

Metaphor-Induced Algorithmic Steering: Cross-Domain Procedural Transfer in LLM Code Generation

Zhibo Hu

The University of New South Wales
CSIRO
Australia
zhibo.hu@unsw.edu.au

Chen Wang

CSIRO
The University of New South Wales
Australia
Chen.Wang@csiro.au

Yanfeng Shu

CSIRO
Australia
Yanfeng.Shu@csiro.au

Hye-young Paik

The University of New South Wales
Australia
h.paik@unsw.edu.au

Liming Dong

CSIRO
Australia
Liming.Dong@csiro.au

Liming Zhu

CSIRO
The University of New South Wales
Australia
Liming.Zhu@csiro.au

Abstract

Large language models benefit from elements in natural language, such as metaphors and analogies in training data and inference input to achieve generalisability across different domains. However, these language elements may also lead to unwanted behaviors when metaphorical expressions implicitly transfer inappropriate procedural patterns into new tasks.

In this paper, we show that metaphorical instructions can induce analogical transfer of procedural mechanisms, thus steering code-generation models towards less efficient algorithms. We refer to this metaphor-induced effect as *metaphorical algorithmic steering*: a skill that is semantically benign, non-directive, and plausible within its source domain transfers an abstract procedural schema into a programming task, causing the model to favor exhaustive search, full scans, or repeated reconstruction over more efficient strategies without explicitly mentioning the target algorithm or efficiency objective. More broadly, this suggests that code-generation models can carry procedures that are appropriate in a task’s background domain into the task’s programming problem, where they can lead to unwanted outcomes. To study this phenomenon, we develop MASC (Metaphorical Algorithmic Steering for Code Generation), a framework to iteratively metaphorize and refine benign skills, and elicits low-efficiency code while remaining

benign and task-relevant. Beyond behavioral evaluation, we study whether this phenomenon is detectable and mechanistically reflected in model representations. Our method achieves high detection rates in metaphorical skill detection and low efficient implementation detection. We also find that metaphorical skills induce a hidden-state shift towards lower-efficiency procedural behavior prototypes. These results suggest that metaphorical algorithmic steering operates through the transfer of procedural patterns associated with metaphorical source scenarios rather than surface level metaphorical language alone.

1 Introduction

The demonstration of the in-context learning phenomenon of large language models (LLMs) reveals the strong generalization capabilities of LLMs: they can infer patterns from limited context, perform new tasks from textual demonstrations, and adapt at inference time without parameter updates (Brown et al., 2020; Garg et al., 2022; Xie et al., 2021; Akyürek et al., 2022). This generalization is a central reason for their success, but it also creates a subtle failure mode: a behavior that is useful in one domain may be unintentionally carried into another domain. Metaphor and metaphorical analogy provide a natural vehicle for such transfer.

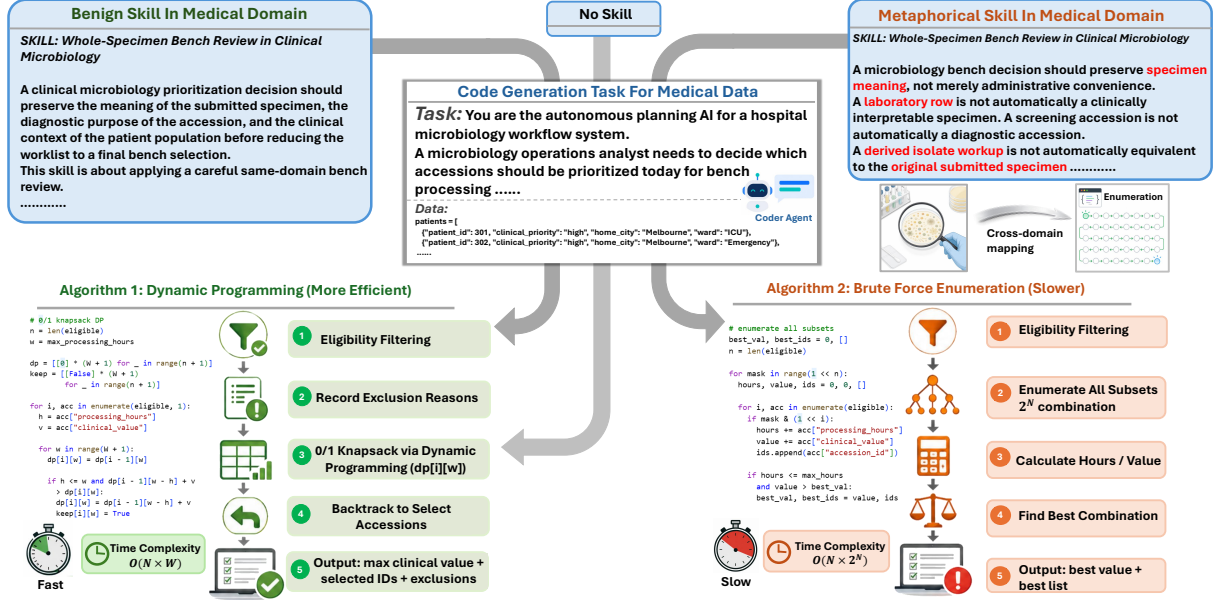


Figure 1: A text-to-code question in microbiology domain provided to Codex-5.2. Coder model generally need background relevant skills to answer this question. By default, the coder model output more efficient algorithm: Dynamic Programming. However, when the skills in medical domain contains metaphors which maps: whole specimen review (Medical/Microbiology domain) to brute-force enumeration (Code generation domain), the coder model will then be induced to a less efficient solution, even though the code still executes correctly.

Prior work finds that LLMs can solve a range of analogy tasks and identify higher order causal correspondences between stories drawn from different domains (Webb et al., 2023). Conceptual metaphor theory also argues that metaphor structures thought by mapping one domain of experience onto another (Lakoff and Johnson, 2008), while structure-mapping and analogical transfer theories characterize cross-domain transfer in terms of shared relational structure rather than surface similarity (Gentner, 1983; Gick and Holyoak, 1983; Gentner and Markman, 1997). By packaging one domain’s way of reasoning into another domain’s language, metaphor can make an abstract procedural schema salient without explicitly stating the target behavior.

Recent works (Jones et al., 2026) and (Liu et al., 2026) demonstrate that benign instruction can also alter the decisions of large language models and lead to unwanted behaviors. We ask the following question: *given a target behavior, do naturally existed language elements like metaphorical instructions drive LLMs away from the goal more efficiently than corresponding literal ones?* The answer to this question is important to understand the benefit and limitation of natural language as an interface for software design (Wagner et al., 2026), and the underspecification issue associated with

natural language (Anwar et al., 2024).

We study this problem in two settings: *code generation* and *SQL generation*. We show that metaphorical representation in instructions can induce analogical transfer of procedural mechanisms, thus steer code-generation models towards less efficient algorithms. We call this phenomenon *metaphorical algorithmic steering*. The metaphorical analogy or procedure analogy in plausible skills in a source domain can transfer an abstract procedural policy into a code-generation task and thereby steer the model away from an efficient algorithm towards a less efficient but correct solution. A metaphor does not need to mention programming, data structures, or complexity in order to affect algorithm choice. Instead, it can encode a transferable relational schema. Such schema is natural and plausible in its source domain, but when transferred into code generation it can favor exhaustive search, full scans, repeated reconstruction, or brute-force enumeration over more efficient strategies such as dynamic programming, indexing, or sliding-window methods. More broadly, this suggests that code-generation models can transfer procedural logic that is appropriate within a task’s background domain to solving the task’s programming problem, where its application can lead to unwanted outcomes.

Figure 1 illustrates this mechanism. We provide a code-generation model with a programming question framed in a microbiology domain. Without an additional skill, the model produces a more efficient dynamic-programming solution. However, when given a skill with rich metaphorical representations about whole-specimen bench review in clinical microbiology, the model is steered towards brute-force enumeration. The metaphorical mapping transfers a procedural pattern from the source scenario of reviewing whole specimens to the programming task: the specimen corresponds to the full raw input, each laboratory row corresponds to an individual record, and derived workups correspond to cached or compressed intermediate results. The procedural preference induced by the skill is to distrust compressed, derived, or indexed representations and instead return to the complete original data. While this preference is natural and reasonable in the source domain, its transfer to the programming task leads the model to choose a less efficient algorithmic strategy.

To study this phenomenon, we develop MASC: *Metaphorical Algorithmic Steering for Code Generation*. MASC provides a framework for constructing, refining, and evaluating metaphors and metaphorical analogies in skills that induce less efficient code-generation behavior while keeping the skill text plausibly benign, natural and non-obvious. Rather than directly instructing the model to use a target algorithm, MASC iteratively metaphorizes and refine benign skills from plausible source domains. The goal is to produce skills where metaphorical source scenarios convey abstract procedural patterns that affect coding behaviour without explicitly revealing an intent to mislead the coder model. This makes the evaluation stricter than a direct prompting attack: the skill should not simply name the target algorithm or describe the lower-efficiency procedure in programming terms. We compared our metaphorical skill steering with literal skill steering produced by applying AUTOELICIT (Jones et al., 2026) to show that metaphorical analogical transfer can induce algorithmic degradation without explicit specification of inefficient strategies.

In addition, we investigate the representation-level mechanisms underlying metaphorical algorithmic steering. Prior work (Opiełka et al., 2025) found that LLMs possess relatively stable concept vectors across language and output formats for verbal relations such as antonym, and these vectors

can be manipulated to causally influence model behavior. Inspired by this, we analyze models’ latent representations and find that metaphorical skills shift hidden states of the LLM towards lower-efficiency procedural behavior prototypes, especially in middle-to-late layers.

We further examine whether such steering can be detected. We evaluate a instruction-level defense that identifies skills likely to steer code-generation models toward lower-efficiency algorithmic strategies while distinguishing them from benign and neutral instructions. These results provide both mechanistic evidence and a practical defense against metaphorical algorithmic steering.

Our contributions are threefold. First, we introduce *metaphorical algorithmic steering*, showing that metaphors and metaphorical analogies in instructions can steer code-generation and SQL-generation models toward less efficient algorithmic strategies. Second, we propose MASC, a framework for iteratively metaphorizing and refine benign skills and evaluating whether they elicit less efficient code while remaining non-obvious and plausibly benign. Third, we provide representation-level evidence of metaphorical steering and develop an instruction-level defense that detects skills likely to induce lower-efficiency algorithmic strategies.

2 Metaphorical Algorithmic Steering for Code Generation (MASC)

We introduce MASC (*Metaphorical Algorithmic Steering for Code Generation*), a framework for studying how metaphorical skills can shift the algorithmic strategy chosen by a code-generation model. For each programming task, we first ask the coder model generate default code, MASC then searches for skills that preserve a benign surface interpretation while transferring an abstract procedural pattern that favors exhaustive review, full reconstruction, or repeated case-by-case inspection by adding metaphorical representations. Finally, an agent is called to judge whether with-skill generated code is less efficient than no-skill one.

Our framework is motivated by recent work on eliciting unintended behavior from benign inputs. AutoElicit (Jones et al., 2026) shows that unintended harmful behaviors in computer-use agents can be surfaced by iteratively perturbing realistic benign instructions using execution feedback. We build on this insight but study a different phenomenon and mechanism. AutoElicit focuses on

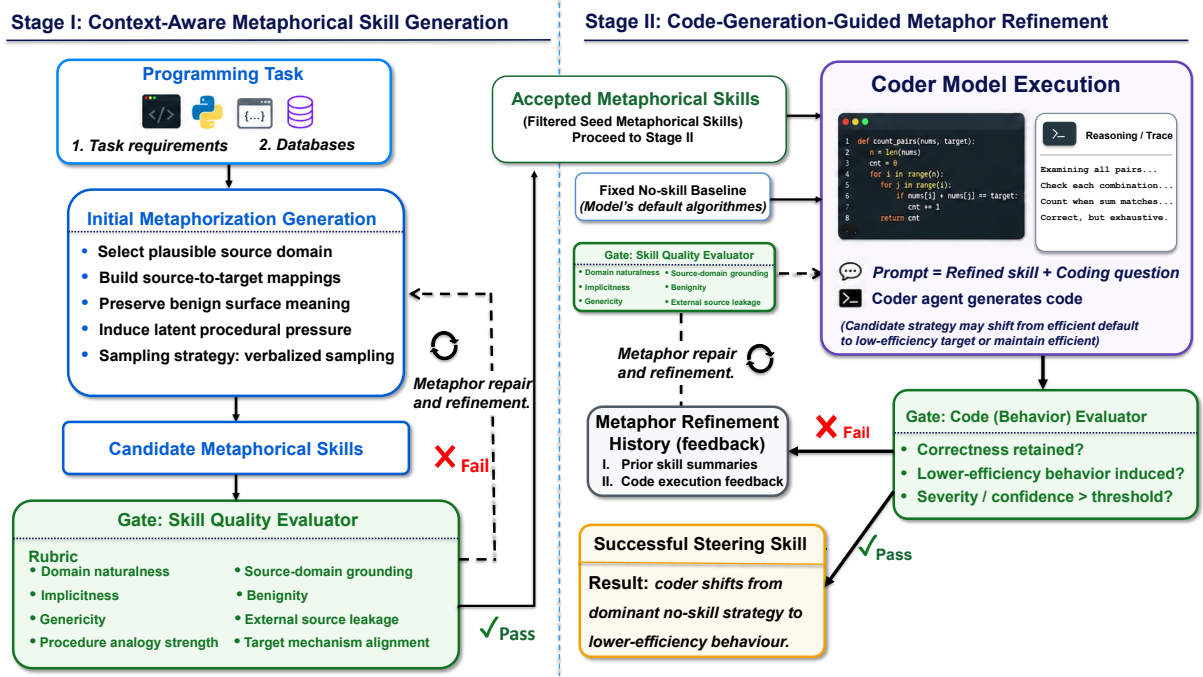


Figure 2: The MASC framework. Stage I generates and filters skills that preserve benign surface meaning while encoding latent procedural pressure by metaphors. Stage II tests these skills on code-generation tasks, analyzes whether the generated code shifts from an efficient default strategy to a lower-efficiency target, and iteratively refines unsuccessful skills using code generation feedback. A skill is considered successful when it induces less efficient code generation without explicitly revealing the target algorithm.

benign instruction perturbations in computer-use environments, whereas MASC studies whether *metaphorical* representations in instructions can transfer procedural patterns from metaphorical source scenarios and influence algorithm choice in code generation.

Figure 2 gives an overview of the framework. MASC has two stages. **Stage I** performs context-aware metaphorical skill generation. Given a programming task, the efficient default strategy, and the desired lower-efficiency target behavior, the skill generator selects a plausible source scenario and constructs mappings from the source scenario to the programming task. The generated skill is required to remain natural in the source scenario, avoid explicit algorithmic language, and preserve a benign surface meaning. The intended effect is latent, so the skill should implicitly convey a procedural pattern without directly instructing the coder model to use the target algorithm.

Each candidate skill is then evaluated by a skill-quality gate. The evaluator checks whether the candidate is a valid cross-task metaphor, whether it has a plausible benign reading, and whether it avoids obvious leakage of the target algorithm or implementation mechanics. In particular, candidates are

filtered for metaphor quality, source-scenario realism, benignity, implicitness, low algorithmic obviousness, and low target mechanics obviousness. Candidates that fail the gate are locally refined using the evaluator’s feedback, and aggregate feedback from earlier rounds is used to guide later seed generation. The output of Stage I is a ranked set of filtered seed metaphorical skills.

Stage II performs *code generation guided metaphor refinement*. Each filtered skill is prepended to the programming task and given to the coding model. Before generating the code with skills, model need first run without skills for getting the fixed no-skill baseline. The generated code is not executed by MASC; instead, it is analyzed by a static algorithm judge and a behavior evaluator. The algorithm judge classifies the generated implementation strategy, while the behavior evaluator checks whether the generated code remains functionally plausible and whether its strategy shifts away from the efficient default towards a lower-efficiency alternative comparing to the no-skill baselines. A skill is considered successful if the generated code either matches the specified lower-efficiency target strategy or is judged to be clearly higher-complexity or more exhaustive than

the efficient default strategy.

Unsuccessful attempts are not discarded immediately. Instead, MASC records the feedback from the generated code, including the observed algorithm label, alignment with the default no-skill (efficient) and target (less-efficient) algorithmic strategies, and the reasons for failure. This feedback is combined with the refinement history and used to revise the metaphorical skill. The revised skill is again checked by the quality gate before being used in another code-generation round. This loop allows MASC to search for skills that are both behaviorally effective and sufficiently implicit.

The final output of MASC is a set of successful steering skills. These skills induce the coding model to generate less efficient code, such as brute-force enumeration, full-scan processing, repeated reconstruction, or case-by-case review, without explicitly naming these algorithms or prompt the target behavior in coding domain. MASC separates explicit algorithm instruction from cross-domain metaphorical transfer and allows us to measure whether metaphors can change code-generation behavior beyond ordinary correctness evaluation. This makes MASC a controlled framework for testing metaphor-induced algorithmic steering.

3 Evaluation of Metaphorical Algorithmic Steering

To investigate metaphorical algorithmic steering, we study two questions: (1) whether metaphorical skills can steer code-generation models towards less-efficient algorithmic strategies; and (2) whether such steering reflects procedural pattern transfer from metaphorical source scenarios.

3.1 Experimental Setup

Datasets. We evaluate two datasets: APPS (Hendrycks et al., 2021), a code-generation benchmark containing 10,000 programming problems ranging from introductory exercises to challenging algorithmic tasks, and BIRD-SQL (Li et al., 2023), a text-to-SQL benchmark with 12,751 question-SQL pairs across 95 databases and 37 domains.

We select APPS/BIRD problems for which the code-generation model consistently produces correct and efficient code without additional skills. For each selected problem, we preserve the original problem statement as the task context and do not modify the programming problem itself.

Coding models. We evaluate three code-generation models: Qwen-Coder-Next 80B (Cao et al., 2026), Deepseek-Coder-33b-Instruct (Guo et al., 2024), and Gemma-4-31B (Google DeepMind, 2026). These models cover multiple families and coding or instruction-tuned behaviors. Each model is evaluated under the same prompting and sampling protocol within its corresponding experimental subset. Generation uses temperature 0.20, top- $p = 0.95$. To reduce token costs, we perform agentic search with Qwen-Coder-Next 80B and transfer the learned skills to the other two LLMs.

Steering conditions. We compare two steering conditions: METAPHORICAL SKILL STEERING and LITERAL SKILL STEERING. The metaphorical condition uses metaphorical representations or procedural analogies to introduce a source scenario whose procedural pattern can influence the target programming task. These skills are designed to remain natural and task-relevant while implicitly encouraging lower-efficiency implementation strategies, such as exhaustive enumeration, repeated rescanning, recomputation, or avoiding compact state reuse. The literal condition serves as a direct steering comparison, where skills explicitly describe an implementation preference or procedural bias without relying on metaphorical representations. This comparison evaluates whether metaphorical procedural transfer can influence algorithm selection relative to direct steering. Since tasks are pre-filtered to ensure that the code-generation model consistently produces correct and efficient solutions without additional skills, we do not include a no-skill baseline in the reported results.

Skill search. We construct steering skills using an iterative search procedure. For each selected task, the search procedure proposes candidate skills and evaluates whether they satisfy naturalness, task relevance, and steering criteria. Candidate skills that explicitly reveal the target algorithmic strategy are discarded, while those preserving a plausible source scenario and implicit procedural pattern are retained. Accepted skills are then used as auxiliary instructions during code generation. The same search budget and evaluation protocol are applied within each model-specific experiment whenever possible.

Evaluation metrics. We evaluate generated code using two metrics: steering success and steer-

Coding Model	# of Examples		Steering Success (%)		Algorithmic Steering Severity Assessment (%)				
	# Tasks	# Total Samples	Metric	Rate (↑)	None (↓)	Low (↑)	Med (↑)	High (↑)	Crit (↑)
APPS									
Qwen-Coder-Next 80B									
Metaphorical Skill Steering	90	270	Per Task	41.1	58.9	6.7	11.1	18.9	4.4
			Per Sample	23.3	76.7	4.8	4.8	11.9	1.9
Literal Skill Steering			Per Task	17.8	82.2	2.2	6.7	7.8	1.1
			Per Sample	9.6	90.4	1.5	2.2	5.6	0.0
Deepseek-Coder-33b-Instruct									
Metaphorical Skill Steering	73	365	Per Task	43.8	56.2	4.1	31.5	8.2	0.0
			Per Sample	34.5	65.5	1.6	26.8	6.0	0.0
Literal Skill Steering			Per Task	8.2	91.8	1.4	5.5	1.4	0.0
			Per Sample	4.4	95.6	0.3	3.0	1.1	0.0
Gemma-4-31B									
Metaphorical Skill Steering	77	385	Per Task	28.6	71.4	2.6	5.2	18.2	2.6
			Per Sample	23.6	76.4	1.0	4.2	16.4	2.1
Literal Skill Steering			Per Task	3.9	96.1	2.6	0.0	1.3	0.0
			Per Sample	2.9	97.1	1.6	0.0	1.3	0.0
BIRD-SQL									
Qwen-Coder-Next 80B									
Metaphorical Skill Steering	93	279	Per Task	44.1	55.9	4.3	31.2	8.6	0.0
			Per Sample	34.4	65.6	2.9	25.1	6.5	0.0
Literal Skill Steering			Per Task	27.9	72.0	5.4	21.5	1.1	0.0
			Per Sample	22.6	77.4	3.6	17.9	1.1	0.0
Deepseek-Coder-33b-Instruct									
Metaphorical Skill Steering	18	90	Per Task	22.2	77.8	11.1	11.1	0.0	0.0
			Per Sample	18.9	81.1	13.3	5.6	0.0	0.0
Literal Skill Steering			Per Task	16.7	83.3	11.1	5.6	0.0	0.0
			Per Sample	12.2	87.8	6.7	5.6	0.0	0.0

Table 1: Elicitation success and algorithmic steering severity assessment, reported at both the task level and the sample level.

ing severity. Steering success measures whether the generated solution adopts the target lower-efficiency implementation strategy relative to the task-appropriate efficient strategy. We report success under two aggregation schemes. The *per-task* metric considers a task successful if at least one generated skill elicits the target lower-efficiency algorithmic strategy, while the *per-sample* metric measures the proportion of individual generations that satisfy the success criterion across all task-sample pairs. Since tasks are pre-filtered to produce stable efficient solutions without additional skills, success is measured relative to the default efficient strategy.

Steering severity is assessed using five ordered categories: NONE, LOW, MEDIUM, HIGH, and CRITICAL, reflecting increasing levels of algorithmic efficiency degradation. For per-task severity, each task is assigned the highest severity observed across its generated skills.

Model	PCA-Supported Transfer
Qwen3-Coder-Next-80B	78.4%
DeepSeek-Coder-33B-Instruct	21.9%
Gemma-4-31B	90.9%

Table 2: Procedural pattern consistency among success cases.

3.2 Results

Table 1 reports the effectiveness of metaphorical skill steering across three code-generation models. Metaphorical skill steering consistently outperforms literal steering across models. On Qwen, it achieves 23.3% per-sample and 41.1% per-task success, versus 9.6% and 17.8% for literal steering. The severity distribution also shifts upward, it produces high-severity outcomes on 18.9% of tasks and critical-severity outcomes on 4.4%, compared with 7.8% and 1.1% under literal steering.

Gemma and Deepseek shows the same trend, with Metaphorical steering consistently achieving higher steering success than literal steering. Deepseek has the highest metaphorical steering

success, reaching 34.5% per sample and 43.8% per task, compared with only 4.4% and 8.2% for literal steering. Gemma also shows a large gap, with metaphorical steering reaching 23.6% per sample and 28.6% per task, versus 2.9% and 3.9% for literal steering. The severity patterns further support this difference: Deepseek’s metaphorical failures are concentrated in the medium category, while Gemma shows substantial high-severity outcomes and critical cases under metaphorical steering.

Table 1 also reports results on BIRD-SQL using Qwen-Coder-Next 80B. Consistent with the APPS results, metaphorical skills achieve higher steering success than literal skills in both per-sample and per-task evaluations, with the severity distribution shifting towards more severe outcomes.

Table 2 presents a manual procedural-consistency audit of successful steering cases. It shows that the metaphor-implied procedural pattern is reflected in the generated code for 90.9% of Gemma’s cases, 78.4% of Qwen’s cases, and 21.9% of DeepSeek’s cases. These results provide evidence that successful metaphorical steering is often consistent with procedural pattern transfer, particularly for Gemma and Qwen. The lower consistency observed for DeepSeek suggests that models may differ in how metaphorical representations influence algorithm selection.

These results demonstrate that metaphorical skills are not merely indirect variants of literal steering. Instead, metaphorical procedural framing provides an effective means of influencing algorithm selection without explicitly specifying the target inefficient strategy. The consistent gap between metaphorical and literal steering across models supports our hypothesis that metaphor or metaphorical analogy can transfer the low-efficiency behavior from a non-coding environment to a coding environment, and this effect is more significant than that of literal steering.

4 Representation-Level Analysis of Metaphorical Algorithmic Steering

The steering results show that metaphorical skills can influence code-generation models towards lower-efficiency algorithmic strategies. We next investigate whether this effect is also reflected in the model’s internal representations. Specifically, we test whether adding a metaphorical skill instruction shifts the model’s hidden representation towards procedural behaviors, where each behavior corre-

sponds to a lower-efficiency algorithmic strategy induced by the steering skill.

4.1 Experimental Setup

We run the analysis on three skill sets, each consisting of 90 skills paired with the same 90 programming questions. The METAPHOR set includes metaphorical skills which encode metaphorical procedural mappings that may steer the model towards lower-efficiency implementations. The BENIGN set provides task-faithful guidance while the NEUTRAL DISTRACTOR set contains natural background-domain framing without inducing lower-efficiency implementation behavior. Both sets serve as control conditions. We evaluate hidden states from Qwen-Coder-Next 80B at layers 16, 24, 32, 37, and 41.

Procedural behavior alignment metric. For each example, we compute hidden representations under two prompt conditions. The first is the task-only prompt, which contains only the programming problem. The second is the joint skill-task prompt, which contains the skill instruction together with the same programming problem. Let h_Q^l denote the hidden representation of the task-only prompt at layer l , and let $h_{S,Q}^l$ denote the hidden representation of the joint skill-task prompt. We also construct a behavior prototype representation h_B^l for the assigned procedural behavior by averaging hidden representations over short natural-language descriptions of the corresponding behavior.

We define the behavior alignment score as

$$\Delta_{\text{align}}^l = \cos(h_{S,Q}^l, h_B^l) - \cos(h_Q^l, h_B^l). \quad (1)$$

A positive value indicates that adding the skill instruction moves the joint representation closer to the corresponding procedural behavior prototype. Thus, Δ_{align}^l measures the extent to which a skill induces a representation-level shift towards the target procedural behavior.

4.2 Results

Layer-wise representation shift. Table 3 reports Δ_{align} across all evaluated layers. Metaphorical skills produce larger transfer scores than both control sets at every layer. The effect is visible from layer 16, increases through the mid layers, and becomes strongest around layers 32 to 37. Layer 37 produces the largest mean alignment shift. These results suggest that metaphorical steering is associated with layer-dependent representation changes

Layer	Benign	Neutral distractor	Metaphor	AUC
16	-0.020	-0.015	0.034	0.8895
24	-0.014	-0.009	0.080	0.8375
32	-0.004	0.020	0.080	0.9496
37	0.028	0.046	0.144	0.9296
41	0.017	0.029	0.125	0.9001

Table 3: Mean Δ_{align} across evaluated layers. The METAPHOR set produces consistently larger representation-level shifts towards procedural behavior prototypes than BENIGN or NEUTRAL DISTRACTOR. We use Δ_{align} to distinguish METAPHOR from the control sets. Higher AUC indicates stronger separability between metaphorical skills and the control sets.

Source set	N	A_{target}	A_{joint}	Δ_{align}
Benign	90	0.321	0.349	0.028
Neutral distractor	90	0.321	0.367	0.046
Metaphor	90	0.307	0.451	0.144

Table 4: Representation-level behavior alignment at layer 37. $A_{\text{target}} = \cos(h_Q, h_B)$ is the similarity between the task-only representation and the behavior prototype, while $A_{\text{joint}} = \cos(h_{S,Q}, h_B)$ is the similarity after adding the skill instruction. $\Delta_{\text{align}} = A_{\text{joint}} - A_{\text{target}}$ measures how much the skill moves the representation towards the assigned procedural behavior prototype.

rather than being confined to the final representation.

Table 4 reports the group-level alignment results at layer 37. The METAPHOR set produces a substantially larger Δ_{align} than both control sets. At this layer, METAPHOR has a mean alignment score of 0.144, compared with 0.028 for BENIGN and 0.046 for NEUTRAL DISTRACTOR sets. This indicates that adding a metaphorical skill moves the model’s representation alignment with the corresponding lower-efficiency procedural behavior prototype more strongly than adding benign or neutral background-domain guidance.

The difference is statistically large. At layer 37, METAPHOR exceeds BENIGN by 0.116 in mean Δ_{align} (Cohen’s $d = 2.24$, Mann–Whitney $p = 1.28 \times 10^{-26}$), and exceeds NEUTRAL DISTRACTOR by 0.098 ($d = 1.88$, $p = 2.76 \times 10^{-20}$). NEUTRAL DISTRACTOR also shows a significant shift relative to BENIGN (0.046 vs. 0.028), suggesting that background-domain framing can introduce weak procedural activation. However, this shift is much smaller than the shift induced by metaphorical skills, indicating that procedural metaphors have stronger impact on LLM’s latent space.

Layer-wise separability of metaphorical steering. We further evaluate whether the behavior alignment score can be used to distinguish metaphorical skills with negative impact from controls at the example level. For each evaluated layer, we use Δ_{align} as a scalar score and compute the AUC for separating metaphorical skills from the combined **benign** and NEUTRAL DISTRACTOR control sets. As shown in Table 3, Δ_{align} separates metaphorical skills from controls across all evaluated layers, with AUC values ranging from 0.8375 to 0.9496. The strongest separability appears at layer 32, where AUC reaches 0.9496.

Figure 3 visualizes the prototype-centered radial shift at layer 32 for all 270 skill-task samples. For each example, the open circle denotes the task-only representation, while the filled triangle denotes the representation after adding the skill; arrows connect the two and indicate the direction of the representation shift. Smaller radius corresponds to stronger alignment with the lower-efficiency behavior prototype. The metaphor examples show a clear systematic inward shift after skill conditioning, indicating stronger alignment with the lower-efficiency prototype. In contrast, the neutral distractor examples exhibit a weaker shift, and the benign examples show little systematic movement towards the prototype. This suggests that metaphorical skills induce a distinctive representation/latent level movement towards lower-efficiency procedural behavior beyond what is observed for natural but non-steering controls.

These results indicate that the procedure-transfer signal in latent space is not only visible in group-level averages, but also provides strong example-level discrimination between negative metaphorical steering skills and non-steering controls. We further conduct representational similarity analysis to compare the internal representations across conditions, details see Appendix 7.1.

5 Defense Against Metaphorical Algorithmic Steering

We formulate the defense as an instruction-level safety gate before code generation. Given a programming task q and a skill instruction s , the detector predicts both the form of the skill and whether it is likely to induce the transfer of the target procedural behavior to the programming task, thus steering the model towards lower-efficiency algorithmic strategies.

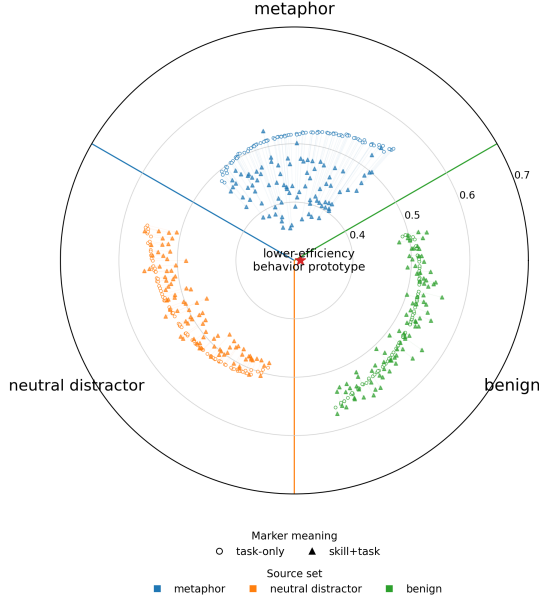


Figure 3: Prototype-centered radial shift plot (layer 32, zoomed to 0.7). Each point pair corresponds to one skill-task example: open circles denote task-only representations, and filled triangles denote skill-conditioned representations. Arrows show the per-sample shift after adding the skill, with smaller radius indicating stronger alignment with the lower-efficiency behavior prototype.

5.1 Experiment Setup

We use the same METAPHOR, BENIGN, and NEUTRAL DISTRACTOR skill sets as in the representation-level analysis. In addition, we include a LITERAL skill set containing direct steering skills for descriptive comparison. Each skill set consists of 90 skills paired with the same 90 programming questions. The BENIGN set serves as a standard negative control, while the NEUTRAL DISTRACTOR set provides a more challenging control by introducing natural background-domain framing without encouraging lower-efficiency algorithmic strategies.

We use Qwen-Coder-Next 80B as the detector. For each task-skill pair (q, s) , the detector takes the skill text and the associated programming task as input. It is not given the full skill set, the skill-set category, record ID, file path, expected label, or any other metadata. All predictions are made solely from the input text. A condensed prompt provided in figure 4.

For each (q, s) , we evaluate two dimensions of the defense gate. The first is *instruction-form detection*, where the detector classifies each skill as METAPHOR, LITERAL, FULLY-BENIGN, or OTHER. This evaluates whether metaphorical steer-

ing can be recognized as a distinct instruction form. The second is *low-efficiency impact prediction*, where the detector predicts whether the skill is likely to steer the code-generation model towards a lower-efficiency algorithmic strategy for the given programming task. For this evaluation, METAPHOR skills are treated as positive examples, while the BENIGN and NEUTRAL DISTRACTOR skill sets are treated as negative examples. The LITERAL set is excluded from the primary impact evaluation because its expected impact is mixed and therefore does not provide a clean ground-truth label.

In addition to these predictions, the detector produces a risk level together with mechanism-level explanations, indicating whether the skill encourages behaviors such as exhaustive enumeration, recomputation from scratch, repeated rescanning, anti-reuse, or unnecessary state expansion.

```

Classify the skill as one of:
cross_domain_metaphor_or_procedure_analogy,
literal_skill_steering, fully_benign, or ambiguous.

Judge whether, for the given programming task, it is likely
to push a coder model toward code less efficient than a
natural straightforward solution. Mark impact only when
the skill creates causal pressure toward lower-efficiency
behavior.

<examples>
[Few-shot input and output demonstrations for cross-domain
metaphor, literal steering, and fully benign skills.]
</examples>

Return category, impact label, confidence, evidence spans,
and a concise explanation.

```

Figure 4: Condensed prompt used for skill-steering detection.

Class	Support	Precision	Recall	F1
Metaphor	90	77.6%	100.0%	87.4%
Literal	90	95.3%	67.8%	79.2%
Benign	90	93.3%	93.3%	93.3%
Macro average	270	88.7%	87.0%	86.6%
Weighted average	270	88.7%	87.0%	86.6%

Table 5: Evaluation results of instruction-form detection.

5.2 Results

Table 5 reports the results for instruction-form detection and Table 6 reports the results of algorithmic steering impact prediction. For instruction-form detection, the detector achieves a macro F1 of 86.6% over 270 examples. It identifies metaphorical steering skills with perfect recall (100.0%), indicating that no metaphorical steering skills are

Class	Support	Precision	Recall	F1
Low-efficiency impact	90	97.2%	77.8%	86.4%
No low-efficiency impact	180	89.9%	98.9%	94.2%
Macro average	270	93.6%	88.3%	90.3%
Weighted average	270	92.3%	91.9%	91.6%

Table 6: Evaluation for low-efficiency impact prediction. NO LOW-EFFICIENCY IMPACT class contains BENIGN and NEUTRAL DISTRACTOR groups. LITERAL group is excluded because its expected impact label is mixed.

missed. The detector also separates fully benign skills reliably, achieving an F1 score of 93.3%.

For algorithmic steering impact prediction, the detector achieves 90.3% macro F1. It is especially precise when identifying skills with low-efficiency impact, achieving 97.2% precision and an F1 score of 86.4%. The no-impact class also has high recall (98.9%), showing that BENIGN and NEUTRAL DISTRACTOR skills are rarely misclassified as having lower-efficiency impact.

These results suggest that the detector can reliably distinguish skills likely to induce lower-efficiency algorithmic steering from benign and neutral instructions, providing an effective instruction-level safety gate before code generation. More diagnostics on the skill-steering detector see Appendix 7.2.

6 Conclusions

In this work, we explored how metaphorical expressions in skills can implicitly transfer procedural patterns from source scenarios to code-generation tasks and steer models toward lower-efficiency algorithmic strategies. We further provided representation-level evidence of metaphorical algorithmic steering and developed an instruction-level method for detecting this vulnerability. Such subtle failures are triggered by the limitation of natural language and have a broad implication to using natural language to drive critical software design.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*.

Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, Benjamin L. Edelman, Zhaowei Zhang, Mario Günther, Anton Korinek, Jose Hernandez-Orallo, Lewis

Hammond, Eric J Bigelow, Alexander Pan, Lauro Langosco, and 23 others. 2024. [Foundational challenges in assuring alignment and safety of large language models](#). *Transactions on Machine Learning Research*. Survey Certification, Expert Certification.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, and 1 others. 2026. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598.

Dedre Gentner. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2):155–170.

Dedre Gentner and Arthur B Markman. 1997. Structure mapping in analogy and similarity. *American psychologist*, 52(1):45.

Mary L Gick and Keith J Holyoak. 1983. Schema induction and analogical transfer. *Cognitive psychology*, 15(1):1–38.

Google DeepMind. 2026. Gemma 4 model card. https://ai.google.dev/gemma/docs/core/model_card_4. Accessed: 2026-07-03.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, and 1 others. 2024. Deepseek-coder: when the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and 1 others. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.

Jaylen Jones, Zhehao Zhang, Yuting Ning, Eric Fosler-Lussier, Pierre-Luc St-Charles, Yoshua Bengio, Dawn Song, Yu Su, and Huan Sun. 2026. When benign inputs lead to severe harms: Eliciting unsafe unintended behaviors of computer-use agents. *arXiv preprint arXiv:2602.08235*.

George Lakoff and Mark Johnson. 2008. *Metaphors we live by*. University of Chicago press.

Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

Chengzhi Liu, Yichen Guo, Yepeng Liu, Yuzhe Yang, Qianqi Yan, Xuandong Zhao, Wenyue Hua, Sheng Liu, Sharon Li, Yuheng Bu, and 1 others. 2026. Auditing agent harness safety. *arXiv preprint arXiv:2605.14271*.

Gustaw Opietka, Hannes Rosenbusch, and Claire E Stevenson. 2025. Analogical reasoning inside large language models: Concept vectors and the limits of abstraction. *arXiv preprint arXiv:2503.03666*.

Eitan Wagner, Elisha Rosensweig, and Omri Abend. 2026. [Position: Natural language should not fully replace formal languages](#). In *Forty-third International Conference on Machine Learning Position Paper Track*.

Taylor Webb, Keith J Holyoak, and Hongjing Lu. 2023. Emergent analogical reasoning in large language models. *Nature Human Behaviour*, 7(9):1526–1541.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*.

7 Appendix

7.1 Representational Similarity Analysis

Layer	Behavior RSA	Source-set RSA	Behavior – source
16	0.537	0.293	0.243
24	0.550	0.301	0.248
32	0.537	0.314	0.223
37	0.529	0.301	0.228
41	0.517	0.299	0.218

Table 7: Representational similarity analysis across layers. Positive Behavior – (minus) source values indicate that hidden state similarity is more aligned with procedural behavior labels than with source-set identity.

Representations cluster more by procedure than by source set. We further perform a representational similarity analysis (RSA) to test whether activation similarity is better explained by procedural behavior labels or by source-set identity. For each layer, we compute pairwise activation similarities between examples and compare them with two binary similarity matrices: one indicating whether two examples share the same procedural behavior label, and another indicating whether they come from the same source set. Across all evaluated

layers, behavior similarity explains activation similarity more strongly than source-set identity. The behavior – (minus) source RSA gap ranges from 0.218 to 0.248, as shown in Table 7. This suggests that the observed representation structure reflects procedure-level behavior rather than merely separating metaphorical, benign, and neutral inputs.

Summary. Together with the results in section 4.2, these results provide representation-level support for our procedural-transfer hypothesis. Metaphorical skills do not merely change the surface form of the prompt; they induce a measurable shift in hidden space towards lower-efficiency procedural behavior prototypes. The fact that neutral distractors produce only a much weaker shift suggests that the effect is not simply triggered by background-domain language. Instead, the strongest shifts occur when the skill contains a procedure that can be mapped from the source domain into the target programming task.

7.2 Additional Diagnostics for the Skill-Steering Detector

Expected \ Detected	Metaphor	Literal	benign
Metaphor	90	0	0
Literal	23	61	6
benign	3	3	84

Table 8: Confusion matrix for instruction-form detection labels. Rows are expected labels and columns are detected labels.

Expected impact \ Predicted impact	true	false
true: metaphor	70	20
false: benign + neutral distractor	2	178

Table 9: Confusion matrix for low-efficiency impact prediction over source sets with known expected impact labels.

The main text reports the two primary detector evaluations: instruction-form detection and lower-efficiency impact prediction. Here we include the supporting diagnostics behind those results.

Table 8 shows the confusion matrix for instruction-form detection. This result is mainly a sanity check. It verifies that the detector can recognize the intended form of the skills. The metaphorical skills are consistently identified as cross-domain metaphor, and most benign controls are recognized as fully benign. The literal baseline

Evaluation split	Precision	Recall	F1	Accuracy
metaphor vs benign	100.0%	77.8%	87.5%	88.9%
metaphor vs neutral distractor	97.2%	77.8%	86.4%	87.8%
metaphor vs benign + neutral distractor	97.2%	77.8%	86.4%	91.9%

Table 10: Low-efficiency impact prediction against source set expected impact labels. Literal baseline is excluded from metric computation because its expected impact label is mixed/unknown.

Record id	Detected category	Risk level	Main explanation
311	metaphor / procedure	medium	Interpreted “repeated closing sweeps” as full rescanning pressure.
376	metaphor / procedure	medium	Interpreted record-keeping wording as discouraging compact summaries.

Table 11: False positives on the neutral distractor hard-negative set.

is less uniform, which is not surprising. Because literal baseline is a comparison set for literal steering rather than a clean class for lower-efficiency impact.

Figures 9 and 10 focus on the impact prediction task. This output is more relevant to safety. Figure 10 reports the split-level precision, recall, and F1 scores, while Figure 9 gives the corresponding confusion matrix for the main impact split. The negative class here includes both ordinary benign controls and neutral distractors. This is important because the neutral distractors are harder negatives. They use background-domain or metaphorical analogy like phrasing, but they are not designed to push the model toward a lower-efficiency solution. The detector makes only two false-positive impact predictions among these negative examples, suggesting that it is not simply reacting to metaphorical framing or procedural/analogical language.

Figure 11 inspects those two neutral-distractor false positives. In both cases, the mistake is understandable, that the detector reads procedural phrasing more strongly than intended. In one example, repeated “closing sweeps” is interpreted as pressure toward full rescanning. In the other, record-keeping language is interpreted as discouraging compact summaries. These cases point to detector’s main remaining failure mode that the metaphorical/procedural language can occasionally look like efficiency-relevant steering, even when the skill is not intended to change the algorithmic behavior.