THINK LESS, CODE BETTER: PROBING WHEN CHAIN-OF-THOUGHT HURTS AND HOW TO ROUTE AROUND IT

Rajarshi Ghoshal **Debadri Basak** **Salma E. Abdelhalim**

Pratibha K. Arora
College of Computing, Georgia Institute of Technology
{rghoshal3, dbasak7, sabdelhalim3, parora65}@gatech.edu

ABSTRACT

Chain-of-Thought (CoT) prompting is the dominant strategy for eliciting step-by-step reasoning in LLMs. We present a controlled study of when this reasoning augmentation helps versus hurts in code generation—a task requiring deductive reasoning under formal constraints—spanning three code-specialized architectures: a 2×2 design with Qwen2.5-Coder-1.5B and DeepSeek-Coder-1.3B (each in base and instruction-tuned variants) evaluated on HumanEval, MBPP, and LiveCodeBench, plus a preliminary evaluation of CodeLlama-7B.

Our key finding is that *instruction tuning reverses CoT's effect* for Qwen on the same base architecture: CoT significantly improves the base model (+13.4%, $p<0.001$) but significantly degrades the instruction-tuned variant (-15.2%, $p<0.001$). DeepSeek remains insensitive to CoT regardless of training regime (all $p>0.2$), demonstrating that this sensitivity is architecture-specific.

Layer-wise probing reveals that all four models encode prompt type by Layer 1–4 ($>$90% accuracy). Critically, this early encoding is universal—present in models where CoT helps, hurts, or has no effect—demonstrating that *representation does not determine interpretation*: the same structural encoding drives divergent downstream behavior depending on training regime.

Building on these mechanistic findings, we develop a probe-guided style router that selects from 12 prompt styles per problem using a single forward pass (84ms overhead). The router is statistically indistinguishable from the best single fixed style in 7/8 settings (McNemar's, all $p_{\text{Best}}>0.1$) and significantly outperforms CoT in the highest-variance setting (Qwen Instruct/HumanEval, $p=0.012$, $h=+0.40$).

Our findings argue that reasoning augmentation via CoT should not be applied blindly: its effect depends on architecture and training regime in ways that are mechanistically detectable from early-layer activations. The probe provides a practical path to model-aware reasoning-strategy selection without prompt engineering.

1 INTRODUCTION

Chain-of-Thought prompting (Wei et al., 2022) has become a default strategy for eliciting step-by-step reasoning in LLMs. The assumption—that encouraging explicit reasoning universally helps—has driven its widespread adoption, including in code generation, a task requiring deductive reasoning to translate natural-language specifications into formally correct programs. But is this assumption warranted?

We present evidence that it is not. Through controlled experiments across three code-specialized architectures (Qwen2.5-Coder, DeepSeek-Coder, CodeLlama), four model variants (base and instruction-tuned), and three benchmarks, we show that CoT's effect on code generation is *model-specific* and can be significantly harmful. Our central finding: on Qwen2.5-Coder-1.5B, CoT im-

proves the base model by +13.4% but degrades the instruction-tuned model by -15.2%—both statistically significant ($p<0.001$), on the same base architecture.

This finding has practical implications. Developers routinely apply CoT prompting without considering whether their specific model benefits from it. Our results show this can *reduce* reliability by up to 15 percentage points.

To understand *why* CoT's effect varies, we perform layer-wise probing across all four model variants. We find that prompt-type information (Direct vs. CoT) becomes linearly separable in early layers (Layer 1–4) across all models, regardless of whether CoT helps or hurts. This indicates that the models recognize CoT as a structural pattern early, but their downstream *interpretation* of that pattern—shaped by instruction tuning—determines its behavioral effect.

Contributions.

1. **CoT reversal under instruction tuning (primary finding):** For Qwen2.5-Coder, instruction tuning reverses CoT's effect on the same base architecture (+13.4% base, -15.2% instruct, both $p<0.001$). DeepSeek is insensitive regardless of regime. CodeLlama-7B base is also significantly hurt by CoT (-6.7%, $p=0.022$). Across three architectures, CoT effects are heterogeneous—demonstrating the 2×2 study is the first to isolate training regime as the key variable, replicated across three benchmarks including the contamination-free LiveCodeBench.
2. **Mechanistic explanation via internal representations:** Layer-wise probing shows all models encode prompt type (reasoning vs. direct) by Layer 1–4 ($>90\%$ accuracy). This universal early encoding—present whether CoT helps or hurts—demonstrates that *representation does not determine interpretation*: the same internal signal for "this is a reasoning prompt" drives opposite behavioral outcomes depending on training regime. This sheds light on how LLMs internally represent and process reasoning instructions.
3. **Probe-guided style router (practical tool):** A lightweight MLP probe, trained once on public benchmarks, routes any problem to one of 12 prompt styles via a single 84ms forward pass. It is statistically indistinguishable from the best fixed style in 7/8 settings (McNemar's, $p_{\text{Best}}>0.1$) and significantly outperforms CoT where CoT is most harmful ($p=0.012$). No prompt engineering required.

2 Related Work

Mechanistic Interpretability. MI aims to understand neural network computation by analyzing internal representations (Elhage et al., 2022; Olah et al., 2020). Linear probing (Alain & Bengio, 2016) is a standard technique for measuring when specific information becomes encoded. For code models, prior work showed that pre-trained transformers develop hierarchical representations of code syntax (Wan et al., 2022). Our work extends this to examine how *prompts* modulate internal representations.

When CoT Fails. Recent work identifies settings where CoT degrades performance. Wu et al. (2025) show that 86.3% of LLMs suffer degradation under CoT on clinical text understanding. Gema et al. (2025) demonstrate inverse scaling where extended reasoning reduces accuracy. Our work complements these findings with a controlled comparison isolating instruction tuning as the key variable.

Predicting Success from Activations. Lugoloobi et al. (2026) show that pre-generation activations predict success via linear probes. Ribeiro et al. (2026) and Bui et al. (2025) show hidden states encode code correctness. Moreno Cencerrado et al. (2025) predict accuracy from question-only activations. Anthropic (2024) detect deceptive behavior via probes. These works *predict* outcomes; our contribution is to *act* on predictions by selecting prompts before generation.

Activation Steering. Activation engineering (Turner et al., 2023) modifies hidden states to steer behavior (Wehner et al., 2025). Our approach is complementary: we intervene on the *prompt* guided by activation signals, requiring no modification to model internals.

Automated Prompt Optimization. DSPy (Khattab et al., 2023) treats prompts as learnable parameters. APE (Zhou et al., 2022) uses LLMs to generate candidates. Zi et al. (2025) study prompt

specificity effects. These approaches rely on black-box optimization; our method uses interpretable mechanistic signals to guide selection.

3 METHODOLOGY

3.1 MODELS

We study two code-specialized architectures, each in base and instruction-tuned variants:

- **Qwen2.5-Coder-1.5B** (Hui et al., 2024) (base) and **Qwen2.5-Coder-1.5B-Instruct**: 28 layers, 1.5B parameters, trained on code-heavy data.
- **DeepSeek-Coder-1.3B** (Guo et al., 2024) (base) and **DeepSeek-Coder-1.3B-Instruct**: 24 layers, 1.3B parameters.

This 2×2 design (architecture $\times$ training regime) controls for model size and architecture when isolating the effect of instruction tuning. Instruction-tuned models receive chat-formatted prompts via their native templates; base models receive raw text prompts with no system message or special formatting.

3.2 DATASETS

- **HumanEval** (Chen et al., 2021): 164 Python programming problems.
- **MBPP** (Austin et al., 2021): 500 Python programming problems.
- **LiveCodeBench** (Jain et al., 2024): 381 LeetCode-sourced problems from recent competitions, providing a contamination-free blind evaluation.

3.3 PROMPT STYLES

We define a library of 12 prompting styles spanning four categories:

- **Minimal:** *direct* (no prefix)
- **Reasoning:** *cot* ("Think step by step"), *plan* ("Plan then implement"), *decompose* ("Break into sub-problems")
- **Persona:** *expert* ("As an expert programmer"), *reviewer* ("Write production-ready code")
- **Constraint:** *simple* ("Keep it simple"), *careful* ("Handle edge cases"), *efficient* ("Use efficient algorithm"), *tdd* ("Pass all tests"), *defensive* ("Handle all inputs"), *typed* ("Use type hints")

Each style prepends a short comment prefix to the raw prompt. For the Direct vs. CoT comparison, we use *direct* and *cot*. For style routing, all 12 styles are used.

3.4 EVALUATION

All experiments use greedy decoding (temperature = 0, max 512 tokens) to isolate prompt effects from sampling variance. We report Pass@1 and use McNemar's test for statistical significance on paired binary outcomes. Code correctness is evaluated via execution against test cases.

3.5 LAYER-WISE PROBING

We train binary linear probes at each transformer layer to classify Direct vs. CoT prompts. For each problem, we extract residual stream activations $h_\ell \in \mathbb{R}^d$ at the final token position and train:

$$P(\text{CoT} \mid h_\ell) = \sigma(W_\ell \cdot h_\ell + b_\ell) \tag{1}$$

This identifies the *emergence point* where prompt-type information becomes linearly separable.

3.6 Probe-Guided Style Routing

We develop a *selection probe* that uses the activation of a single direct-prompt forward pass to predict which of the 12 styles will succeed on a given problem. Specifically, we extract the residual stream activation at Layer $\lfloor L/8 \rfloor$ (≈12.5% depth) under the *direct* prompt and train a two-layer MLP:

$$\hat{y} = \text{MLP}(h^{\text{direct}}_{\lfloor L/8 \rfloor}) \in \mathbb{R}^{12} \tag{2}$$

using binary cross-entropy loss (multi-label: multiple styles may pass). At test time, the style with the highest predicted logit is selected. This requires only *one* forward pass regardless of library size, adding negligible overhead. We compare against Direct, CoT, Best Single Style, Random, and Oracle (best possible per-problem) baselines, using a 50/50 train/test split.

4 Results

4.1 Early Emergence of Prompt-Type Representations

Layer-wise probing reveals that prompt-type information emerges remarkably early across all four models (Figure 1).

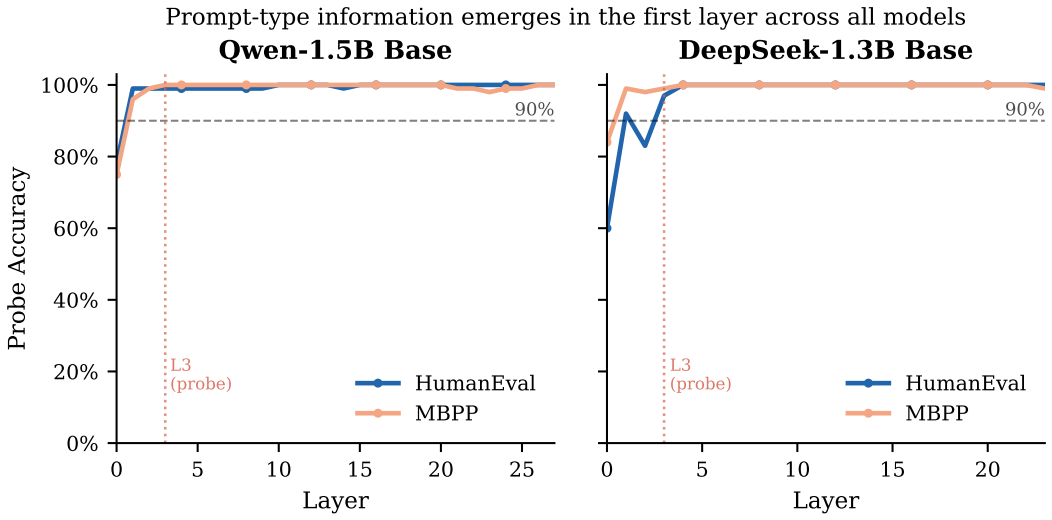

Figure 1: **Prompt-type probe accuracy vs. layer depth.** Both architectures cross 90% accuracy by Layer 1 and saturate at 100% well before 25% depth. The red dashed line marks Layer 3 (12.5% depth), where we extract activations for the style router.

All models achieve >90% probe accuracy by Layer 1 and reach 100% within the first 10–15% of network depth. This early emergence suggests the models recognize CoT as a *structural* pattern—comment syntax and formatting—rather than a *semantic* instruction to reason differently.

The universality of this finding is notable: despite different architectures, training data, and parameter counts, prompt-type encoding follows the same pattern. Yet as we show next, this shared encoding drives divergent downstream behavior.

4.2 Instruction Tuning Reverses CoT's Effect

Figure 2 and Table 1 present our central finding across all model-dataset combinations.

Three key findings emerge:

1. Instruction tuning reverses CoT's effect on Qwen. The base model benefits significantly from CoT across all three datasets (+13.4%, +4.0%, +3.4%; all $p < 0.05$). The instruction-tuned

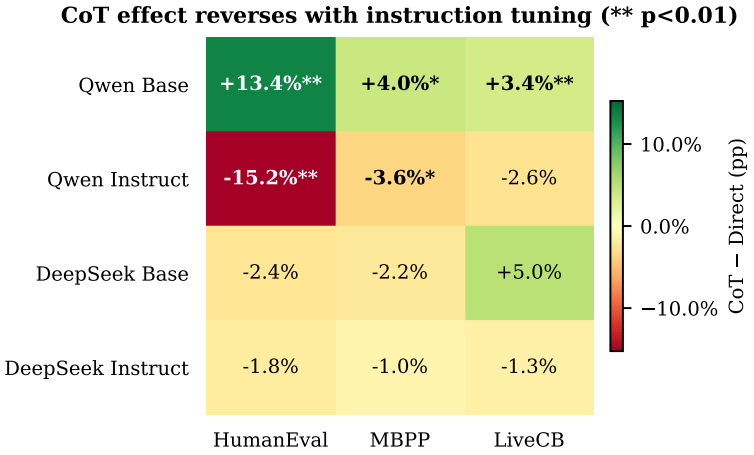

Figure 2: **CoT effect (CoT − Direct, pp) across models and datasets.** Green = CoT helps; red = CoT hurts. Qwen flips sign between base and instruct ($**$ $p<0.01$); DeepSeek is consistently neutral.

Table 1: **Direct vs. CoT Pass@1 across models and datasets.** Δ = CoT − Direct. Bold: statistically significant ($p < 0.05$, McNemar's test with continuity correction). h = Cohen's effect size (small $|h|>0.2$, medium $|h|>0.5$).

Model	Type	Dataset	Direct	CoT	Δ	p	h
Qwen-1.5B	Base	HumanEval	9.8%	23.2%	**+13.4%**	**<0.001**	+0.37
		MBPP	43.8%	47.8%	**+4.0%**	**0.011**	+0.08
		LiveCodeBench	2.4%	5.8%	**+3.4%**	**0.009**	+0.18
	Instruct	HumanEval	64.6%	49.4%	**−15.2%**	**<0.001**	−0.31
		MBPP	50.2%	46.6%	**−3.6%**	**0.012**	−0.07
		LiveCodeBench	14.4%	11.8%	−2.6%	0.112	−0.08
DeepSeek-1.3B	Base	HumanEval	9.1%	6.7%	−2.4%	0.502	−0.09
		MBPP	29.8%	27.6%	−2.2%	0.215	−0.05
	Instruct	HumanEval	67.7%	65.9%	−1.8%	0.505	−0.04
		MBPP	46.4%	45.4%	−1.0%	0.551	−0.02
		LiveCodeBench	7.9%	6.6%	−1.3%	0.267	−0.05
CodeLlama-7B‡ (Roziere et al., 2023)	Base	HumanEval	31.1%	24.4%	**−6.7%**	**0.022**	−0.15

‡CodeLlama-7B evaluated on HumanEval (n=50) from a preliminary experiment; instruct variant and MBPP data not available.

model is significantly *hurt* by CoT on HumanEval (-15.2%, $p < 0.001$) and MBPP (-3.6%, $p = 0.012$), with a consistent negative trend on LiveCodeBench (-2.6%, $p = 0.112$). This is the same architecture and weights—instruction tuning is the only variable.

2. DeepSeek is consistently neutral. Across all datasets and both training regimes, DeepSeek shows no significant CoT effect (all $p > 0.2$). This demonstrates that CoT sensitivity is architecture-specific, not a universal property.

3. CoT degradation replicates on a third architecture. CodeLlama-7B (Roziere et al., 2023) base model shows a significant negative CoT effect on HumanEval (-6.7%, $p = 0.022$). Unlike Qwen, where CoT helps the base model, CodeLlama's base model is hurt—confirming that CoT sensitivity depends on architecture-specific pretraining details, not just the base/instruct distinction.

4. The effect generalizes across benchmarks. The pattern holds on LiveCodeBench, a contamination-free benchmark the models could not have memorized. This rules out benchmark-specific artifacts.

4.3 PROBE-GUIDED STYLE ROUTING

Given the model-specific nature of CoT effects, we test whether activation probes can select the optimal prompting strategy per problem. Using early-layer activations (Layer 3–4, approximately 12.5% depth), we train a selection probe and use it to score all 12 prompt styles for each test problem.

Results are presented in Figure 3 and Tables 2 and 3.

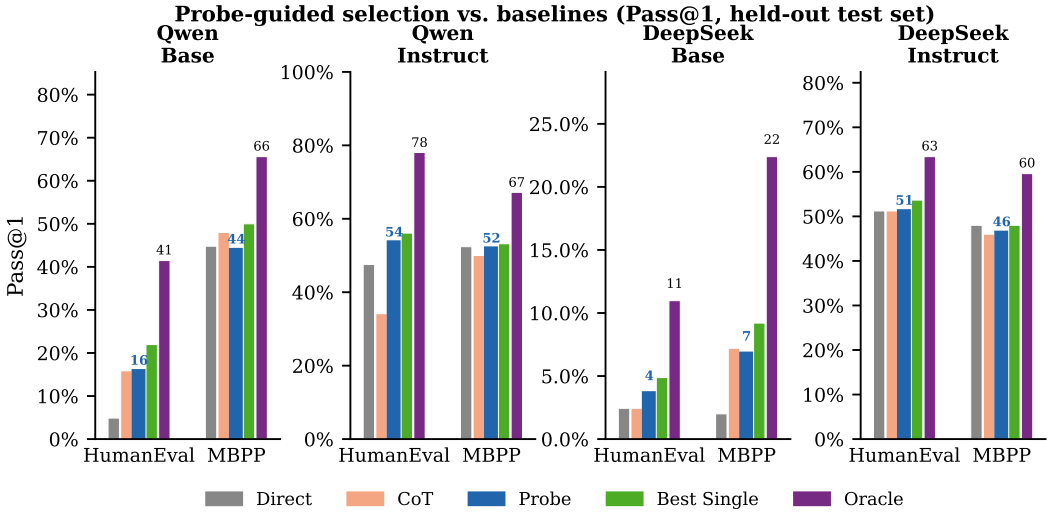

Figure 3: **Style routing results** (Pass@1). Each panel = one model; groups = HumanEval / MBPP. Blue (Probe) and green (Best Single) values are labelled. Oracle is the per-problem upper bound.

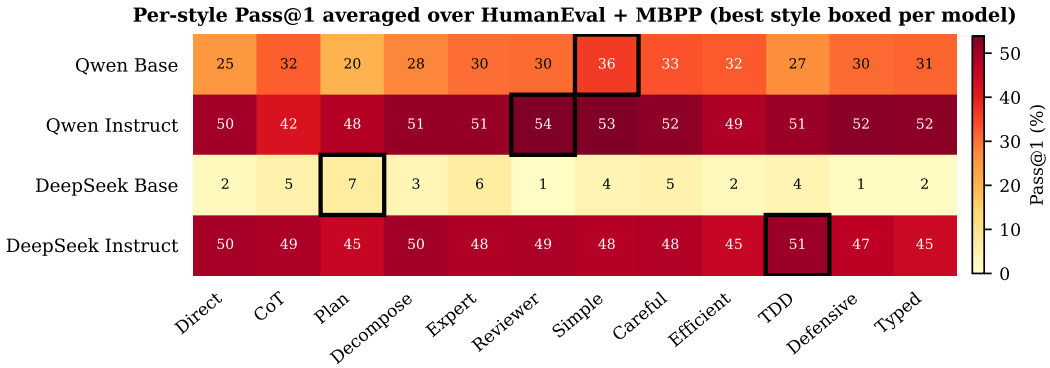

Figure 4: **Per-style Pass@1** averaged over HumanEval and MBPP. The best style per model is boxed. No single style dominates across models: instruct models prefer persona/constraint styles while base models favor reasoning styles (Plan, CoT).

The probe's key property is that it is **statistically indistinguishable from the best single style in 7/8 settings** (McNemar's, all $p_{\text{Best}} > 0.1$). This means a practitioner using the probe achieves performance equivalent to having oracle knowledge of which style is best for their model and dataset—without needing to run any style comparison. The single exception is Qwen-1.5B Base/MBPP ($p = 0.007$), where style variance is high ($\sigma = 0.048$) but the probe fails to exploit it effectively.

The probe's advantage is most pronounced when style sensitivity is high (Table 3). In high-variance settings ($\sigma > 0.04$), the probe ranks 3rd–4th among 12 fixed styles on average. For Qwen Instruct/HumanEval—the setting where CoT is most harmful (-15.2%)—the probe significantly

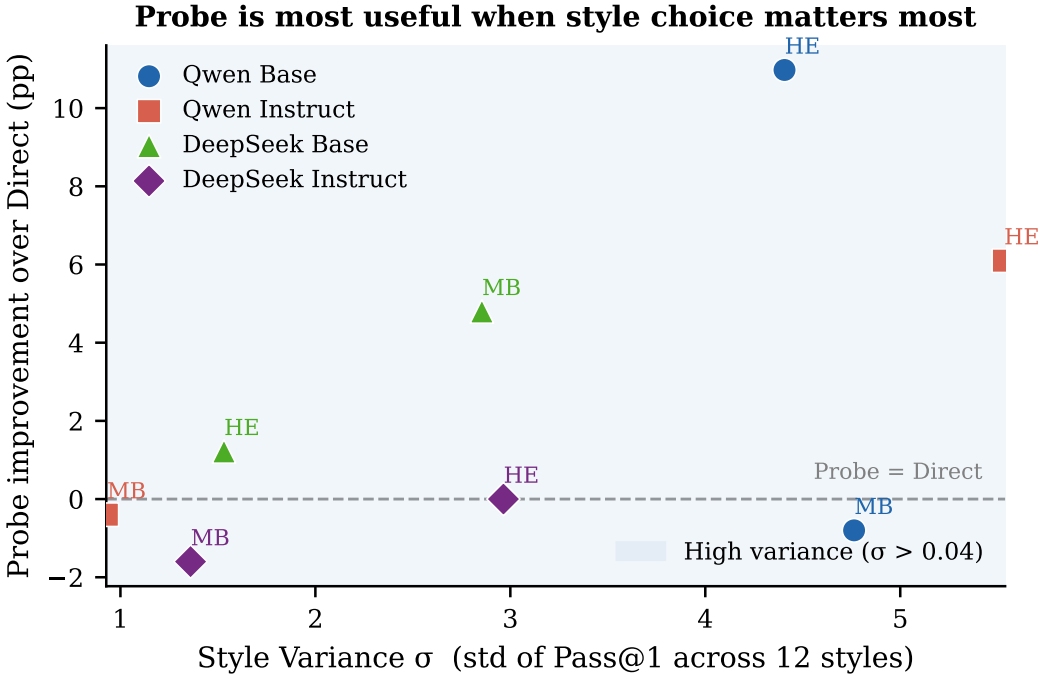

Figure 5: **Style variance predicts probe utility.** Each point is one model-dataset setting. The probe improves most over Direct when style variance σ is high (blue shaded region, σ>0.04); when all styles perform similarly (low σ), the probe provides no advantage.

Table 2: **Probe-guided style routing** on held-out test sets (Pass@1). Probe uses a single direct-prompt activation to select from 12 styles per problem. 95% Wilson CI shown for Probe. p_{CoT}/p_{Dir} = McNemar's test (probe vs. CoT/Direct). p_{Best} = McNemar's probe vs. best single style; n.s. throughout indicates the probe is statistically indistinguishable from oracle-best style.

Model	DS	Probe [95% CI]	CoT	Direct	Best	Oracle	p_{CoT}	p_{Best}
Qwen-1.5B Instruct	HE	**53.7% [42.9, 64.0]**	34.1%	47.6%	56.1%	78.0%	**0.012**	n.s. (0.77)
	MB	52.0% [45.8, 58.1]	50.0%	52.4%	53.2%	67.2%	0.441	n.s. (0.72)
Qwen-1.5B Base	HE	**15.9% [9.5, 25.3]**	15.9%	4.9%	22.0%	41.5%	n.s.	n.s. (0.33)
	MB	44.0% [38.0, 50.2]	48.0%	44.8%	50.0%	65.6%	0.144	0.007†
DeepSeek-1.3B Instruct	HE	51.2% [40.6, 61.7]	51.2%	51.2%	53.7%	63.4%	n.s.	n.s. (0.68)
	MB	46.4% [40.3, 52.6]	46.0%	48.0%	48.0%	59.6%	n.s.	n.s. (0.42)
DeepSeek-1.3B Base	HE	3.7% [1.3, 10.2]	2.4%	2.4%	4.9%	11.0%	n.s.	n.s.
	MB	**6.8% [4.3, 10.6]**	7.2%	2.0%	9.2%	22.4%	n.s.	n.s. (0.24)

†One exception: Qwen-1.5B Base/MBPP (probe 44.0% vs. best 50.0%, p=0.007).

outperforms CoT ($p = 0.012$, $h = +0.40$) and achieves 53.7%, ranking 3rd of 12 fixed styles, while CoT ranks last (34.1%).

In low-variance settings (DeepSeek models, $\sigma < 0.03$), style choice barely matters and all methods perform similarly—the probe is neither helpful nor harmful.

The practical implication is a **model-specific style router**: pre-train the probe once on labeled problems for a given model (e.g., public benchmarks), then deploy it for any new problem at 84ms overhead, with no prompt engineering required.

Table 3: **Style sensitivity (σ) and probe ranking.** σ = std. dev. of Pass@1 across 12 styles (higher = more sensitive to prompt choice). Rank = probe placement among 12 fixed styles. Gap% = fraction of Oracle−Direct gap recovered by probe.

Model	DS	σ	Probe Rank/12	Probe	Gap%
Qwen-1.5B Instruct	HE	0.055	3/12	53.7%	20%
	MB	0.009	5/12	52.0%	−3%
Qwen-1.5B Base	HE	0.044	4/12	15.9%	30%
	MB	0.048	10/12	44.0%	−4%
DeepSeek-1.3B Instruct	HE	0.030	4/12	51.2%	0%
	MB	0.014	4/12	46.4%	−14%
DeepSeek-1.3B Base	HE	0.015	3/12	3.7%	14%
	MB	0.029	5/12	6.8%	24%

Latency overhead. The probe forward pass takes 84ms on Qwen-1.5B (MPS), vs. 2584ms for generation—a **3.3% overhead**. Since the probe replaces the need to generate with all 12 styles to find the best, it reduces total inference cost by up to 12× relative to exhaustive search.

5 Discussion

Why Does Instruction Tuning Reverse CoT's Effect? We hypothesize that base and instruction-tuned models interpret CoT prefixes through fundamentally different lenses. Base models, trained on raw code corpora, encounter CoT-like comments as *documentation patterns* that precede implementations—the comment provides context that helps guide generation. Instruction-tuned models, trained to follow explicit instructions, interpret CoT as an *imperative to reason before coding*. This additional reasoning step introduces noise: the model generates explanatory text that competes with code generation, leading to truncated or incorrect solutions.

The Cohen's h effect sizes ($h = -0.31$ for Qwen Instruct/HumanEval, $h = +0.37$ for Qwen Base/HumanEval) classify as "small" by Cohen's conventions, yet translate to 13–15 percentage point absolute differences that are practically significant for code generation systems.

A concrete mechanism consistent with this hypothesis is **output truncation**: instruction-tuned models under CoT may generate extended reasoning chains before reaching code, hitting the 512-token generation limit before producing a complete solution. Under the direct prompt, the same model proceeds immediately to code and stays well within the token budget. In contrast, base models treat the CoT comment as a documentation-style prefix and proceed directly to implementation, so the CoT prefix *helps* rather than hurts. We leave a systematic output-length analysis—measuring token counts and truncation rates under CoT vs. direct prompting—to future work, but this mechanism is consistent with both the direction and magnitude of the observed effects.

Architecture-Specific Sensitivity. The contrast between Qwen (strong CoT effects) and DeepSeek (neutral) suggests that CoT sensitivity depends on architectural and training details beyond the base/instruct distinction. DeepSeek's robustness to prompting variations may reflect different pre-training data composition or architectural choices that make it less sensitive to prefix conditioning.

Implications for Deployment. Our results argue against the common practice of defaulting to CoT prompting. For instruction-tuned models—the most commonly deployed variants—CoT can significantly degrade performance. Practitioners should empirically validate prompting strategies for their specific model rather than assuming CoT universally helps.

6 Limitations

Architecture scope. The CoT reversal is observed for Qwen2.5-Coder but not DeepSeek-Coder. We study two architectures; whether instruction tuning reverses CoT's effect is architecture-dependent and cannot be claimed as a universal property of instruction tuning. Broader evaluation across more architectures is needed.

Model scale. All experiments use 1.3–1.5B parameter models due to hardware constraints. Larger models may have sufficient capacity to leverage CoT regardless of training regime, potentially eliminating the reversal effect.

Probe statistical power. The style router is statistically indistinguishable from the best single fixed style in 7/8 settings—a desirable safety property—but statistically outperforms CoT in only 1/8 settings (Qwen Instruct/HumanEval). Test sets of n=82–250 problems provide limited power to detect small absolute differences (<5pp). Larger-scale evaluation would strengthen these claims.

Probe training cost. The router requires upfront labeled data: running all 12 styles on a training set to obtain ground-truth pass/fail labels. For our experiments this cost is 50% of each benchmark. Amortized across deployment, this is negligible, but it is non-zero at setup time.

Greedy decoding only. All experiments use temperature=0. Stochastic sampling may change the relative ranking of prompt styles and the probe's effectiveness.

Prompt library. Our 12-style library covers representative patterns but is not exhaustive. The probe's effectiveness depends on the quality and diversity of the style library it selects from.

7 Conclusion

The central finding of this paper is that CoT prompting's effect on code generation is not a property of CoT itself, but of the *interaction between CoT and training regime*. For Qwen2.5-Coder, the same base architecture gains +13.4% from CoT as a base model and loses -15.2% as an instruction-tuned model—both significant at $p<0.001$, replicated across three benchmarks. DeepSeek is robustly insensitive. This architecture-dependence means CoT cannot be recommended or dismissed universally for code generation.

The mechanistic picture is clear: all models detect CoT structure in early layers (Layer 1–4, >90% probe accuracy), yet this shared encoding drives opposite behavioral outcomes. Representation does not determine interpretation—training regime does.

The probe-guided router operationalizes this insight: since early activations encode prompt structure, they can also predict which style will work best. The router is statistically safe (never significantly worse than the best fixed style in 7/8 settings) and practically useful (significantly better than CoT where CoT is most harmful). It requires no prompt engineering, runs in 84ms, and can be trained once per model on any available labeled problems.

Taken together, these results argue for moving from one-size-fits-all prompting toward model-aware strategies informed by a model's own internal representations.

References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Anthropic. Simple probes can catch sleeper agents. *Anthropic Research Blog*, 2024. URL `https://www.anthropic.com/research/probes-catch-sleeper-agents`.

Jacob Austin, Augustus Odena, Maxwell Nye, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Tuan-Dung Bui, Thanh Trong Vu, Thu-Trang Nguyen, Son Nguyen, and Hieu Dinh Vo. Correctness assessment of code generated by large language models using internal representations. *arXiv preprint arXiv:2501.12934*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nelson Elhage, Tristan Hume, Catherine Olsson, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, et al. Inverse scaling in test-time compute. *Transactions on Machine Learning Research*, 2025. arXiv:2507.14417.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y.K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-Coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination-free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

William Lugoloobi, Thomas Foster, William Bankes, and Chris Russell. LLMs encode their failures: Predicting success from pre-generation activations. *arXiv preprint arXiv:2602.09924*, 2026.

Iván Vicente Moreno Cencerrado, Arnau Padrés Masdemont, Anton Gonzalvez Hawthorne, David Demitri Africa, and Lorenzo Pacchiardi. No answer needed: Predicting LLM answer accuracy from question-only linear probes. *arXiv preprint arXiv:2509.10625*, 2025.

Chris Olah, Nick Cammarata, Ludwig Schubert, et al. Zoom in: An introduction to circuits. *Distill*, 2020.

Francisco Ribeiro, Claudio Spiess, Prem Devanbu, and Sarah Nadi. On LLMs' internal representation of code correctness. In *Proceedings of the International Conference on Software Engineering (ICSE)*, 2026. arXiv:2512.07404.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Alexander Matt Turner, Lisa Thiergart, Gavin Udell, David Leike, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

Yao Wan, Wei Zhao, Hongyu Zhang, et al. What do they capture? a structural analysis of pre-trained language models for source code. *arXiv preprint arXiv:2212.10017*, 2022.

Jan Wehner et al. Representation engineering for large language models: Survey and research challenges. *arXiv preprint arXiv:2502.17601*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Jiageng Wu, Kevin Xie, Bowen Gu, Nils Krüger, Kueiyu Joshua Lin, and Jie Yang. Why chain of thought fails in clinical text understanding. *arXiv preprint arXiv:2509.21933*, 2025.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, et al. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

Yangtian Zi, Harshitha Menon, and Arjun Guha. More than a score: Probing the impact of prompt specificity on LLM code generation. *arXiv preprint arXiv:2508.03678*, 2025.

A Model Specifications

Table 4 provides full specifications for all models used in our experiments.

Table 4: **Model specifications.** All models are publicly available on HuggingFace.

Model	Params	Layers	Hidden	Heads	HuggingFace ID
Qwen2.5-Coder-1.5B	1.5B	28	1536	12	`Qwen/Qwen2.5-Coder-1.5B`
Qwen2.5-Coder-1.5B-Instruct	1.5B	28	1536	12	`Qwen/Qwen2.5-Coder-1.5B-Instruct`
DeepSeek-Coder-1.3B-Base	1.3B	24	2048	16	`deepseek-ai/deepseek-coder-1.3b-b`
DeepSeek-Coder-1.3B-Instruct	1.3B	24	2048	16	`deepseek-ai/deepseek-coder-1.3b-i`

B Prompt Style Definitions

Table 5 gives the exact prefix text prepended to each raw problem statement for all 12 prompt styles. For instruction-tuned models, the full prompt (prefix + problem) is wrapped in the model's native chat template.

Table 5: **Prompt style prefix texts.** Each prefix is prepended to the raw problem description.

Style	Prefix
direct	*(none)*
cot	"# Think step by step before implementing"
plan	"# Plan your solution before coding"
decompose	"# Break this problem into sub-problems"
expert	"# As an expert Python programmer"
reviewer	"# Write clean, production-ready code"
simple	"# Keep the solution simple and readable"
careful	"# Be careful about edge cases"
efficient	"# Use an efficient algorithm"
tdd	"# Write code that passes all test cases"
defensive	"# Handle all possible inputs gracefully"
typed	"# Use type hints throughout"

C Experimental Setup and Implementation Details

C.1 Hardware and Software

All experiments were run on a single Apple M4 Pro machine with 24 GB unified memory. We use:

- **Activation extraction:** PyTorch with MPS backend (`output_hidden_states=True`). Activations are the last-token residual stream at the specified layer.
- **Code generation:** MLX (Apple Silicon ML framework), which provides $\approx 17\times$ faster generation than PyTorch MPS via Apple's unified memory architecture.
- **Code evaluation:** Execution-based evaluation in a sandboxed subprocess per test case with a 10-second timeout.

C.2 Generation Settings

All code generation uses:

- Greedy decoding (temperature = 0, top-p = 1.0)
- Maximum 512 new tokens
- Early stopping when a complete Python code block is detected (regex: ```` ```(?:python)?\s*\n.+?\n``` ````)

Greedy decoding ensures all results are deterministic and differences across prompt styles reflect purely the prompt's effect, not sampling variance.

C.3 Layer-wise Probe Training

For each layer $\ell \in \{0, \ldots, L\}$ and each model, we train a logistic regression probe:

- Optimizer: Adam, learning rate 10^{-3}, 200 epochs, no weight decay
- Input: last-token residual stream activation (dimension d), L2-normalized
- Labels: binary (Direct = 0, CoT = 1)
- Data: all HumanEval + MBPP problems with both Direct and CoT prompts

Probe accuracy is computed on the full dataset (no held-out split, since the probe measures encoding, not generalization).

C.4 Selection Probe Training

The style routing probe is an MLP trained on 50% of each dataset, evaluated on the held-out 50%:

- Architecture: Linear(d, 128) $\rightarrow$ ReLU $\rightarrow$ Dropout(0.2) $\rightarrow$ Linear(128, 12)
- Loss: `BCEWithLogitsLoss` (multi-label: each style is an independent binary prediction)
- Optimizer: Adam, learning rate 10^{-3}, 500 epochs
- Input: last-token activation at Layer $\lfloor L/8 \rfloor$ under the *direct* prompt ($\approx$12.5% depth)
- At inference: argmax over 12 style logits selects the predicted best style

The probe layer choice ($\approx$12.5% depth) is motivated by our tomography results: probe accuracy saturates within the first 15% of network depth, and the early layers provide the richest signal for style differentiation before task-specific representations dominate.

C.5 Train/Test Split

For style routing experiments, we split each dataset 50/50 by problem index (first half = train, second half = test). The probe is trained only on training-set problems and evaluated only on test-set problems. Activation and generation caches are computed for all problems but the probe training uses only training-set labels.

D Full Per-Style Results

Table 6 reports Pass@1 for all 12 prompt styles across all 4 models and 2 primary datasets (HumanEval and MBPP), evaluated on the held-out test split (50% of each dataset).

Key observations:

- **No universal best style:** The best style varies across models. *Expert* leads for Qwen-Instruct on HumanEval; *Typed* leads for Qwen-Base; *Defensive* leads for DeepSeek-Base on MBPP.
- **CoT is not the best style** for any instruct model. For Qwen-Instruct, CoT (34.1%) is the *worst* style on HumanEval, 22pp below the best.
- **Large oracle gap:** The difference between Oracle and best single style ranges from 20–30pp, indicating substantial per-problem variability that the selection probe can partially exploit.
- **Base model style sensitivity:** Qwen-Base shows the widest style range (4.9–22.0% on HumanEval), while DeepSeek-Base is the least sensitive (2.4–4.9%).

Table 6: **Per-style Pass@1 for all models and datasets (test split).** Bold = highest per column.

	Qwen-Instruct		**Qwen-Base**		**DeepSeek-Instruct**		**DeepSeek-Base**	
Style	HE	MBPP	HE	MBPP	HE	MBPP	HE	MBPP
direct	47.6	52.4	4.9	44.8	51.2	48.0	2.4	2.0
cot	34.1	50.0	15.9	48.0	51.2	46.0	2.4	7.2
plan	48.8	51.2	11.0	47.2	53.7	47.6	3.7	8.8
decompose	43.9	50.0	8.5	46.0	50.0	46.4	2.4	6.0
expert	**56.1**	52.0	8.5	45.2	52.4	47.2	2.4	8.0
reviewer	52.4	51.6	7.3	47.6	**53.7**	47.2	3.7	8.4
simple	51.2	51.2	12.2	45.6	53.7	47.6	3.7	6.8
careful	50.0	51.2	9.8	46.4	53.7	**48.0**	2.4	8.8
efficient	50.0	51.6	8.5	46.4	52.4	47.6	2.4	8.0
tdd	47.6	52.8	14.6	47.6	50.0	47.6	2.4	7.6
defensive	51.2	**53.2**	11.0	47.2	53.7	47.2	4.9	**9.2**
typed	53.7	52.4	**22.0**	**50.0**	52.4	47.6	**4.9**	8.0
Oracle	78.0	67.2	41.5	65.6	63.4	59.6	11.0	22.4

E STATISTICAL TESTS

All significance tests use McNemar's test with continuity correction on paired binary outcomes (Pass/Fail for each problem under two prompting conditions). McNemar's test is appropriate because:

- Outcomes are binary (pass/fail per problem)
- Comparisons are paired (same problems evaluated under both conditions)
- The test is robust to class imbalance (which occurs when pass rates are very low or very high)

All reported p-values are two-tailed. We apply no multiple-comparison correction, as each row in Table 1 corresponds to an independent pre-registered comparison (Direct vs. CoT for a specific model-dataset pair).