vertical bars show min–max ranges; endpoint labels indicate the distributions attaining large extrema.

Task 3: Order–invoice reconciliation. This task models a structured record verification scenario, where a normalized invoice must be checked against a source order. It contains components such as schema validity, field copying, item-code mapping, quantity and line-amount calculation, conditional fee rules, and total consistency.

Together, these tasks span symbolic transformation, hidden structural verification, and application-like structured reconciliation, while repeatedly testing whether models treat format, transformation, structural, and semantic consistency constraints as necessary boundary components.

4 Experiments

4.1 Experimental Setup

Models and reasoning modes. We evaluate four models from two model families: DeepSeek-V4-Flash, DeepSeek-V4-Pro, Qwen3.5-Flash, and Qwen3.5-Plus (DeepSeek AI, 2026; Alibaba Cloud, 2026). These models represent different capability levels within each family. For each model, we consider two reasoning modes: $r \in \{\text{DIRECT}, \text{COT}\}$. In the DIRECT setting, the model is instructed to output only the final labels. In the COT setting, the model is encouraged to reason before giving the

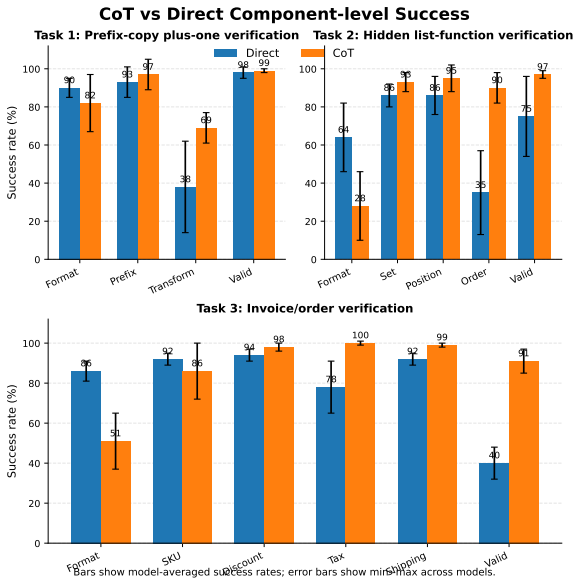

Figure 3: Effect of CoT on component-level success. Bars compare Direct and CoT settings after averaging over models.

final boolean prediction.

Prompt construction and shot sizes. Each prompt contains a small set of labeled demonstrations followed by held-out probes. Templates for the implicit and explicit-rule settings are provided in Appendix B. For a task with component set $\mathcal{C}$, we scale the number of demonstrations with the number of controlled components: $|\mathcal{S}| \in \{2|\mathcal{C}|, 4|\mathcal{C}|, 6|\mathcal{C}|\}$. This scaling keeps the amount of evidence comparable across tasks with different numbers of components. Within each shot size, we use an equal number of positive and negative demonstrations. Negative demonstrations are sampled according to the negative-demonstration distribution described below.

Negative-demonstration distributions. The main experimental variable is the distribution of violation types among negative demonstrations. For each task, let $\mathcal{J}$ denote the set of component-level violation types. We construct the following set of negative-demonstration distributions:

$$\mathcal{D}_{\text{exp}} = \{d_{\text{bal}}\} \cup \{d^{(k)}_{\text{single}} : k \in \mathcal{J}\}.$$

The balanced distribution d_{bal} samples negative demonstrations as evenly as possible from all violation types: $q_j = \frac{1}{|\mathcal{J}|}, \quad j \in \mathcal{J}$, up to integer rounding. Each single-type distribution $d^{(k)}_{\text{single}}$ uses only negative demonstrations that violate component k: $q_k = 1, \quad q_j = 0, \quad \forall j \neq k$. Thus, by changing d, we control which component-level violations are shown as negative evidence in the prompt.

Repetitions and evaluation. For each configuration, defined by task, model, reasoning mode, shot size, and negative-demonstration distribution $d \in \mathcal{D}_{\text{exp}}$, we run 300 independent repetitions. In each repetition, demonstrations and held-out probes are newly sampled, while the task rule and the distribution setting remain fixed. This allows us to measure stable component-level behavior rather than the effect of a particular prompt instance.

Held-out probes include both full-correct cases and component-level violation cases. A prediction is counted as successful if the model accepts a full-correct case or rejects a violation case. Unless otherwise specified, results are first averaged over repetitions within each configuration and then macro-averaged over the three shot sizes.

4.2 RQ1: What Factors Shape the Induced Boundary under Implicit Few-Shot Prompting?

We first study how the boundary induced by implicit few-shot prompting is affected by negative-demonstration distributions, model capability, and reasoning mode. Direct prediction is used only as a behavioral reference for final component-level success.

Negative demonstrations shape the expressed boundary. Figure 2 shows that models do not recover the full conjunctive boundary uniformly. Instead, the components incorporated into the expressed verifier depend strongly on the negative evidence provided in the prompt. The effect is most pronounced for format constraints. Across negative-demonstration distributions, the average min–max range of format recognition reaches 34.0 points in Task 1, 61.7 points in Task 2, and 51.8 points in Task 3. The corresponding ranges in overall success are similarly large: 34.5, 66.0, and 55.3 points. Thus, whether the model treats format as a necessary boundary component depends heavily on whether format violations are represented in the demonstrations.

This distributional sensitivity is not limited to format, but it is component-dependent. Relational and structural components also vary with negative evidence, although less dramatically. For example, in Task 2, order, position, and set show average recognition ranges of 20.5, 20.3, and 17.2 points, respectively. In Task 3, SKU (Stock-Keeping-Unit)

Table 1: Boundary recognition of low-performing components under implicit prompting and explicit rule specification. Each cell reports Implicit → Explicit with Δ in parentheses. We report only components whose implicit boundary-recognition rate is below 90%. Explicit results close to ceiling are rounded to 100.0 for readability. Entries marked with "–" indicate components not included in the low-performing set or unavailable measurements.

Model	Task 1		Task 2				Task 3	
	Format	Transform	Format	Set	Position	Order	Format	SKU
Qwen3.5-Plus	–	–	36.8 → 100.0 (+63.2)	–	–	–	59.6 → 88.1 (+28.5)	–
DeepSeek-Pro	72.2 → 100.0 (+27.8)	80.5 → 100.0 (+19.5)	18.4 → 74.0 (+55.6)	–	–	–	56.6 → 79.3 (+22.7)	–
Qwen3.5-Flash	83.8 → 100.0 (+16.2)	64.5 → 100.0 (+35.5)	32.0 → 89.3 (+57.3)	86.8 → 100.0 (+13.2)	–	87.0 → 100.0 (+13.0)	56.5 → 74.8 (+18.3)	74.3 → 88.0 (+13.7)
DeepSeek-Flash	66.8 → 100.0 (+33.2)	73.2 → 100.0 (+26.8)	10.1 → 25.8 (+15.7)	86.7 → 100.0 (+13.3)	87.1 → 100.0 (+12.9)	81.7 → 100.0 (+18.3)	38.4 → 47.8 (+9.4)	82.4 → 90.0 (+7.6)

mapping varies by 16.8 points in recognition and 18.2 points in overall success. By contrast, discount, tax, and shipping remain close to ceiling across distributions, with average success ranges below 4.0 points. These results indicate that negative demonstrations do more than improve detection of their own violation type. They reshape the operative verifier by determining which necessary components are emphasized, weakened, or omitted.

Stronger models are less sensitive to distributional bias. Model capability modulates this boundary shift. Stronger models are generally more stable across negative-demonstration distributions and more often preserve multiple necessary components. For example, Qwen-Plus remains at 100.0% on Task 1 prefix violations across all distributions, and several Task 3 components such as discount, tax, and shipping are near ceiling for most models. In contrast, Flash-level models show larger ranges and are more likely to collapse toward the component emphasized by the prompt. This suggests that stronger models can better infer underrepresented boundary components from positive evidence and task regularities, whereas weaker models rely more heavily on the observed negative directions.

CoT changes behavioral component success, but Direct does not reveal boundary recognition. We further compare Direct and CoT settings at the behavioral level in Figure 3. Because Direct predictions provide no reasoning trace, this comparison does not tell us whether Direct models explicitly recover the boundary. It only shows how eliciting reasoning changes final component-level success. The pattern is consistent across tasks: CoT improves most reasoning-intensive or semantic components, such as transform, order, tax, shipping, and valid probes, but it consistently reduces success on format constraints. For example, format success drops in all three tasks, with the largest decrease appearing in Task 2. This suggests that CoT encourages models to perform relational, sequential, and arithmetic checks, while making them more tolerant of surface-form deviations.

Table 2: Decomposing transform verification in Task 1. We show DeepSeek-V4-Flash as a representative case; other models exhibit the same qualitative trend.

Setting	$\mathbf{P(B)}$	$\mathbf{P(A \mid B)}$	$\mathbf{P(A)}$
Implicit, length 15	31.00	54.84	17.00
Explicit, length 15	100.00	82.95	82.95
Implicit, length 20	27.00	40.74	11.00
Explicit, length 20	100.00	70.93	70.93

Overall, RQ1 shows that implicit few-shot prompting induces a component-biased verifier rather than a reliably complete boundary. Negative demonstrations shape which components are recognized as necessary, model capability controls the robustness of this recognition under biased evidence, and CoT changes final component-level behavior by improving most non-format components while weakening strict format enforcement. Thus, the central issue is not merely overall accuracy, but which parts of the true boundary become operative under the prompt.

4.3 RQ2: Does Explicit Boundary Specification Recover the Full Boundary and Ensure Correct Execution?

RQ1 shows that, under implicit few-shot prompting, models often induce incomplete or component-biased verifiers. We next ask whether this problem disappears when the intended boundary is explicitly specified. To this end, we consider an idealized explicit-rule setting in which the instruction directly states all necessary boundary components. The explicit-rule setting is not intended to show the

unsurprising fact that stated rules improve performance. Instead, it serves as a diagnostic intervention: by removing the need to infer the boundary from demonstrations, it lets us separate failures caused by boundary omission from failures caused by executing an already stated component. Following the recognition–execution decomposition defined in Section 3.3, this setting allows us to ask two separate questions: whether explicit rules help the model recover missing boundary components, and whether the recovered components can then be executed correctly.

Table 1 first examines boundary recovery. It reports components that are weak under implicit prompting and compares whether they are treated as necessary conditions after explicit rule specification. Overall, explicit rules substantially reduce boundary omission. In Task 1, low-performing format and transform components are raised to 100.0% recognition for most measured models. In Task 2, structural components such as set, position, and order also reach 100.0% once they are explicitly stated. These results indicate that many implicit failures are boundary-recovery failures: when the task boundary must be inferred only from demonstrations, models do not always incorporate all necessary components into the expressed verifier.

However, explicit specification does not make boundary recovery uniformly complete. Format remains difficult in the more structured tasks. For instance, Task 2 format recognition improves substantially but remains below ceiling for several models, and DeepSeek-Flash still reaches only 25.8%. In Task 3, format recognition remains below 90% for all measured models, and SKU recognition is still imperfect for Flash-level models. Thus, explicit rules greatly reduce boundary omission, but whether a component is recovered still depends on both the component type and the model.

We next examine whether recovered components are executed correctly. Task 1 transform verification provides a controlled example for this analysis. We vary digit length to increase execution difficulty while holding the target boundary fixed: both settings require the same digit-wise $+1$ transform, but longer digit bodies place a higher burden on reliable execution. This helps separate failures of boundary induction from failures of post-recognition execution. Using DeepSeek-V4-Flash as a representative case, Table 2 shows that explicit rules increase transform boundary recognition from 31.00% to 100.00% at length 15 and from 27.00% to 100.00% at length 20. This indicates that many implicit errors arise because the model fails to recover transform as a necessary condition. At the same time, explicit rules also improve recognition-conditioned execution: $P(A \mid B)$ increases from 54.84% to 82.95% at length 15, and from 40.74% to 70.93% at length 20. Consequently, final success rises from 17.00% to 82.95% and from 11.00% to 70.93%, respectively. Thus, explicit rules do not merely make the missing component explicit; they also make the recovered transform rule more reliably executable. The other evaluated models show the same qualitative trend, so we report this case for clarity.

The remaining failures are component-specific. SKU errors in Task 3 mainly reflect omitted constraints: because the verifier contains many necessary conditions, models may fail to consistently keep SKU mapping as a required boundary component. Format errors follow a different pattern. Once format is treated as necessary, execution is essentially perfect; failures arise because models often notice surface-form deviations but treat them as minor typos or acceptable notation variants rather than strict rejection conditions. Thus, explicit rules reduce boundary omission and can improve recognition-conditioned execution, but they do not eliminate all errors. Different components fail for different reasons: SKU failures come from forgetting a necessary constraint in a crowded verifier, whereas format failures come from refusing to enforce a surface-form constraint as a hard boundary.

5 Conclusion

This paper reframes few-shot ICL evaluation as decision-boundary recovery rather than final correctness alone. We introduce a component-level diagnostic framework that separates boundary induction from rule execution, enabling failures to be attributed to omitted components or unreliable execution after recognition. Across controlled verification tasks, we show that few-shot demonstrations often induce component-biased verifiers, with model capability, CoT reasoning, and explicit-rule specification shaping which components are recovered and reliably executed. These findings suggest that ICL evaluation should directly diagnose which parts of the intended boundary are recovered, ignored, or unreliably executed.

6 Limitations

Our tasks use controlled, decomposable boundaries, which enable precise diagnosis but simplify the ambiguity and overlapping constraints of real-world tasks. Although the structured invoice task is closer to practical validation scenarios, future work should evaluate boundary recovery in more naturalistic domains such as information extraction, function-call validation, and safety filtering.

Our trace-based analysis treats CoT as an observable expression of the verifier, not as faithful access to the model's internal mechanism. Even with human-validated judging, reasoning traces may rationalize or omit factors that influence the final prediction. Future work should combine trace-based diagnostics with intervention-based or logit-level analyses.

Finally, we study a limited set of models, prompting modes, and negative-demonstration distributions. The explicit-rule setting is also idealized: in real applications, instructions are often partial, noisy, or mixed with demonstrations. Extending the analysis to hybrid and less controlled settings would further clarify how ICL recovers and executes decision boundaries in practice.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2023. What learning algorithm is in-context learning? investigations with linear models. In *International Conference on Learning Representations*.

Alibaba Cloud. 2026. Supported models and capabilities overview: Qwen3.5 models. Alibaba Cloud Model Studio documentation for Qwen3.5-Plus and Qwen3.5-Flash.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

DeepSeek AI. 2026. Deepseek v4 technical documentation. Model card for the DeepSeek-V4 series, including DeepSeek-V4-Pro and DeepSeek-V4-Flash.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, and 7 others. 2020. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online. Association for Computational Linguistics.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. In *Advances in Neural Information Processing Systems*, volume 35, pages 30583–30598.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673.

Giwon Hong, Emile Van Krieken, Edoardo Ponti, Nikolay Malkin, and Pasquale Minervini. 2025. Mixtures of in-context learners. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26332–26351, Vienna, Austria. Association for Computational Linguistics.

Joonwon Jang, Sanghwan Jang, Wonbin Kweon, Minjin Jeon, and Hwanjo Yu. 2024. Rectifying demonstration shortcut in in-context learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4294–4321, Mexico City, Mexico. Association for Computational Linguistics.

Difan Jiao, Di Wang, and Lijie Hu. 2026. Understanding the dynamics of demonstration conflict in in-context learning. *arXiv preprint arXiv:2603.04464*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213.

Jannik Kossen, Yarin Gal, and Tom Rainforth. 2024. In-context learning learns label relationships but is not conventional learning. In *International Conference on Learning Representations*.

Jiaoda Li, Yifan Hou, Mrinmaya Sachan, and Ryan Cotterell. 2024a. What do language models learn in context? the structured task hypothesis. In *Proceedings of the 62nd Annual Meeting of the Association*

for Computational Linguistics (Volume 1: Long Papers), pages 12365–12379, Bangkok, Thailand. Association for Computational Linguistics.

Lvxue Li, Jiaqi Chen, Xinyu Lu, Yaojie Lu, Hongyu Lin, Shuheng Zhou, Huijia Zhu, Weiqiang Wang, Zhongyi Liu, Xianpei Han, and Le Sun. 2024b. Debiasing in-context learning by instructing LLMs how to follow demonstrations. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7203–7215, Bangkok, Thailand. Association for Computational Linguistics.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online. Association for Computational Linguistics.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, Dublin, Ireland. Association for Computational Linguistics.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Chengwei Qin, Aston Zhang, Chen Chen, Anirudh Dagar, and Wenming Ye. 2024. In-context learning with iterative demonstration selection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7441–7455, Miami, Florida, USA. Association for Computational Linguistics.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671, Seattle, United States. Association for Computational Linguistics.

Stephanie Schoch and Yangfeng Ji. 2025. The good, the bad, and the debatable: A survey on the impacts of data for in-context learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 29798–29812, Suzhou, China. Association for Computational Linguistics.

Chenhui Shen, Liying Cheng, Xuan-Phi Nguyen, Yang You, and Lidong Bing. 2023. Large language models are not yet human-level evaluators for abstractive summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 4215–4233, Singapore. Association for Computational Linguistics.

Chenglei Si, Dan Friedman, Nitish Joshi, Shi Feng, Danqi Chen, and He He. 2023. Measuring inductive biases of in-context learning with underspecified demonstrations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11289–11310, Toronto, Canada. Association for Computational Linguistics.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Selective annotation makes language models better few-shot learners. In *International Conference on Learning Representations*.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. 2023. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems*, volume 36.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 35151–35174. PMLR.

Haochun Wang, Chaofen Yang, Jiatong Liu, Jingbo Wang, Zewen Qiang, Sendong Zhao, Bing Qin, and Ting Liu. 2026. Easier to judge than to find: Predicting in-context learning success for demonstration selection. *arXiv preprint arXiv:2605.18512*.

Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. 2023a. Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 15614–15638.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Albert Webson and Ellie Pavlick. 2022. Do prompt-based models really understand the meaning of their

prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2300–2344, Seattle, United States. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. 2023. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2022. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*.

Jianfei Zhang, Bei Li, Bai Jun, Rumei Li, Yanmeng Wang, Chenghua Lin, and Wenge Rong. 2025. Selecting demonstrations for many-shot in-context learning via gradient matching. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11686–11704, Vienna, Austria. Association for Computational Linguistics.

Siyan Zhao, Tung Nguyen, and Aditya Grover. 2024. Probing the decision boundaries of in-context learning in large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 130408–130432.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36.

A Task Illustrations and Capability Coverage

This appendix provides additional details about the controlled task suite used in our experiments. The main paper describes the tasks at a high level, while Figure 4 gives concrete examples of the input–candidate format, the valid transformation, and representative component-level violations. These examples are intended to clarify how each task instantiates a decomposable decision boundary and how negative probes are constructed by targeting one primary component at a time.

Table 3: Capability coverage across tasks (T1–T3). ✓indicates at least one component of that type. Tasks are complementary: T1 targets symbolic transformation, T2 relational structure, and T3 arithmetic and conditional rules.

Capability	**T1**	**T2**	**T3**
Format compliance	✓	✓	✓
String copying	✓		✓
Arithmetic transformation	✓		✓
Set membership		✓	
Positional constraint		✓	
Sequence ordering		✓	
Symbol mapping			✓
Quantity calculation			✓
Conditional fee rules			✓
# Components	3	4	5
Boundary nature	Symbolic	Relational	Arithmetic

Table 3 summarizes the types of capabilities covered by the three tasks. The tasks are not designed to exhaustively cover all possible real-world verification scenarios. Instead, they provide complementary diagnostic settings with increasing structural complexity. Task 1 focuses on symbolic string transformation, Task 2 focuses on hidden relational structure, and Task 3 moves closer to application-like record reconciliation involving copying, mapping, arithmetic calculation, and conditional rules. Across these tasks, the shared design principle is that a candidate is valid only if all required components are satisfied, allowing us to test whether models recover the full boundary or only a subset of its necessary conditions.

B Prompt Templates

This appendix provides the prompt templates used in the implicit few-shot and explicit-rule settings. For each task, the implicit prompt requires the model to infer the labeling rule from demonstrations, whereas the explicit-rule prompt directly states the necessary boundary components in the task instruction. In both settings, the demonstrations and target probes follow the same input–candidate–label format. The CoT setting is implemented as a reasoning option applied to these templates, rather than as a separate task template.

Figure 5 shows the prompt templates for Task 1. The implicit version only describes the surface format and asks the model to infer the hidden rule from examples. The explicit-rule version additionally states the three necessary components: ex-

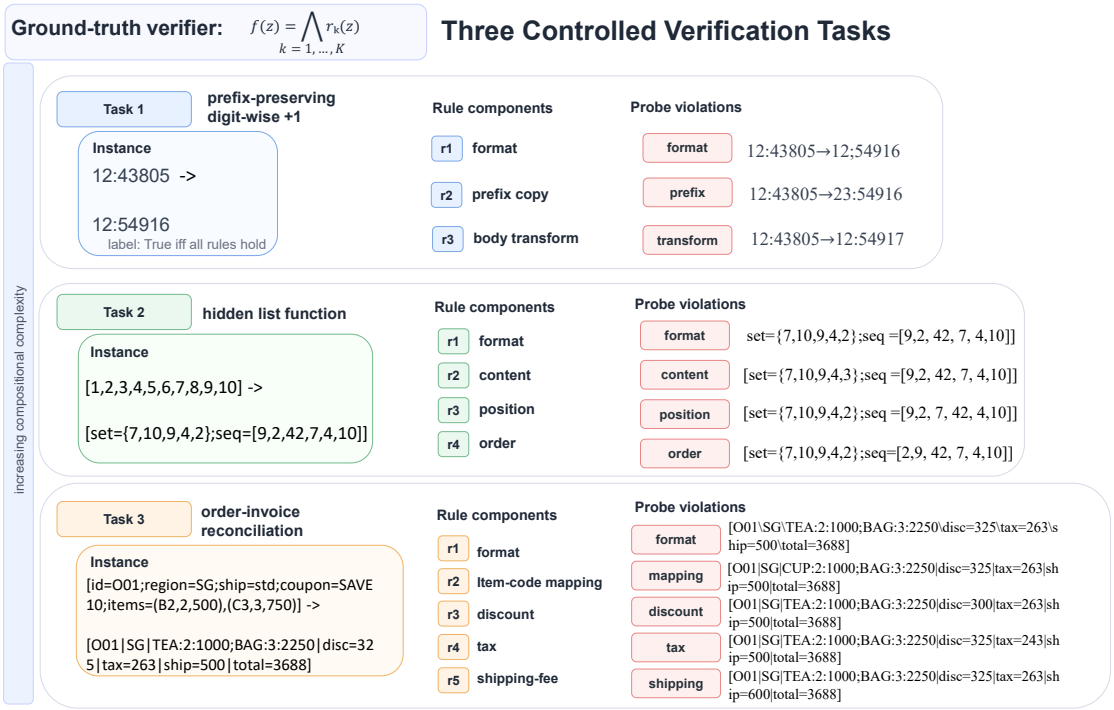

Figure 4: Controlled verification tasks with decomposable boundaries. Each task consists of multiple necessary components, and each negative probe targets one primary component while keeping the other components valid whenever possible. This design enables component-level diagnosis of which conditions are enforced, weakened, or omitted by the model-induced verifier.

act format, prefix preservation, and digit-wise $+1$ transformation without carry.

Figure 6 shows the prompt templates for Task 2. The implicit version describes the task as a hidden list transformation, while the explicit-rule version specifies the required canonical annotation format, set selection, sequence order, fixed-token insertion, and consistency across fields.

Figure 7 shows the prompt templates for Task 3. The implicit version presents the task as an order-to-invoice conversion problem, while the explicit-rule version enumerates the required constraints over record structure, field preservation, product-code mapping, quantity preservation, line-total calculation, item order, discount, tax, shipping, and final total consistency.

Task 1 Prompt Templates: Prefix-Preserving Digit-Wise Transformation

Implicit Prompt

```
Each example has the form source -> candidate = label.

Each source and candidate has a short prefix
followed by a colon and a digit body.

Infer the hidden labeling rule from the examples,
then label the target pair(s).
```

Example

```
ab:123 -> ab:234 = True
ab:123 -> ac:234 = False
ab:123 -> ab:224 = False
```

The rule must be inferred from demonstrations.

Explicit Prompt

```
Each example has the form source -> candidate = label.

Each source and candidate must have the exact format
prefix:digit_body.

The label is True iff ALL conditions hold:

(1) both source and candidate use the exact
    prefix:digit_body format;

(2) the candidate prefix is exactly the same
    as the source prefix;

(3) the candidate digit_body is obtained by adding
    1 to each digit of the source digit_body
    independently, without carry.

Otherwise, the label is False.
```

Example

```
ab:123 -> ab:234 = True
ab:123 -> ac:234 = False
ab:123 -> ab:224 = False
```

Implicit prompts require rule induction; explicit prompts directly enumerate boundary components.

Figure 5: Prompt templates for Task 1: prefix-preserving digit-wise transformation. The implicit prompt requires the model to infer the transformation rule from demonstrations, while the explicit-rule prompt states all necessary boundary components.

Task 2 Prompt Templates: Hidden List-Function Verification

Implicit Prompt

```
Each example has the form source_list ->
candidate_annotation = label.

A pair is True only when the candidate
annotation follows the same hidden decomposed
list transformation.

Infer the hidden labeling rule from the
examples, then label the target pair(s).
```

Example

```
[8, 3, 5, 1] -> ANN[set={3,1};seq=[3,42,1]] = True
[8, 3, 5, 1] -> ANN[set={3,1};seq=[1,42,3]] = False
[8, 3, 5, 1] -> ANN[set={3,5};seq=[3,42,1]] = False
```

The rule must be inferred from demonstrations.

Explicit Prompt

```
Each example has the form source_list ->
candidate_annotation = label.
Each source is a list of values, and each candidate
annotation has the exact format
ANN[set={...};seq=[...]].

The label is True iff ALL conditions hold:
(1) the candidate annotation follows exactly this
    canonical format, using set={...} and seq=[...]
    with comma-separated values;
(2) the selected source positions are 2 and 4 in
    1-based indexing; the set field must contain
    exactly the values from these positions, with
    no missing or extra values;
(3) the seq field must contain the selected values
    in the same position order 2,4;
(4) the fixed token 42 must be inserted at position
    2 of the seq field, using 1-based indexing;
(5) the set field and seq field must be consistent
    with the same selected source values.

Otherwise, the label is False.
```

Example

```
[8, 3, 5, 1] -> ANN[set={3,1};seq=[3,42,1]] = True
[8, 3, 5, 1] -> ANN[set={3,1};seq=[1,42,3]] = False
[8, 3, 5, 1] -> ANN[set={3,5};seq=[3,42,1]] = False
```

Explicit prompts reveal format, selected positions, ordering, token insertion, and cross-field consistency.

Figure 6: Prompt templates for Task 2: hidden list-function verification. The explicit-rule prompt states the decomposed conditions over format, selected set, sequence order, fixed-token insertion, and cross-field consistency.

Task 3 Prompt Templates: Order-Invoice Reconciliation

Implicit Prompt

```
Each example has the form source -> candidate = label.
Each pair describes a conversion from an order to an invoice.
Infer the hidden labeling rule from the examples,
then label the target pair(s).
```

Example

```
ORD[id=1007;region=US;ship=std;coupon=NONE;
     items=(A1,2,300),(B2,1,500)]
  -> INV[1007|US|ALPHA:2:600;BETA:1:500|disc=0|tax=110|
     ship=500|total=1710] = True

ORD[id=1007;region=US;ship=std;coupon=NONE;
     items=(A1,2,300),(B2,1,500)]
  -> INV[1007|US|ALPHA:2:600;BETA:1:400|disc=0|tax=110|
     ship=500|total=1610] = False

The conversion rules must be inferred from demonstrations.
```

Explicit Prompt

```
Each example has the form source -> candidate = label.
Each source is an order record with the exact format
ORD[id=...;region=...;ship=...;coupon=...;items=(SKU,qty,unit_price),...].
Each candidate is a normalized invoice record with the exact format
INV[order_id|region|PRODUCT:qty:line_total;...|disc=...|tax=...|ship=...|total=...].

The label is True iff ALL conditions hold:
(1) both records follow the required structures exactly;
(2) order_id and region are copied exactly;
(3) each SKU is mapped using a fixed mapping, e.g., A1->ALPHA, B2->BETA;
(4) each quantity is copied exactly;
(5) each line_total equals quantity times unit_price;
(6) invoice items stay in the same order as the source items;
(7) discount follows the coupon rule;
(8) tax is computed from the net amount using the region-specific tax rate;
(9) shipping follows the shipping rule;
(10) the final total equals net amount plus tax plus shipping.

Otherwise, the label is False.
```

Example

```
ORD[id=1007;region=US;ship=std;coupon=NONE;
     items=(A1,2,300),(B2,1,500)]
  -> INV[1007|US|ALPHA:2:600;BETA:1:500|disc=0|tax=110|
     ship=500|total=1710] = True

ORD[id=1007;region=US;ship=std;coupon=NONE;
     items=(A1,2,300),(B2,1,500)]
  -> INV[1007|US|ALPHA:2:600;BETA:1:400|disc=0|tax=110|
     ship=500|total=1610] = False
```

Explicit prompts enumerate structural, copying, mapping, arithmetic, and conditional constraints.

Figure 7: Prompt templates for Task 3: order–invoice reconciliation. The explicit-rule prompt specifies the full set of required verification conditions, including structural validity, field preservation, product-code mapping, arithmetic checks, conditional rules, and final total consistency.