StrategyBench: Evaluating Explicit Strategy Induction in Large Language Models

Anonymous ACL submission

Abstract

As large language models are increasingly used in data-scarce and evolving task scenarios, few-shot in-context learning (ICL) has become a key paradigm for task adaptation. However, direct ICL often uses a small set of examples without explicitly abstracting task rules, making it sensitive to example construction. In contrast, human learners often reduce such sensitivity by first summarizing task rules from examples and then applying them to new instances. To evaluate this ability, we propose STRATEGYBENCH, which selects strategy-inducible tasks from BIG-Bench, constructs reference strategies, and defines evaluation metrics along two dimensions: strategy quality and downstream utility. We further analyze strategy induction from three perspectives: task variation, model configuration, and adaptation setting, covering category-wise differences, generator-executor choices, demonstration design, and SFT-based adaptation. Experiments show that explicit strategy utility differs substantially across task categories and depends on both strategy generation and execution conditions. The benchmark is released at: https://anonymous.4open.science/r/StrategyBench-D53C/.

1 Introduction

Large language models have shown strong task adaptation ability under in-context learning (ICL) (Brown et al., 2020; Wei et al., 2022; Wang et al., 2023). In standard few-shot ICL, which we refer to as direct ICL, models solve new tasks by conditioning on a small set of input-output examples. However, this paradigm often exploits examples only at a surface level and struggles to capture stable task rules. The limited context window restricts the number and diversity of examples, preventing models from observing sufficient example variations (Agarwal et al., 2024; Liu et al., 2021; Lu et al., 2022; Min et al., 2022). Moreover, even with more examples, models may rely on surface patterns or shortcuts rather than infer robust rules. Consequently, direct ICL remains sensitive to example construction and fragile under distribution shifts and complex reasoning scenarios (Liu et al., 2024; Mirzadeh et al., 2024).

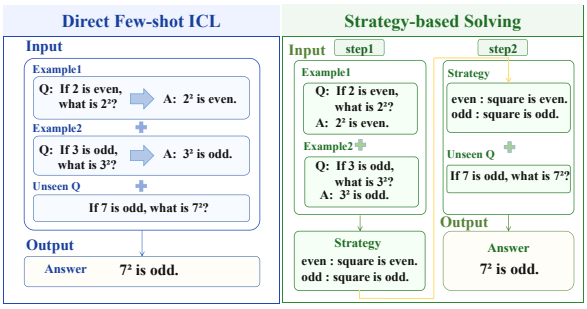

Figure 1: Comparison between direct few-shot ICL and strategy-based solving. In direct few-shot ICL, the model directly predicts the answer for a new query from the provided input-output examples. In strategy-based solving, the model first induces an explicit task-level strategy from the few-shot examples and then applies the induced strategy to solve the unseen query.

To address these limitations, recent studies draw inspiration from the human learning process of "summarizing before applying" and abstract strategy-like representations, such as instructions, skills, or rules, from few-shot examples to guide subsequent problem solving (Honovich et al., 2023; Zheng et al., 2024; Chen et al., 2024). We refer to this paradigm as strategy-based ICL, where models first induce explicit task strategies and then apply them to unseen questions. Unlike Direct ICL, strategy-based ICL makes the underlying task rules explicit and reduces dependence on example construction. Figure 1 illustrates how the two paradigms use example information differently.

Despite the potential of intermediate representations such as explicit strategies, systematic benchmarks for evaluating strategy induction in large language models remain limited. Existing evaluations

mainly measure final-task performance in knowledge understanding, task solving, and mathematical reasoning (Hendrycks et al., 2021; Srivastava et al., 2023; Suzgun et al., 2023; Cobbe et al., 2021), offering only partial insight into intermediate reasoning processes. Recent process-level evaluations have begun to address this gap, but they mostly assess instance-specific solution steps rather than general task-level strategies (Lightman et al., 2024; Zeng et al., 2023; Zheng et al., 2024). However, evaluating strategies is non-trivial, as it requires assessing not only whether a strategy is clear and abstract, but also whether it can be reliably applied by models and improve performance on unseen examples.These requirements place higher demands on benchmark construction and metric design.

To fill this gap, we propose StrategyBench, a benchmark for evaluating whether large language models can induce explicit task-level strategies from few-shot examples. StrategyBench selects strategy-inducible tasks from BIG-Bench and constructs high-quality reference strategies as training data. Based on this dataset, we design a multi-dimensional framework that assesses generated strategies from two perspectives: strategy text quality and downstream application utility. We further analyze how strategy induction and application are affected by model scale, few-shot example number, strategy format constraints, supervised fine-tuning, and strategy executors.

The main contributions of this paper are summarized as follows:

- **Benchmark construction.** We introduce StrategyBench, which is built from BIG-Bench to evaluate explicit strategy induction and supports both evaluation and training with reference strategies.

- **Evaluation framework.** We design a multi-dimensional evaluation framework, which measures both strategy quality and downstream utility.

- **Experimental analysis.** We analyze how model scale, few-shot settings, strategy format constraints, supervised fine-tuning, and strategy executors affect strategy induction and application. Results show that readable strategies do not always yield better execution, and that strategy utility depends on the interaction among the strategy generator, executor, and format constraints.

2 Preliminaries

2.1 Task and Episode Formulation

This section formalizes the task and episode setting and introduces the direct ICL and strategy-based ICL paradigms adopted in this work. Let $\mathcal{T}$ denote the evaluation task set. For each task $T \in \mathcal{T}$, an example is represented as $(q, a) \sim T$, where q is the question and a is the gold answer. Each task defines a class of problems that share the same input–output format and are governed by common underlying rules. For example, in a Chinese-remainder-theorem task, all examples require solving modular constraints.

In each few-shot episode, the model observes a demonstration set

$$\mathcal{D}^T_{\text{few}} = \{(q_i, a_i)\}_{i=1}^{k}, \tag{1}$$

and is evaluated on an unseen example set

$$\mathcal{D}^T_{\text{new}} = \{(q_j^{\text{new}}, a_j^{\text{new}})\}_{j=1}^{N}. \tag{2}$$

Direct ICL. Direct ICL predicts each unseen answer directly from the demonstrations and the query:

$$a_j = h_\theta(\mathcal{D}^T_{\text{few}}, q_j^{\text{new}}), \tag{3}$$

where h_θ is the task-solving model and a_j is the prediction for q_j^{new}.

Strategy-based ICL. Strategy-based ICL separates task solving into strategy induction and strategy application. A generator first induces a task-level strategy from the demonstrations:

$$s_T = f_{\theta_g}(\mathcal{D}^T_{\text{few}}), \tag{4}$$

where s_T is the explicit strategy and f_{θ_g} is the strategy generator. An executor then predicts the answer using the strategy and the unseen query:

$$a_j = g_{\theta_e}(q_j^{\text{new}}, s_T), \tag{5}$$

where g_{θ_e} is the strategy executor.

3 Benchmark Construction

We build StrategyBench from BIG-Bench, which covers diverse task types such as commonsense reasoning, mathematical computation, language understanding, instruction following, and symbolic manipulation. Because BIG-Bench tasks differ substantially in generation methods, evaluation protocols, and input-output formats, we first perform task filtering, dataset splitting, and task categorization, and then construct the reference strategy set. Figure 2 illustrates the overall data construction pipeline.

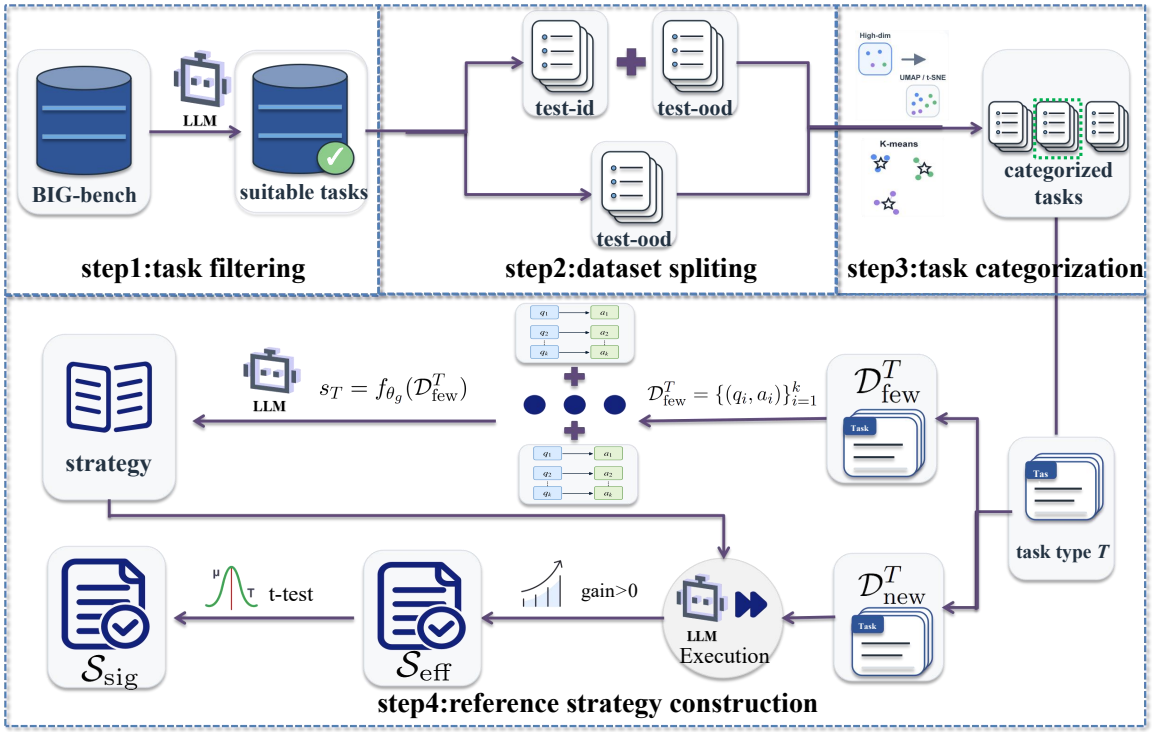

Figure 2: Overall data construction pipeline of StrategyBench. Starting from BIG-Bench, we first filter tasks with the assistance of LLM-based pre-screening and human verification to obtain a suitable task pool. The filtered tasks are then processed along two independent dimensions: dataset splitting, which produces training, ID test, and OOD test subsets, and task categorization, which assigns tasks to coarse-grained categories. Based on the resulting task types and few-shot examples, candidate strategies are generated and further filtered by downstream utility and statistical significance to form the effective strategy set $\mathcal{S}_{\text{eff}}$ and the significant reference strategy set $\mathcal{S}_{\text{sig}}$.

3.1 Task Filtering & Data Normalization

We follow the BBH-style procedure for task filtering and normalization. For filtering, we first remove tasks with dynamic evaluation, strong subjectivity, or overly long inputs. We then select strategy-inducible tasks through a "model-assisted pre-filtering + human verification" procedure along five dimensions: objective scorability, format stability, rule inductibility, low dependence on external knowledge, and strategy application potential. As shown in Table 1, the filtering process substantially reduces the original candidate pool while retaining a diverse set of strategy-inducible tasks.

For normalization, we merge scattered task information into a unified input field and convert original answers into a unified target field. Detailed filtering rules and normalization procedures are provided in Appendix A.

3.2 Dataset Splitting

After filtering, we split the data into training and test sets. The test set is further divided into in-distribution and out-of-distribution subsets to evaluate strategy induction and generalization under different distribution shifts. We also retain BBH as an additional challenging test set. Detailed split definitions and rules are provided in Appendix B.

Stage	#Tasks	#Subtasks	#Examples
Before filtering	108	581	1,614,716
After filtering	44	356	123,845

Table 1: Dataset statistics before and after task filtering.

3.3 Task Categorization

After task filtering, we categorize the retained tasks to characterize their required abilities. We use an "unsupervised clustering + human verification" procedure and group tasks into six categories: **Numerical**, **Logic**, **Language**, **Spatial**, **Procedural**, and **Induction**. Unlike the original BIG-Bench labels, our categorization reorganizes tasks by required abilities, input-output patterns, and underlying solving rules. Details on semantic representation construction, clustering, and human verification are provided in Appendix C.

Strategy Set	#Tasks	#Subtasks	#Strategies
$\mathcal{S}_{\text{eff}}$	22	241	11,993
$\mathcal{S}_{\text{sig}}$	21	221	7,801

Table 2: Statistics of the constructed reference strategy sets. $\mathcal{S}_{\text{eff}}$ contains strategies that improve over Direct ICL, while $\mathcal{S}_{\text{sig}}$ further retains strategies with statistically significant gains.

3.4 Reference Strategy Construction

We construct reference strategy sets to evaluate explicit strategies and support subsequent SFT experiments. The construction process includes two stages: candidate strategy generation and utility-driven filtering.

We first use gemini-3-flash-preview to generate candidate strategies for tasks in each category. We then compare their execution performance on unseen examples against direct ICL and retain strategies that improve performance as the effective strategy set $\mathcal{S}_{\text{eff}}$. From this set, we further select strategies with statistically significant gains to form the significant reference strategy set $\mathcal{S}_{\text{sig}}$. Table 2 reports the statistics of both sets. Detailed generation settings, filtering rules, and additional statistics are provided in Appendix D.

4 Evaluation Metrics

We build a multi-dimensional evaluation framework covering two perspectives: strategy application utility and strategy content quality. The former measures whether a strategy can be effectively applied to improve performance, while the latter evaluates the readability of the generated strategy.

4.1 Strategy Utility

For strategy application utility, we evaluate generated strategies from three aspects: correctness, stability, and answer efficiency. These metrics characterize the practical utility of strategies on unseen examples.

Correctness. Correctness measures whether a strategy helps the model answer unseen examples correctly. We use Corr as the main metric. Let $a_j^{(m)}(s_T)$ denote the prediction in the m-th sampling run under strategy s_T, with $M = 3$ by default. We define:

$$\text{Corr}_j(s_T) = \frac{1}{3}\sum_{m=1}^{3} \mathbf{1}\{a_j^{(m)}(s_T) = a_j^{\text{new}}\}. \quad (6)$$

Corr computes the fraction of predictions that match the gold answer across three sampling runs for the same query. A higher value indicates that the strategy more reliably guides the model to produce correct answers under repeated sampling.

Stability. Stability measures whether the model produces the same results when answering the same question multiple times (regardless of correctness). We evaluate it from two perspectives: sampling consistency Sta_{samp}, which measures consistency across repeated sampling runs, and robustness consistency Sta_{rob}, which measures consistency under perturbations of few-shot example order.

For sampling consistency, let $\{a_j^{(m)}(s_T)\}_{m=1}^{3}$ denote the three sampled answers for the j-th unseen example under strategy s_T, and let v denote a candidate answer. We define:

$$\text{Sta}_{\text{samp}}^{j}(s_T) = \max_{v} \frac{1}{3}\sum_{m=1}^{3} \mathbf{1}\{a_j^{(m)}(s_T) = v\}. \quad (7)$$

This metric computes the fraction of the most frequent prediction across three sampling runs for the same query. A higher value indicates that the strategy more consistently guides the model to produce the same answer under repeated sampling.

For robustness consistency, let $s_T^{(m)} = f_{\theta_g}(\pi_m(\mathcal{D}_{\text{few}}^{T}))$ denote the strategy generated after the m-th permutation of the few-shot examples, and let $a_j^{(m)}$ denote the corresponding prediction. We define:

$$\text{Sta}_{\text{rob}}^{j} = \max_{v} \frac{1}{3}\sum_{m=1}^{3} \mathbf{1}\left(a_j^{(m)} = v\right). \quad (8)$$

This metric computes the fraction of the most frequent prediction across three random example-order perturbations for the same query. A higher value indicates that the strategy more robustly guides the model to produce consistent answers under such perturbations.

Efficiency. Answer efficiency measures whether a method can obtain correct answers with a lower token cost. Let $c_j(s_T) \in [0, 1]$ denote the correctness score of strategy s_T on the j-th query example. Let $l_j^{\text{in}}(s_T)$ and $l_j^{\text{out}}(s_T)$ denote the numbers of input and output tokens, respectively. Given the input length budget $L_{\text{in}}^{\text{budget}}$ and the output length budget $L_{\text{out}}^{\text{budget}}$, we define input and output length efficiency as:

$$\text{LenEff}_{\text{out}}^{j}(s_T) = c_j(s_T) \cdot \min\left(\frac{L_{\text{out}}^{\text{budget}}}{l_j^{\text{out}}(s_T)}, 1\right), \quad (9)$$

$$\text{LenEff}_{\text{in}}^{j}(s_T) = c_j(s_T) \cdot \min\left(\frac{L_{\text{in}}^{\text{budget}}}{l_j^{\text{in}}(s_T)}, 1\right). \quad (10)$$

If the prediction is incorrect, the efficiency score is 0. If the prediction is correct and the token cost does not exceed the budget, the method receives a high score. If the prediction is correct but requires a large token cost, the score is discounted according to the length ratio.

4.2 Strategy Content Quality

Strategy content quality assesses the generated strategy text itself, independent of its effectiveness in specific tasks. Here, we evaluate two metrics: Conciseness and Format Compliance.

Conciseness. Conciseness measures repetition and redundancy within a strategy. Let $s_T = \{u_1, \ldots, u_m\}$ denote a strategy, where u_i is the i-th sentence, and let $\mathrm{sim}(u_i, u_j)$ denote the semantic similarity between two sentences. For $m \geq 2$, we define Conc. as:

$$\mathrm{Conc}(s_T) = 1 - \frac{2}{m(m-1)} \sum_{i<j} \mathrm{sim}(u_i, u_j). \quad (11)$$

When $m < 2$, we set $\mathrm{Conc}(s_T) = 1$. This formula computes the conciseness score by penalizing repeated semantic content within a strategy. A higher value indicates less internal repetition and more concise strategy text.

Format Compliance. Format compliance measures whether the model output follows the predefined strategy format specified in the prompt. An output is assigned a score of 1 if it contains a complete strategy field that satisfies the required parsing format, and 0 otherwise:

$$\mathrm{Fmat}(s_T) = \mathbf{1}\{\text{valid strategy format in } s_T\}. \quad (12)$$

where $\mathbf{1}[\cdot]$ is the indicator function. This metric is used to analyze whether the model can consistently generate strategy outputs that satisfy the format constraints required for automatic parsing. A higher Fmat. indicates that the model produces more format-compliant strategy outputs.

5 Experimental Setup

5.1 Baseline

We compare three few-shot ICL baselines. **Direct ICL** solves unseen questions directly based on few-shot examples (Brown et al., 2020). **CoT-prompted ICL** adds step-by-step reasoning instructions during inference (Kojima et al., 2022). **Least-to-Most Prompting** decomposes a problem into subproblems and solves them sequentially (Zhou et al., 2023). These methods represent three paradigms: example-based prediction, implicit reasoning-enhanced answer generation, and explicit intermediate reasoning.

5.2 Experimental Configuration

We use Qwen3-1.7B, 4B, 8B, and 14B as backbone models. Unless otherwise specified, Qwen3-8B serves as the default strategy generator and Qwen3-4B as the default executor. Prompting-based inference uses $k = 3$ few-shot examples. For stability evaluation, we sample each query 3 times and apply $M = 3$ random perturbations to the few-shot example order. For fine-tuning, we construct SFT data from few-shot examples and reference strategies, and apply LoRA for parameter-efficient training. Detailed decoding parameters, generation and application settings, and training hyperparameters are provided in Appendix E.

6 Results and Analysis

This section analyzes explicit strategy induction on StrategyBench through three research questions:

- **RQ1: How do explicit strategies affect performance across task categories?**
- **RQ2: What factors affect strategy generation and application?**
- **RQ3: How does SFT affect strategy generation capability?**

6.1 Main Result

To answer **RQ1**, we compare our method with different baselines across task categories. We merge the ID and OOD test examples and compute results by task category. We also evaluate all methods on BBH. Table 3 reports the results.

Overall, CoT+Ours achieves the best results on most task categories and metrics, with clear gains on Numerical, Logic, and Language tasks. This suggests that explicit strategies and step-by-step reasoning prompts are complementary. Explicit strategies provide task-level rule constraints, while CoT strengthens instance-level reasoning. The strong results of CoT+Ours on $\mathrm{Sta}_{\mathrm{samp}}$, $\mathrm{Sta}_{\mathrm{rob}}$, $\mathrm{LenEff}_{\mathrm{in}}$, and $\mathrm{LenEff}_{\mathrm{out}}$ further show that this combination improves both output stability and answer efficiency.

Ours alone does not always outperform CoT or L2M, especially on Numerical and Procedural tasks. This suggests that explicit strategies serve better as high-level constraints and cannot fully replace instance-level reasoning or multi-step execution. When combined with CoT, the model can

Metric	Method	Task Category						BBH
		Numerical	Logic	Language	Spatial	Procedural	Induction	
Corr.	L2M	58.43	42.34	22.21	6.21	9.47	27.50	24.09
	Direct ICL	34.31	11.18	4.40	0.69	0.76	0.55	11.55
	Ours	15.90	38.58	22.11	3.59	0.97	17.03	15.34
	CoT	57.30	43.12	19.43	4.71	19.91	28.99	21.40
	CoT+Ours	**69.58**	**55.32**	**36.09**	**8.62**	**25.95**	**40.90**	**37.79**
Sta_{samp}	L2M	72.25	64.95	49.05	36.90	49.23	49.81	48.06
	Direct ICL	73.27	39.80	39.47	34.67	34.95	35.60	44.73
	Ours	60.15	68.53	52.10	36.24	54.84	48.90	51.90
	CoT	73.00	60.98	47.63	36.34	49.76	50.59	45.93
	CoT+Ours	**82.96**	**75.18**	**59.41**	**39.07**	**56.37**	**61.29**	**60.01**
Sta_{rob}	L2M	77.82	72.41	60.83	52.84	60.61	60.03	59.89
	Direct ICL	78.13	54.27	54.02	50.87	51.79	51.18	57.66
	Ours	71.39	75.54	62.28	51.55	**66.91**	62.44	62.31
	CoT	78.67	69.69	59.65	52.23	61.48	61.14	58.54
	CoT+Ours	**86.40**	**80.16**	**67.89**	**53.96**	66.70	**70.48**	**68.91**
$\text{LenEff}_{\text{in}}$	L2M	58.33	41.78	20.77	6.21	9.37	27.45	23.46
	Direct ICL	34.21	11.05	4.20	0.66	0.75	0.52	11.53
	Ours	15.89	38.22	20.89	3.58	0.97	17.03	15.21
	CoT	57.28	42.77	18.53	4.71	19.91	28.97	21.03
	CoT+Ours	**69.49**	**54.78**	**32.80**	**8.62**	**25.95**	**40.87**	**37.13**
$\text{LenEff}_{\text{out}}$	L2M	23.47	18.50	9.09	2.51	3.85	10.70	9.58
	Direct ICL	33.77	11.16	4.38	0.63	0.71	0.54	11.43
	Ours	15.87	38.57	22.10	3.58	0.96	17.02	15.32
	CoT	37.55	33.86	13.40	3.30	13.46	17.93	14.01
	CoT+Ours	**56.04**	**49.00**	**30.19**	**6.76**	**22.69**	**35.66**	**29.49**

Table 3: Utility comparison across different task categories and BBH. Results are computed over all tasks within each category. **Bold** values indicate the best performance among the five methods for each metric within the same task category or BBH setting.

use both task-level rules and step-by-step reasoning, leading to more stable gains across most task categories.

6.2 Factors Affecting Strategy Generation and Application

To answer **RQ2**, we further analyze factors that affect strategy-based ICL, including model scale, the number of few-shot examples, and strategy format constraints.

6.2.1 Effect of Generator and Executor Scale

We use Qwen3-1.7B, 4B, 8B, and 14B as strategy generators, and use Qwen3-1.7B, 4B, and 8B as strategy executors. We evaluate different generator-executor combinations to analyze the effect of model scale. Table 4 reports the results.

Overall, model scale affects strategy quality and utility differently. With a fixed executor, Qwen3-14B produces the best text quality, whereas Qwen3-8B often yields stronger execution performance. With a fixed generator, Qwen3-4B achieves the best execution results under most settings, while scaling the executor to Qwen3-8B does not consistently improve correctness. These results suggest that larger generators improve strategy readability and format quality, but effective strategy application depends on a proper generator–executor match.

6.2.2 Effect of Few-shot Number

To analyze the effect of the number of examples, we compare the strategy quality and utility of strategy-based ICL under 1-, 3-, 5-, 10-, and 30-shot settings. Table 5 reports the results.

This is because as the number of examples increases, the generated strategy becomes more detailed. Although such a strategy may be less concise and could even exceed length limits and become unusable, it helps the model answer questions more effectively. This implies that the model may be better at leveraging detailed rather than concise strategies.

Generator	Strategy Quality		Strategy Utility					
	Conc.	Fmat.	Exec. 1.7B		Exec. 4B		Exec. 8B	
			Corr.	Sta_{samp}	Corr.	Sta_{samp}	Corr.	Sta_{samp}
1.7B	90.98	34.88	14.50	48.30	18.48	54.97	15.86	54.49
4B	90.03	25.12	13.27	42.44	19.41	53.66	14.96	53.98
8B	91.57	22.09	**14.57**	45.03	**20.86**	56.83	**17.74**	54.30
14B	**92.98**	**38.84**	13.21	**48.51**	20.83	**57.51**	16.74	**54.73**

Table 4: Effect of generator and executor model scales. We compare four strategy generators and evaluate their generated strategies with three executors. **Bold** values indicate the best result under the same metric and executor setting.

k	Strategy Quality		Strategy Utility	
	Conc.	Fmat.	Corr.	Sta_{samp}
1	**92.67**	**31.33**	17.34	46.51
3	91.57	22.09	20.89	48.69
5	91.54	21.83	21.26	48.06
10	90.77	15.41	21.19	48.41
30	89.83	14.58	**25.49**	**51.29**

Table 5: Effect of the number of few-shot examples on strategy generation and downstream application under $k = 1, 3, 5, 10, 30$. **Bold** values indicate the best performance across different few-shot settings.

6.2.3 Effect of Prompt Variant

In addition to model scale and the number of examples, prompt design also affects how models interpret the objective of strategy generation. We compare two prompt variants: a free-form prompt and a structured prompt. The free-form prompt uses flexible output constraints to elicit concise and executable strategies, whereas the structured prompt imposes explicit field and format constraints to improve output regularity and parsability. Their full prompt templates are provided in Appendix F, and Table 6 reports the comparison results.

Prompt Type	Strategy Quality		Strategy Utility	
	Conc.	Fmat.	Corr.	Sta_{samp}
Structured Prompt	**91.60**	22.79	**20.86**	**48.55**
Free-form Prompt	90.91	**57.38**	16.96	45.02

Table 6: Comparison between structured and free-form prompts for strategy generation and downstream application. **Bold** values indicate the better result between the two prompt types under the same metric.

The free-form prompt achieves higher Fmat., while the structured prompt performs better on Corr. and Sta_{samp}. This suggests that the free-form prompt is more likely to produce outputs that satisfy the required format, whereas the structured prompt yields strategies with stronger downstream utility and sampling stability. Overall, prompt design affects strategy generation in different ways: format compliance does not necessarily imply better downstream performance, and the utility of a strategy also depends on whether it captures task-relevant rules effectively.

6.3 Effect of SFT on Strategy Generation Capability

To answer **RQ3**, we compare the original Ours with the supervised fine-tuned variant Ours+SFT on ID, OOD, and BBH test sets. Table 7 reports the results.

Ours+SFT consistently improves Conc. and Fmat. over Ours on ID, OOD, and BBH. This indicates that SFT improves the conciseness and format compliance of generated strategies and generalizes across different test distributions. Ours+SFT also improves Sta_{samp} and Sta_{rob} on all three test sets, showing that SFT makes strategy generation more stable and robust. On the other hand, although stability has improved, it remains far from 100%, meaning the model retains a certain level of exploratory capability.

However, Ours+SFT does not improve Corr. and even leads to a slight decrease. This suggests that improving the form and stability of strategy generation does not necessarily translate into higher single-run answer correctness. Strategy utility is also affected by task difficulty and the ability of the executor to apply the generated strategy.

Therefore, although the SFT model does not become "smarter", it achieves greater stability and controllability while maintaining some exploratory ability. This suggests that it could serve as a strong initialization for reinforcement learning (Ouyang et al., 2022), which remains a direction for future work.

7 Related Work

Few-shot In-context Learning. Few-shot in-context learning (ICL) enables large language models to adapt to new tasks using only a small number of examples without parameter updates (Brown

Method	ID					OOD					BBH				
	Quality		Utility			Quality		Utility			Quality		Utility		
	Conc.	Fmat.	Corr.	Sta_{samp}	Sta_{rob}	Conc.	Fmat.	Corr.	Sta_{samp}	Sta_{rob}	Conc.	Fmat.	Corr.	Sta_{samp}	Sta_{rob}
Ours	92.13	6.68	**22.71**	59.61	68.08	91.46	21.94	**25.42**	59.67	68.18	91.77	5.15	**15.34**	51.90	62.32
Ours+SFT	**94.17**	**62.06**	21.33	**70.08**	**77.79**	**93.03**	**61.95**	23.29	**62.89**	**72.58**	**93.44**	**50.35**	14.20	**66.51**	**76.41**

Table 7: Comparison between Ours and Ours+SFT on ID, OOD, and BBH test sets. Each split reports two strategy quality metrics and three strategy utility metrics. **Bold** values indicate the best performance between the two variants under the same evaluation setting.

et al., 2020). Prior studies show that ICL is highly sensitive to example selection, semantic relevance, and example order (Liu et al., 2021; Rubin et al., 2022; Lu et al., 2022). Increasing the number of examples does not fully eliminate the effects of context length and example organization (Agarwal et al., 2024). To improve the task-solving ability of few-shot ICL, researchers have proposed several enhanced prompting paradigms, including chain-of-thought prompting (Wei et al., 2022), self-consistency (Wang et al., 2023), and least-to-most prompting (Zhou et al., 2023). Recent studies further explore explicit intermediate structures, such as Self-Discover, PAL, and PoT (Zhou et al., 2024; Gao et al., 2023; Chen et al., 2023). These methods improve performance in specific reasoning scenarios, but they usually serve individual queries or specific reasoning forms. In contrast, this paper focuses on whether models can induce transferable task-level strategies from few-shot examples.

Benchmarks for In-context Learning. Existing benchmarks mainly evaluate final answer accuracy or task scores, such as MMLU, BIG-Bench, BBH, and GSM8K (Hendrycks et al., 2021; Srivastava et al., 2023; Suzgun et al., 2023; Cobbe et al., 2021). They provide limited evaluation of the reasoning process. Some recent studies begin to evaluate or supervise process-level reasoning, such as step verification for mathematical reasoning and reasoning trace evaluation (Lightman et al., 2024; Lee and Hockenmaier, 2025). However, these evaluations usually focus on the solution process of individual examples, rather than whether models can induce general strategies that transfer to similar examples. This paper takes explicit strategy induction as the core evaluation target and further evaluates the executability of induced strategies through downstream task performance. Table 8 compares representative benchmarks and evaluations.

Benchmark / Eval.	Ans. Eval.	Proc. Eval.	Strategy Ind.	Strategy Utility
MMLU (Hendrycks et al., 2021)	✓	×	×	×
BBH (Suzgun et al., 2023)	✓	×	×	×
GSM8K (Cobbe et al., 2021)	✓	×	×	×
Step Verif. (Lightman et al., 2024)	✓	✓	×	×
Trace Eval. (Lee and Hockenmaier, 2025)	✓	✓	×	×
Ours	✓	✓	✓	✓

Table 8: Comparison of representative benchmarks and process-level evaluations. Ans. Eval., Proc. Eval., Strategy Ind., and Strategy Utility denote answer-level evaluation, process-level evaluation, strategy induction, and strategy utility evaluation, respectively.

Concurrent work. A concurrent anonymous study uses StrategyBench and an SFT model trained from our reference strategies to develop CoDI, a downstream method for OOD generalization (Anonymous, 2026). This work is complementary to ours, which focuses on benchmark construction and evaluation; details are provided in Appendix G.

8 Conclusion

This paper introduces StrategyBench, a benchmark for evaluating explicit strategy induction in large language models, together with metrics covering both strategy text quality and downstream application utility. Experiments show that explicit strategies can improve task-solving performance, but their utility depends on task type, example configuration, and the generator–executor match. Supervised fine-tuning further improves the stability and generalization of strategy generation.

Future work will extend StrategyBench to more task sources, including real-world, interactive, and tool-use scenarios. It is also important to distinguish strategy generation failures from execution failures more precisely and to explore training methods that enhance the transferability and executability of generated strategies.

9 Limitations

This paper has several limitations in methodology and experiments.

Methodology. (1) StrategyBench is mainly built from BIG-Bench and BBH. Although the filtered tasks cover multiple reasoning categories, the data sources are still relatively limited. Future work can incorporate more independent benchmarks to evaluate the generalization of explicit strategy induction across broader task distributions. (2) This paper evaluates generated strategies from two perspectives: strategy text quality and downstream application utility. However, it does not fully distinguish different sources of failure. For example, an incorrect answer may result from an inaccurate induced strategy, or from the executor's failure to understand and apply the strategy correctly. Future work should conduct a more fine-grained analysis of strategy generation errors and strategy execution errors.

Experiments. (1) Due to computational resource constraints, this paper mainly evaluates models from the Qwen3 family. This setting enables controlled scale comparisons within the same model family, but it does not cover more model families or closed-source models. (2) The fine-tuning experiments in this paper mainly focus on the effect of SFT on strategy generation capability. We have not further explored reinforcement learning, preference optimization, or multi-stage curriculum learning. These methods may further improve the executability and downstream utility of generated strategies.

References

Rishabh Agarwal, Avi Singh, Lei M. Zhang, Bernd Bohnet, Lorena Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, John D. Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra Faust, and Hugo Larochelle. 2024. Many-shot in-context learning. *arXiv preprint arXiv:2404.11018*.

Anonymous. 2026. Cognitively decoupled meta-induction for out-of-distribution generalization. Under review.

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*.

Ping-Chen Chen, Szu-Lin Wei, Hen-Hsen Huang, and Hsin-Hsi Chen. 2024. Induct-learn: Short phrase prompting with instruction induction. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 5204–5231.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gheorghe Comanici, David Bieber, Mike Schaekermann, Panupong Pasupat, Noveen Sachdeva, Inderjit Dhillon, Michael Blistein, Omer Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*.

Gemini Team. 2023. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemini Team. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. 2023. Instruction induction: From few examples to natural language task descriptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 1935–1952.

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.

Jinu Lee and Julia Hockenmaier. 2025. Evaluating step-by-step reasoning traces: A survey. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 1789–1814.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

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*.

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. Association for Computational Linguistics.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

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*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Alex R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL*.

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

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*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhiyuan Zeng, Peng Chen, Sheng Liu, Haoming Jiang, and Jia Jia. 2023. Mr-gsm8k: A meta-reasoning benchmark for large language model evaluation. *arXiv preprint arXiv:2312.17080*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2024. Take a step back: Evoking reasoning via abstraction in large language models. In *International Conference on Learning Representations*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024. Self-discover: Large language models self-compose reasoning structures. In *Advances in Neural Information Processing Systems*.

A Task Selection and Standardization Details

This appendix provides additional details on the task selection and format standardization process of STRATEGYBENCH. Overall, we first follow the BBH-style procedure to preprocess and filter the original BIG-Bench tasks, removing tasks that are not suitable for unified few-shot strategy induction

evaluation. We then adopt a "model-assisted prefiltering + human verification" procedure to further select tasks suitable for strategy induction. Finally, we standardize the input and target formats of the retained examples.

A.1 Task Filtering

We first perform an initial filtering over the original BIG-Bench tasks and remove three types of tasks that are clearly unsuitable for static few-shot strategy induction evaluation. The first type includes tasks that rely on dynamic generation or external program execution for evaluation, as these tasks do not provide fixed input-output examples. The second type includes highly subjective or sensitive tasks, such as bias, toxicity, and ethical judgment tasks, which usually lack a single stable and objective gold answer. The third type includes examples whose input length exceeds 2500 characters. We remove these examples to reduce the interference of long contexts with few-shot example construction, strategy generation, and strategy application, and to control inference cost.

A.2 Model-assisted Task Selection

After the initial filtering, we further determine whether each candidate subtask is suitable for explicit strategy induction evaluation. Specifically, we use Qwen3-32B to automatically score each candidate subtask, and then conduct human verification to finalize the selection. The scoring dimensions include objective scorability, format stability, rule inductibility, external knowledge risk, and strategy transfer potential. Each dimension is scored from 0 to 2. For all dimensions except external knowledge risk, a higher score indicates that the task is more suitable for strategy induction. External knowledge risk is a negative indicator, where a higher score means that the task relies more heavily on external knowledge.

Objective scorability. This dimension measures whether a task has clear and stable objective evaluation criteria. If a task can be evaluated by exact match or clear multiple-choice answers, it receives a score of 2. If its answers are partially objective but contain some ambiguity, it receives a score of 1. If the task is open-ended or subjective and has unclear answer boundaries, it receives a score of 0.

Format stability. This dimension measures whether the input-output format is stable within the same task or subtask. If the task format is fixed and the input-output structure is clear, it receives a score of 2. If the overall format is relatively stable but contains some noise or variants, it receives a score of 1. If the task format varies substantially and makes it difficult to extract stable patterns, it receives a score of 0.

Rule inductibility. This dimension measures whether a task contains rules, steps, or algorithms that can be abstracted from a small number of examples. If a task has clear inductive rules and can be written as stable operation steps, it receives a score of 2. If it contains certain patterns but the rules are incomplete or weakly generalizable, it receives a score of 1. If the task mainly relies on commonsense memory, linguistic intuition, or cultural background and is difficult to abstract into a general strategy, it receives a score of 0.

External knowledge risk. This dimension measures whether solving a task depends on the model's prior world knowledge, cultural knowledge, or domain-specific knowledge, rather than information that can be induced from few-shot examples. This dimension is a negative indicator. If a task heavily depends on external knowledge, it receives a score of 2. If it requires some external knowledge but external knowledge is not the dominant factor, it receives a score of 1. If the task can mostly be solved through examples and induced rules, it receives a score of 0.

Strategy utility potential. This dimension measures whether an explicit strategy can potentially bring practical gains compared with the no-strategy setting. For example, BIG-Bench tasks such as moral permissibility and simple ethical questions usually depend on value judgments, contextual understanding, or safety preferences, and are difficult to formulate as stable objective rules. Explicit strategies may oversimplify complex contexts into partial judgment rules and reduce the model's adaptability to specific examples. Therefore, these tasks have low strategy utility potential. If errors in a task can be substantially reduced by a clear strategy, the task receives a score of 2. If a strategy may provide some help but the improvement is limited or unstable across examples, it receives a score of 1. If a strategy is unlikely to improve performance, or the task mainly relies on memorization, commonsense knowledge, or surface matching, it receives a score of 0.

A.3 Selection Rule

Based on the five dimensions above, we define the suitability score of a candidate subtask as:

$$\text{Score} = s_{\text{obj}} + s_{\text{format}} + s_{\text{rule}} + s_{\text{transfer}} - s_{\text{know}}, \tag{13}$$

where s_{obj}, s_{format}, s_{rule}, and s_{transfer} denote the scores of objective scorability, format stability, rule inductibility, and strategy transfer potential, respectively. s_{know} denotes the score of external knowledge risk.

Using only the total score may retain some tasks that are not suitable for strategy induction evaluation. For example, some tasks are format-stable and easy to score, but heavily rely on external knowledge or cultural background. Their performance changes may mainly come from the model's memorized knowledge rather than strategies induced from few-shot examples. To avoid this issue, we use a filtering rule based on both the total score and necessary constraints. A candidate subtask must satisfy:

$$\text{Score} \geq 7, \tag{14}$$

and:

$$s_{\text{know}} \leq 1, \quad s_{\text{obj}} = 2, \quad s_{\text{transfer}} = 2. \tag{15}$$

Here, $s_{\text{know}} \leq 1$ controls the risk of external knowledge dependence, $s_{\text{obj}} = 2$ ensures stable objective evaluation criteria, and $s_{\text{transfer}} = 2$ ensures clear strategy transfer potential. After model-assisted scoring, we further conduct human verification and remove tasks that clearly deviate from our research goal, such as multimodal tasks, highly sensitive tasks, and tasks that are clearly inconsistent with rule induction and strategy transfer.

A.4 Input and Target Standardization

After task filtering, we standardize the retained examples so that different BIG-Bench tasks can be uniformly used for few-shot strategy induction and automatic evaluation. The standardization process mainly includes input field standardization and target answer standardization.

Input standardization. The input information in the original BIG-Bench examples may be scattered across multiple fields, including task prefix, example input prefix, input prefix, and example output prefix. We integrate these fields into a unified input field. Specifically, task prefix usually represents task instructions, input represents the concrete question text, choice prefix provides the option prefix for multiple-choice tasks, and example output prefix specifies the expected output format.

Field	Example Value
`task`	`disambiguation_qa`
`subtask`	`disambiguation_qa`
`input`	In the following sentences, explain the antecedent of the pronoun. Options: (A) ... (B) ... (C) Ambiguous.
`target`	`["A"]`

Table 9: Example of a standardized instance.

For multiple-choice examples, if the original options do not have explicit labels, we assign unified option IDs such as A/B/C/D. This ensures that model outputs can be aligned with automatic evaluation. This processing enables different tasks to be organized into a unified few-shot prompt format and supports subsequent exact-match evaluation.

Target standardization. We convert the original answers into a unified target field. For binary classification tasks, such as yes/no questions, we directly retain the answer word. For general multiple-choice tasks, we convert the answer into the corresponding option ID, such as A, B, C, or D. For tasks with too many options or unstable option labels, we directly retain the original answer string. For exact-match tasks, we retain the gold answer in string form.

Standardized instance format. After standardization, each example is converted into a unified format. A simplified example is shown below:

Here, task denotes the original BIG-Bench task name, subtask denotes the corresponding subtask, input denotes the normalized model input, and target denotes the gold answer list used for automatic evaluation.

B Dataset Splitting and Sampling Details

This section provides additional details on the dataset splitting and sampling rules of STRATEGYBENCH. Since the original BIG-Bench tasks vary substantially in sample size, subtask scale, input-output format, and evaluation difficulty, directly using all examples may cause large-scale tasks to dominate training or evaluation and reduce comparability across tasks. Therefore, we follow two main principles when constructing the benchmark. First, we avoid obvious information leakage

between the training and test sets. Second, we aim to keep the number of examples relatively balanced across tasks and subtasks.

Training and test split. We first split the preprocessed BIG-Bench data into training and test sets. The training set is mainly used to construct few-shot examples, generate reference strategy data, and support model development and debugging. The test set is used to evaluate the strategy induction and strategy application abilities of models under different distribution conditions. To systematically analyze model generalization, we further divide the test set into in-distribution and out-of-distribution subsets.

In-distribution test set. The in-distribution test set is denoted as ID. Test examples in this set come from the same subtask set as the training examples, but the concrete examples do not overlap. This setting evaluates whether models can induce stable and effective strategies from few-shot examples and apply them to new test examples when the task rules and input-output formats are relatively consistent.

Out-of-distribution test set. The out-of-distribution test set is used to evaluate strategy induction and transfer ability beyond the training distribution. We assign examples to the OOD test set when they differ clearly from the training set in task source, input-output format, or required ability. We also avoid direct overlap with task rules and example templates used during training. This setting examines whether models can induce transferable explicit strategies from few-shot examples when facing unseen task structures or distribution shifts.

BBH test set. In addition to the internal BIG-Bench split, we retain BIG-Bench Hard (BBH) as an additional challenging test set. BBH contains a set of tasks that are more challenging for large language models and can be used to examine strategy induction ability in complex reasoning, symbolic manipulation, and multi-step problem-solving scenarios. We select tasks from BBH that are suitable for automatic evaluation and strategy induction, and ensure that they do not directly overlap with tasks in the training set. The final retained BBH set serves as an additional challenging out-of-distribution evaluation set.

Balanced sampling strategy. Since different tasks and subtasks vary greatly in sample size, we use a k-cut based balanced sampling method when constructing the training and evaluation sets. Given a set of data collections $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$, where $|\mathcal{D}_i|$ denotes the number of examples in the i-th collection, the sampling goal is to retain at most k examples from each collection. If a collection contains fewer than k examples, all examples in that collection are retained. To ensure that the overall sample size is not smaller than a predefined target size N, we solve for the smallest integer k that satisfies:

$$f(k) = \sum_{i=1}^{m} \min(k, |\mathcal{D}_i|) \geq N. \qquad (16)$$

Here, $f(k)$ denotes the total number of examples obtained when at most k examples are sampled from each collection. Since $f(k)$ is monotonically increasing with respect to k, the smallest feasible k can be efficiently found by binary search. After obtaining k, we apply the following sampling rule to each collection: if $|\mathcal{D}_i| \leq k$, all examples are retained; if $|\mathcal{D}_i| > k$, we randomly sample k examples from the collection. This method controls the overall sample size while preventing a small number of large-scale tasks from contributing too many examples, making the influence of different tasks more balanced during training and evaluation.

Boundary cases. We handle boundary cases as follows. If the total number of examples across all candidate collections is still smaller than the target size, i.e.,

$$\sum_{i=1}^{m} |\mathcal{D}_i| \leq N, \qquad (17)$$

we do not perform truncation and directly retain all examples. If the number of candidate collections is already no smaller than the target size, i.e., $m \geq N$, we set $k = 1$ to ensure that each collection contributes at least one example and preserve task coverage as much as possible. In other cases, we solve for the smallest k that satisfies $f(k) \geq N$ and then perform sampling as described above.

C Task Categorization

To support category-aware strategy generation and task-type-specific result analysis, we further divide the retained tasks into six coarse-grained semantic categories. This categorization does not directly follow the original BIG-Bench task labels. Instead,

it considers the core abilities required by each task, the input-output format, and the underlying solving process. Compared with the original task names, our categorization focuses more on the types of strategies that models need to induce and execute when solving the tasks. Therefore, it is more suitable for analyzing how explicit strategies affect different task types.

Category definitions. We define the six coarse-grained task categories as follows:

- **Numerical.** This category mainly involves numerical computation, arithmetic operations, counting, modular arithmetic, quantity comparison, or other rules related to numerical manipulation. Models usually need to perform deterministic mathematical or symbolic computations based on the numerical information in the question.

- **Logic.** This category mainly evaluates logical reasoning, rule constraints, symbolic relations, and consistency judgments. Models need to identify the conditions or constraints in the question and follow specific reasoning steps to obtain the answer.

- **Language.** This category focuses on natural language understanding, including lexical relations, syntactic patterns, textual entailment, paraphrase recognition, and judgments based on linguistic cues. The key ability is to extract semantic relations from text and make predictions accordingly.

- **Spatial.** This category involves spatial positions, directional relations, object movement, state changes, or transformations in structured spaces. Models usually need to maintain intermediate spatial states and reason based on changes in positions or directions.

- **Procedural.** This category requires models to follow given steps, operation sequences, or procedural rules. The main challenge is to correctly track intermediate states during multi-step operations and avoid state update errors in long-step reasoning.

- **Induction.** This category requires models to induce hidden rules, mapping relations, or abstract patterns from examples and transfer the induced rules to new query examples. Since these tasks explicitly rely on the process of summarizing rules from examples and then applying them, they are highly related to the strategy induction ability studied in this paper.

Categorization procedure. During categorization, we first construct a textual representation for each task. This representation consists of standardized input examples, output formats, and task-related metadata. We then use a semantic embedding model to encode the task representations and perform unsupervised clustering based on the embedding vectors to obtain initial task groups. Since automatic clustering may be affected by surface textual forms and place tasks with different solving mechanisms into the same group, we further conduct human semantic verification. Based on the main ability requirements and solving structures of the tasks, we merge the initial clusters into the six coarse-grained categories defined above. This process ensures that tasks within the same category have similar strategy requirements, while different categories reflect clear differences in reasoning abilities.

D Reference Strategy Construction Details

This section provides additional details on the construction of the reference strategy sets. We do not manually annotate solution processes for each test example. Instead, we automatically generate candidate strategies that summarize task-solving rules from few-shot examples, and then filter effective reference strategies based on downstream execution performance. The overall process consists of three steps: category-aware prompt design, candidate strategy generation, and utility-driven strategy filtering.

Category-aware prompt design. Different task types rely on different solving rules. For example, numerical tasks usually require step-by-step computation and boundary condition checking. Logic tasks rely more on constraint identification and stepwise reasoning. Procedural tasks require maintaining intermediate states, while induction tasks require extracting reusable hidden rules from examples. Therefore, we do not use a single generic prompt to generate all candidate strategies. Instead, we design category-aware prompts for strategy generation according to task categories.

Specifically, we design strategy generation prompts for the six coarse-grained task categories.

Each category has two generation templates. One template emphasizes step-by-step reasoning and executable operation procedures, while the other emphasizes concise summaries of core rules. The former is suitable for tasks that require explicit state tracking, constraint checking, or multi-step reasoning. The latter is suitable for tasks with short rules, stable patterns, or relatively direct solving processes. In total, we obtain 12 category-aware strategy generation prompts to cover the strategy expression needs of different task types.

Candidate strategy generation. Given the few-shot example set $\mathcal{D}_{\text{demo}}$ for a task or subtask, we use a strategy generator f_{θ_g} to generate candidate strategies:

$$s = f_{\theta_g}(\mathcal{D}_{\text{demo}}, p_c), \tag{18}$$

where p_c denotes the category-aware prompt corresponding to the current task category c, and s denotes the generated candidate strategy. A candidate strategy may take the form of natural language rules, operation steps, input-output mapping relations, constraint checking methods, or error avoidance instructions. Its goal is not to restate the few-shot examples, but to extract general solving rules that can transfer to new query examples.

For each few-shot setting used to construct strategies, we select the corresponding prompt according to the task category and feed the few-shot examples into the strategy generator to obtain candidate strategies. The generated candidate strategies are not directly used as final reference strategies. Instead, they are passed to the downstream utility filtering stage.

Utility-based strategy filtering. Automatically generated candidate strategies may contain redundancy, errors, or information that is not helpful for downstream answering. Therefore, we further filter candidate strategies based on their execution performance on unseen query examples. Specifically, we use Direct ICL as the baseline and add the candidate strategy into the context to form strategy-based ICL. We then compare the accuracy of the two methods on the corresponding unseen query set $\mathcal{D}_{\text{query}}$.

Let $\text{Acc}_{\text{direct}}$ denote the accuracy of Direct ICL, and let $\text{Acc}_{\text{strategy}}(s)$ denote the accuracy after using the candidate strategy s. If the candidate strategy satisfies:

$$\text{Acc}_{\text{strategy}}(s) > \text{Acc}_{\text{direct}}, \tag{19}$$

we consider the strategy to bring downstream performance gains and add it to the effective strategy set $\mathcal{S}_{\text{eff}}$.

Based on the effective strategy set, we further select strategies with more stable gains and obtain the significant reference strategy set $\mathcal{S}_{\text{sig}}$:

$$\mathcal{S}_{\text{sig}} \subseteq \mathcal{S}_{\text{eff}}. \tag{20}$$

Here, $\mathcal{S}_{\text{eff}}$ retains all candidate strategies that improve performance over Direct ICL, while $\mathcal{S}_{\text{sig}}$ further emphasizes the stability and reliability of strategy gains. We use $\mathcal{S}_{\text{sig}}$ as the main high-quality reference strategy set and use $\mathcal{S}_{\text{eff}}$ for supplementary statistics and analysis.

Resulting strategy sets. Finally, we construct two levels of reference strategy data: the effective strategy set $\mathcal{S}_{\text{eff}}$ and the significant reference strategy set $\mathcal{S}_{\text{sig}}$. The former characterizes the range of automatically generated strategies that bring practical gains, while the latter provides a higher-quality and more stable source of reference strategies. The overall statistics of the two strategy sets are shown in Table 2 in the main paper.

E Implementation Details

This section provides additional details on strategy generation, strategy application, and supervised fine-tuning settings. The main paper only reports the major experimental settings, while the detailed implementation parameters are summarized in this section.

Strategy generation settings. We use a local OpenAI-compatible inference interface for strategy generation. Unless otherwise specified, strategy generation uses prompt v1 as the prompt template, the default number of few-shot examples is $k = 3$, and each bucket generates one candidate strategy by default. The generation temperature is set to 0.2, and the maximum generation length is set to 700 tokens. To improve request stability, we allow at most 6 retries and set a 0.2-second interval between adjacent requests. For stability and example perturbation experiments, we construct 5 strategy variants for each bucket, including 3 generations with the original example order and 2 generations with randomly permuted example orders. The random seed is set to 42.

Strategy application settings. During strategy application, the executor model receives the induced strategy and a new query question, and is

Hyperparameter	Value
Base model	Qwen3-8B
Max length	4096
Max steps	2200
Batch size	1
Gradient accumulation	8
Learning rate	1×10^{-4}
Warmup ratio	0.03
Eval / Save steps	100 / 100
Logging steps	10
LoRA rank	16
LoRA alpha	32
Target modules	Attention layers
Precision	bf16

Table 10: Main hyperparameter settings for supervised fine-tuning.

instructed to output only the final answer. Unless otherwise specified, the decoding temperature in the strategy application stage is set to 0.7, the maximum output length is set to 256 tokens, and at most 3 retries are allowed. The inference process supports concurrent requests, with the default concurrency set to 8. For test example loading, the script prioritizes the test20 json file under the corresponding task directory. If full testing is enabled, all test examples in the file are evaluated. For answer judgment, we first extract the final answer from the model output, apply lightweight normalization, and then compare it with the gold answer using exact match.

Supervised fine-tuning settings. For SFT experiments, we construct supervised fine-tuning data from few-shot examples and their reference strategies, and use LoRA for parameter-efficient fine-tuning. The training model is Qwen3-8B. The maximum sequence length is set to 4096, and the maximum number of training steps is set to 2200. The per-device batch size is 1, and the gradient accumulation step is 8. The learning rate is set to 1×10^{-4}, and the warmup ratio is set to 0.03. The model is evaluated and saved every 100 steps, and training logs are recorded every 10 steps. The LoRA rank is set to 16, LoRA alpha is set to 32, the target modules are attention layers, and training uses bf16 precision.

SFT data format. Each SFT instance follows the previous JSONL message style. The input contains the strategy-generation instruction and the few-shot QA block, while the output is the corresponding reference strategy used as the supervised target. Figure 3 shows an example of the SFT data structure.

E.1 Model and License Details

We use open-source Qwen3 models (Yang et al., 2025) with different parameter scales, including Qwen3-1.7B, Qwen3-4B, Qwen3-8B, and Qwen3-14B. These open-weight models are released under the Apache 2.0 license. We deploy them locally through an OpenAI-compatible API interface to ensure a unified inference pipeline across strategy generation, strategy application, and baseline evaluation.

We additionally use Gemini-3-Flash-Preview as an API-based model for candidate strategy generation during reference strategy construction. Since no separate technical report is available for this preview model, we cite the Gemini model family reports (Gemini Team, 2023, 2024; Comanici et al., 2025). The API model is accessed through its official service interface and is used only to produce candidate strategies, which are further filtered by downstream utility before being included in the reference strategy sets.

For supervised fine-tuning, we fine-tune a selected Qwen backbone on the constructed strategy data and evaluate the resulting model under the same prompts, data splits, and metrics as the base models.

Use of AI Writing Assistance. We used GPT-based writing assistance solely to polish the language of author-written text, including grammar correction, wording refinement, and clarity improvement. The tool was not used to generate research ideas, experimental results, analyses, citations, or conclusions. All content was reviewed, revised, and verified by the authors, who take full responsibility for the final manuscript.

F Prompt Templates

This section provides the prompt templates used for strategy generation. We consider two types of prompts: a structured prompt and a free-form prompt. The structured prompt specifies explicit output fields and format constraints, aiming to make the generated strategies more regular and easier to parse. The free-form prompt imposes fewer structural constraints and instead encourages the model to summarize concise and executable task-level strategies from the demonstrations. In this paper, Prompt V1 refers to the structured prompt,

and Prompt V2 refers to the free-form prompt. Unless otherwise specified, Prompt V1 is used as the default strategy generation prompt in the main experiments.

G Concurrent Work

A concurrent anonymous study proposes CoDI for cognitively decoupled meta-induction and OOD generalization (Anonymous, 2026). It uses STRATEGYBENCH as a training and evaluation testbed and further leverages an SFT model trained from our reference strategies. The concurrent study develops a downstream method based on these resources, whereas this paper focuses on constructing the benchmark, reference strategy resources, and evaluation framework for explicit strategy induction.

Figure 6 illustrates the overall framework of CoDI. The method separates strategy induction from answer execution: a strategy generator first induces task-level representations from support examples, and a frozen answer solver then applies the induced representation to answer new queries. This design is complementary to our benchmark-oriented contribution.

SFT Data Structure Example (JSONL) `messages + meta`

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are a Strategy Induction Assistant.\n\nTask:\nGiven a ...",
    },
    {
      "role": "user",
      "content": "Inputs:\nQA Samples:\n<sample>\n<question>\nThere are 3 ho...",
    },
    {
      "role": "assistant",
      "content": "<strategy>\n1. Extract all fixed positions and relative co..."
    }
  ],
  "meta": {
    "task": "logic_grid_puzzle",
    "subtask": "logic_grid_puzzle",
    "qid": "007",
    "strategy_path": "positive_strategies/train_all/v1/gain_pos_sig/coarse_4...",
    "qa_json_path": "data/data_train_all/kmeans_6/bucketed_outputs_k6_split..."
  }
}
```

Figure: JSONL-formatted SFT training sample with system, user, and assistant messages.

Figure 3: Example of the JSONL-style data structure used for strategy supervised fine-tuning. Each instance contains the strategy-generation instruction, the few-shot QA block, and the reference strategy as the supervised target.

Prompt for Strategy Generation (V1)

You are a Strategy Induction Assistant.

Task:

Given QA samples from the same task family, infer a concise, externally executable strategy that a small model can follow.

Hard constraints (VERY IMPORTANT):

- Derive the strategy ONLY from patterns explicitly supported by the QA samples.
- Do NOT assume a specific task type beyond what samples demonstrate.
- Do NOT introduce new entities unless clearly present in samples.
- Do NOT rely on answer choices to guide intermediate reasoning.
- Prefer deterministic checks and rule application over vague advice.
- Focus on operations that remain valid under surface variation.

Output requirements:

- Output ONLY a single <strategy>...</strategy> block.
- 3 to 5 numbered steps.
- Each step MUST start with an action verb:
  Extract / Identify / Map / Transform / Apply / Compute / Check / Verify / Select / Output
- Each step MUST be <= 12 words.
- Steps MUST be concrete operations, not explanations.

Inputs:

QA Samples:

`{QA_BLOCK}`

Figure 4: Prompt V1 used for strategy generation.

Prompt for Strategy Generation (V2)

You are a Strategy Induction Assistant.

Task:

Given a set of QA samples from the same task, infer a general strategy that can help solve similar questions.

The inferred task type is for internal use only and must NOT be output. The strategy should increase the likelihood of solving unseen questions from the same task.

Core Requirements:

- Do NOT output hidden chain-of-thought reasoning.
- Do NOT describe the task itself.
- Do NOT reference the specific QA samples.
- Focus only on reusable reasoning actions.

Expression Guidelines:

- Write the strategy as 2–4 concise sentences.
- Do NOT use numbered steps.
- Do NOT use bullet points or lists.
- Each sentence should describe a useful reasoning action or decision rule.
- Prefer practical actions over generic advice.

Output Structure:

The output MUST contain ONLY the <strategy> block.

```
<strategy>
Describe the general strategy using a few concise sentences.
</strategy>
```

Inputs:

QA Samples:

```
{QA_BLOCK}
```

Figure 5: Prompt V2 used for strategy generation.

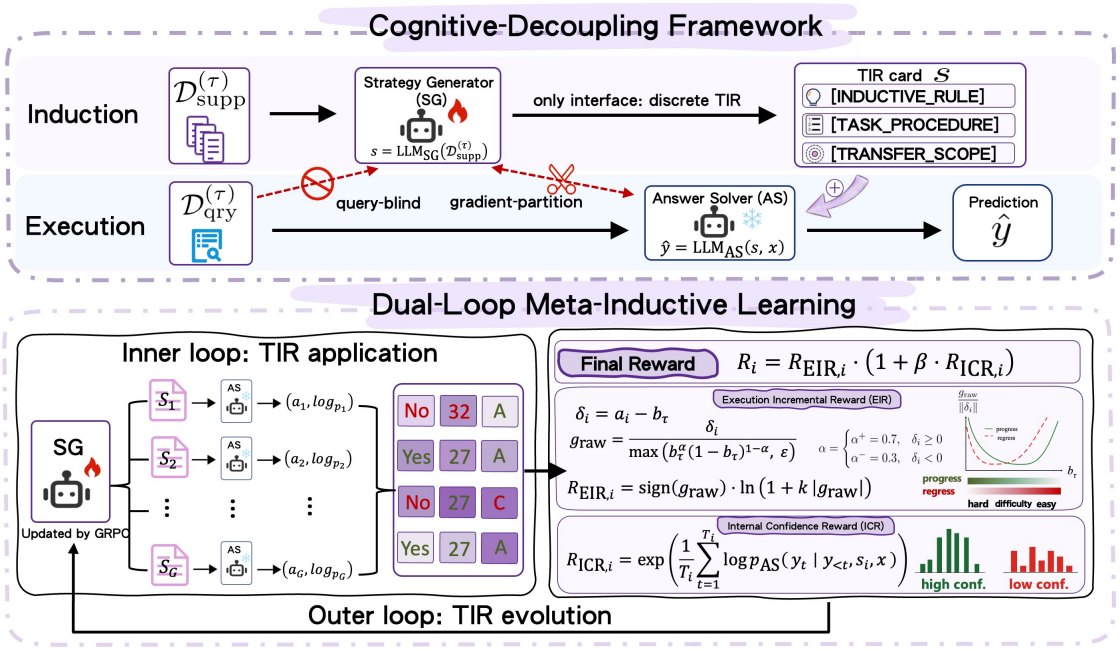

Figure 6: Overview of the concurrent CoDI framework. CoDI uses StrategyBench as a training and evaluation testbed and leverages an SFT model trained from reference strategies. The framework separates task-level strategy induction from answer execution, serving as a downstream method built on the benchmark resources introduced in this paper.