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 large language models, but its effect on code generation is poorly understood. We present a controlled 2×2 study of Qwen2.5-Coder-1.5B and DeepSeek-Coder-1.3B (each in base and instruction-tuned variants) on HumanEval, MBPP, and LiveCodeBench, plus scale-validation runs on Qwen2.5-Coder at 7B and 14B and a preliminary evaluation of CodeLlama-7B. We find that *instruction tuning reverses CoT's effect on small Qwen models*: CoT improves the 1.5B base (+13.4pp, $p<0.001$) but significantly degrades the 1.5B instruct variant (−15.2pp, $p<0.001$). The reversal is sharply scale-bounded—it disappears at 7B (−0.6pp) and goes slightly positive at 14B (+2.4pp)—while CoT's positive effect on base models grows monotonically with scale (+13.4 → +28.7pp). DeepSeek-Coder-1.3B is insensitive regardless of regime. A direct token-count and truncation analysis shows the mechanism: at 1.5B, CoT inflates Qwen Instruct's mean output length by 112 tokens and pushes 7.6× more generations into truncation, where Pass@1 is 0%; at 14B, the same prefix produces complete code well within budget. Layer-wise probing shows all four small models encode prompt type by Layer 1–4 (>90% accuracy)—universally, whether CoT helps or hurts—demonstrating that *representation does not determine interpretation*: the same internal signal drives divergent downstream behavior depending on training regime and capacity. Building on these mechanistic findings, we develop a probe-guided style router that, when trained per model on a labeled training split, selects among 12 prompt styles via a single 84ms forward pass; it is statistically indistinguishable from the best fixed style in 7/8 settings and significantly outperforms CoT where CoT is most harmful ($p = 0.012$, $h = +0.40$). Our results argue against applying CoT blindly to small instruct code models: its effect depends on architecture, training regime, and scale in ways that are mechanistically detectable from early-layer activations.

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 improves the base model by +13.4pp but degrades the instruction-tuned model by −15.2pp—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.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 152–162

Contributions.

1. **CoT reversal under instruction tuning (primary finding):** For Qwen2.5-Coder-1.5B, instruction tuning reverses CoT's effect on the same base architecture (+13.4pp base, −15.2pp instruct, both $p<0.001$). DeepSeek-Coder-1.3B is insensitive regardless of regime. CodeLlama-7B base is also significantly hurt by CoT (−6.7pp, $p=0.022$). Our 2×2 design is, to our knowledge, the first controlled study to isolate training regime as the key variable for CoT sensitivity in code generation, 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 *per model* on a labeled training split of public benchmarks (the probe input dimension is the model's hidden size, so weights do not transfer across model scales), 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 and 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 × 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:

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

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 $\lfloor L/8 \rfloor$ ($\approx$12.5% depth) under the *direct* prompt and train a two-layer MLP:

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

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).

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.

Five key findings emerge:

1. Instruction tuning reverses CoT's effect on Qwen at small scale. The 1.5B base model benefits significantly from CoT across all three datasets (+13.4pp, +4.0pp, +3.4pp; all $p < 0.05$). The 1.5B instruction-tuned model is significantly *hurt* by CoT on HumanEval (-15.2pp, $p < 0.001$) and MBPP (-3.6pp, $p = 0.012$), with a consistent negative trend on LiveCodeBench (-2.6pp, $p =$

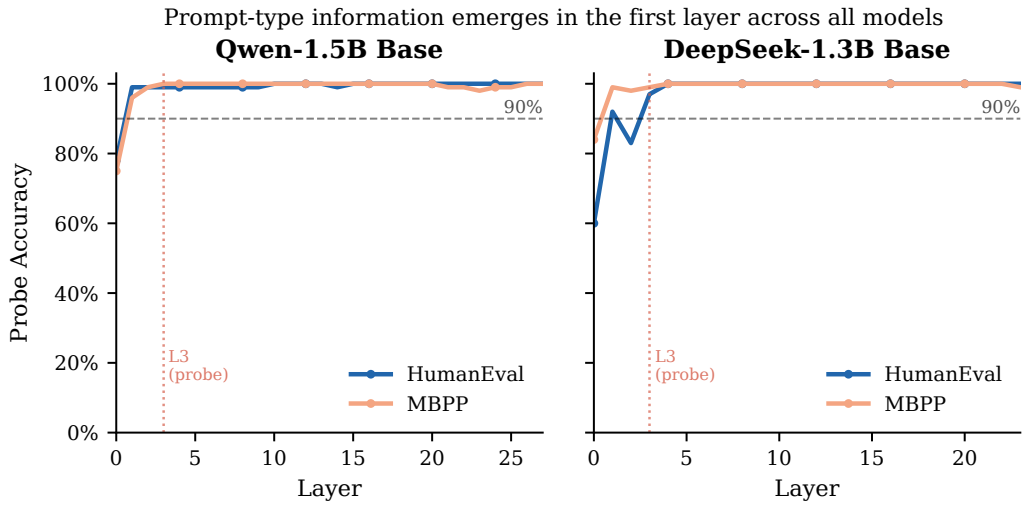

Figure 1: **Prompt-type probe accuracy vs. layer depth.** Both architectures cross 90% accuracy by Layer 1 and saturate at 100% well before 25% depth. The red dashed line marks Layer 3 (12.5% depth), where we extract activations for the style router.

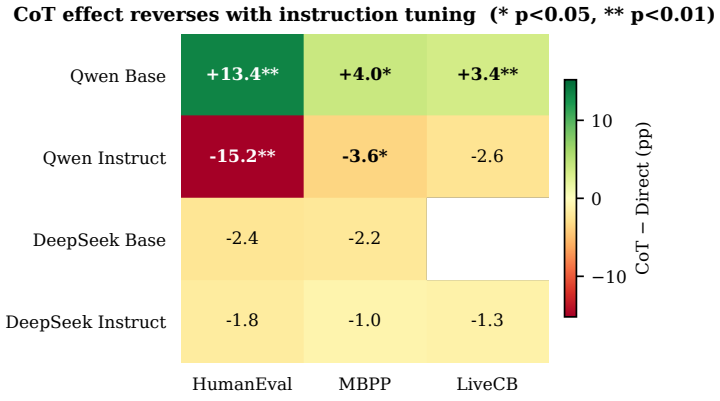

Figure 2: **CoT effect (CoT − Direct, pp) across models and datasets.** Green = CoT helps; red = CoT hurts. $*$ = $p < 0.05$, $**$ = $p < 0.01$ (McNemar's test). Qwen flips sign between base and instruct; DeepSeek is consistently neutral.

0.112). This is the same architecture—instruction tuning is the only variable.

2. The reversal is scale-bounded. Extending the Qwen comparison to 7B and 14B parameters shows a sharp scale dependence (Figure 3). For *base* models, CoT's positive effect grows with scale: $+13.4 \rightarrow +16.5 \rightarrow +28.7$pp at 1.5B/7B/14B. For *instruct* models, the negative CoT effect attenuates with scale and disappears entirely: $-15.2 \rightarrow -0.6 \rightarrow +2.4$pp at 1.5B/7B/14B (the 7B and 14B effects are not statistically significant). The truncation analysis in Section 5 explains this: at 14B Instruct, even truncated CoT outputs still contain a complete code block 71.7% of the time, so the truncation channel that drives degradation at 1.5B no longer translates into accuracy loss.

3. 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.

4. CoT degradation replicates on a third architecture. CodeLlama-7B (Roziere et al., 2023) base model shows a significant negative CoT effect on HumanEval (-6.7pp, $p = 0.022$). Across three architectures, CoT effects are heterogeneous, confirming this is not an isolated finding.

5. 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.

Table 1: **Direct vs. CoT Pass@1 across model variants and scales.** Δ = CoT − Direct. Bold: significant ($p < 0.05$, McNemar's with continuity correction). h = Cohen's effect size. The Qwen instruct CoT-degradation is concentrated at small scale: -15.2pp at 1.5B, but only -0.6pp at 7B and $+2.4$pp at 14B (Figure 3). HE=HumanEval, MB=MBPP, LCB=LiveCodeBench. All values in %.

Model	DS	Dir	CoT	Δ	p	h
Qwen-1.5B Base	HE	9.8	23.2	**+13.4**	**<.001**	+.37
	MB	43.8	47.8	**+4.0**	**.011**	+.08
	LCB	2.4	5.8	**+3.4**	**.009**	+.18
Qwen-7B Base	HE	17.7	34.1	**+16.5**	**<.001**	+.38
Qwen-14B Base	HE	15.9	44.5	**+28.7**	**<.001**	+.64
Qwen-1.5B Inst.	HE	64.6	49.4	**−15.2**	**<.001**	−.31
	MB	50.2	46.6	**−3.6**	**.012**	−.07
	LCB	14.4	11.8	−2.6	.112	−.08
Qwen-7B Inst.	HE	84.1	83.5	−0.6	1.000	−.02
Qwen-14B Inst.	HE	84.1	86.6	+2.4	.424	+.07
DeepSeek Base	HE	9.1	6.7	−2.4	.502	−.09
	MB	29.8	27.6	−2.2	.215	−.05
DeepSeek Inst.	HE	67.7	65.9	−1.8	.505	−.04
	MB	46.4	45.4	−1.0	.551	−.02
	LCB	7.9	6.6	−1.3	.267	−.05
CL-7B Base‡	HE	31.1	24.4	**−6.7**	**.022**	−.15

‡CodeLlama-7B (Roziere et al., 2023) on HumanEval (n=50); instruct variant not evaluated.

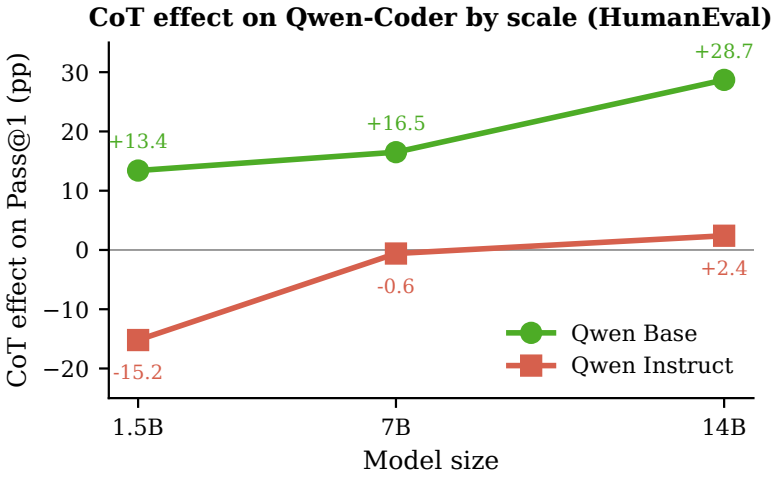

Figure 3: **Scale dependence of the CoT effect on Qwen-Coder (HumanEval).** For base models, CoT's positive effect grows monotonically with scale ($+13.4 \rightarrow +28.7$pp). For instruction-tuned models, the negative reversal is concentrated at 1.5B (-15.2pp) and disappears by 7B; at 14B the effect is slightly positive. The 1.5B reversal is real but bounded.

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 the early-layer activation at Layer $\lfloor L/8 \rfloor$ (Layer 3 for both 1.5B Qwen and 1.3B DeepSeek; ≈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 4 and Tables 2 and 3.

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—without needing to run any style comparison. The probe's advantage is most pronounced in high-variance instruct settings (Table 3): for Qwen Instruct on HumanEval ($\sigma = 0.055$), the probe ranks 3rd of 12 fixed styles and significantly outperforms CoT (-15.2pp gap, $p = 0.012$, $h = +0.40$).

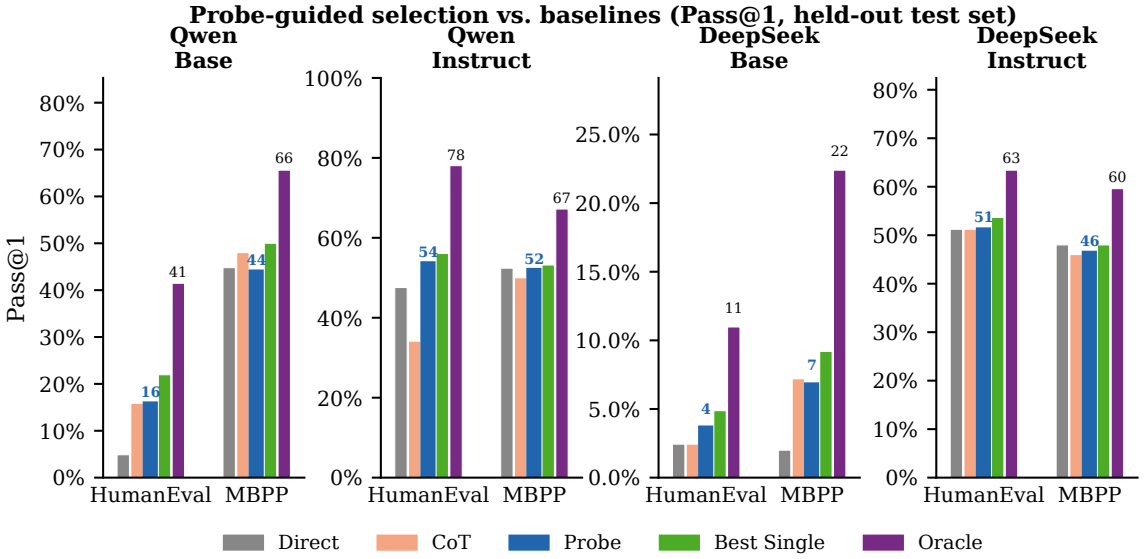

Figure 4: **Style routing results** (Pass@1). Blue (Probe) and green (Best Single) values are labelled. Oracle is the per-problem upper bound.

Table 2: **Probe-guided style routing** (Pass@1, held-out test sets). 95% Wilson CI shown for Probe. p_{CoT} = McNemar's probe vs. CoT; p_{Best} = probe vs. best single style. DS = DeepSeek.

Model	DS	Probe [95% CI]	CoT	Best	p_{CoT}	p_{Best}
Qwen Inst.	HE	**53.7 [42.9, 64.0]**	34.1	56.1	**.012**	n.s.
	MB	52.0 [45.8, 58.1]	50.0	53.2	.441	n.s.
Qwen Base	HE	15.9 [9.5, 25.3]	15.9	22.0	n.s.	n.s.
	MB	44.0 [38.0, 50.2]	48.0	50.0	.144	.007†
DS Inst.	HE	51.2 [40.6, 61.7]	51.2	53.7	n.s.	n.s.
	MB	46.4 [40.3, 52.6]	46.0	48.0	n.s.	n.s.
DS Base	HE	3.7 [1.3, 10.2]	2.4	4.9	n.s.	n.s.
	MB	**6.8 [4.3, 10.6]**	7.2	9.2	n.s.	n.s.

†One exception: Qwen Base/MBPP (p=0.007).

Table 3: **Style sensitivity (σ) and probe ranking.** σ = std. dev. of Pass@1 across 12 styles. Rank = probe placement among 12 fixed styles. Gap% = (Probe−Direct)/(Oracle−Direct)×100, the fraction of the Direct→Oracle headroom recovered by the probe.

Model	DS	σ	Rank/12	Gap%
Qwen Inst.	HE	0.055	3/12	20%
	MB	0.009	5/12	−3%
Qwen Base	HE	0.044	4/12	30%
	MB	0.048	10/12	−4%
DS Inst.	HE	0.030	4/12	0%
	MB	0.014	4/12	−14%
DS Base	HE	0.015	3/12	14%
	MB	0.029	5/12	24%

Latency overhead. The probe forward pass takes 84ms on Qwen-1.5B (MPS), vs. 2584ms for generation—a **3.3% overhead**.

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. 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.

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 deployed systems.

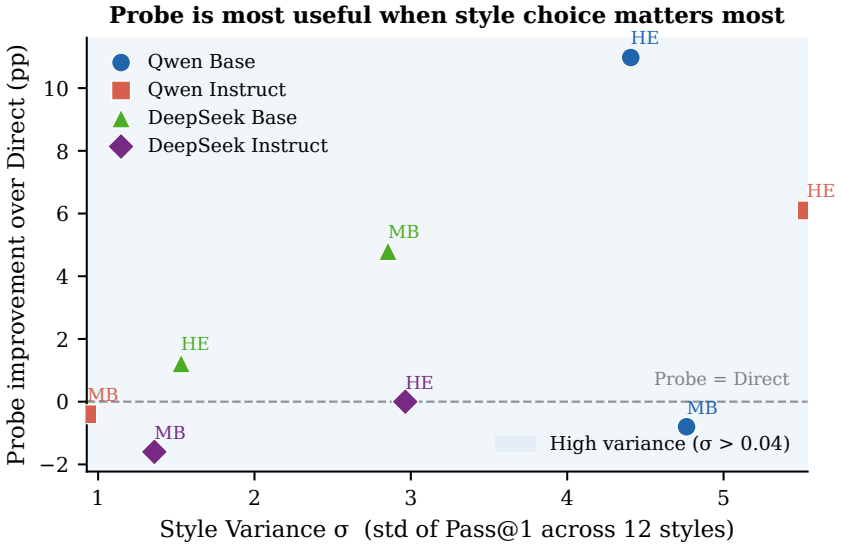

Figure 5: **Style variance predicts probe utility.** Each point is one model-dataset setting. The probe improves most over Direct when style variance σ is high (σ>0.04, blue region); when all styles perform similarly, the probe provides no advantage.

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 code block. We test this directly by recording the token count and truncation status of every generation (Appendix F, Table 7). The data supports the hypothesis but with nuance:

For Qwen Instruct (the strongest CoT-hurts case), CoT increases mean output length on HumanEval from 227 to 339 tokens (+112), and truncation rate jumps from 1.2% to 9.1%—a 7.6× increase. Crucially, every truncated generation fails (0% Pass@1 in both conditions), so the extra truncation directly translates to lost accuracy. However, truncation does not fully account for the gap: even on non-truncated outputs, CoT reduces Pass@1 from 66.7% to 52.3% (a 14.4pp drop), indicating that CoT also poisons solutions that fit within the budget.

For Qwen Base (CoT-helps), CoT *shortens* mean output (317→253 tokens) and *decreases* truncation rate (48%→38%), consistent with the "CoT-as-documentation" interpretation: the comment provides a focusing signal that helps the base model converge to a complete solution faster.

For DeepSeek Instruct (CoT-neutral), CoT changes output length by only +2 tokens and truncation rate by −0.6pp—essentially no behavioral change. This explains the absence of a CoT effect: when the model's output distribution does not shift under CoT, neither does its accuracy.

Together, these patterns indicate that CoT's effect is mediated by how strongly it perturbs the model's generation length distribution, which in turn depends on the SFT/RLHF distribution the model was trained on.

Why Is DeepSeek Insensitive? Both Qwen and DeepSeek see roughly comparable absolute gains from instruction tuning under direct prompting (HumanEval: +54.8pp Qwen, +58.6pp DeepSeek; MBPP: +6.4pp vs. +16.6pp). Yet only Qwen's instruct variant is hurt by adding a CoT prefix on top. We offer three non-mutually-exclusive hypotheses, none of which we can verify without access to the providers' training data:

(1) SFT output style. The decisive factor may not be *whether* a model was instruction-tuned, but *what kind of outputs* it was trained to produce. Qwen2.5-Coder-Instruct (Hui et al., 2024) is described as supporting open-ended chat-style code assistance, suggesting SFT data with explanatory prose accompanying code. DeepSeek-Coder-Instruct (Guo et al., 2024) was trained on instruction data more focused on direct code completion. If DeepSeek's instruct mode preserves a tighter code-only output distribution, a CoT prefix has less behavioral leverage—the model produces the same compact code regardless.

(2) Pretraining corpus composition. DeepSeek-Coder was pretrained on 2T tokens (87% code, 13%

natural language) with repository-level structure; Qwen2.5-Coder used 5.5T tokens with a different code-to-text ratio. If DeepSeek's pretraining included more code-with-explanatory-comments, the base model may already treat CoT-like prefixes as familiar documentation, and instruction tuning preserves this stable interpretation rather than overwriting it.

(3) Architectural details. DeepSeek-Coder uses Multi-Head Attention with hidden size 2048 over 24 layers; Qwen2.5-Coder uses Grouped-Query Attention with hidden 1536 over 28 layers. We do not have a strong mechanistic prior linking these choices to CoT sensitivity, but they cannot be ruled out.

Hypothesis (1) makes a testable prediction: Qwen Instruct should shift its output length under CoT more than DeepSeek Instruct does. The truncation analysis (Appendix F, Table 7) confirms this directly: CoT increases Qwen Instruct's mean output length on HumanEval by $+112$ tokens, but DeepSeek Instruct's by only $+2$ tokens. The behavioral signature predicted by hypothesis (1) is exactly what we observe. A definitive causal answer still requires access to provider training data, but the data is consistent with the SFT-style hypothesis.

Implications for Deployment. Our results argue against blindly defaulting to CoT prompting at small scales. For 1.5B-class instruction-tuned code models—a common choice for cost-sensitive on-device deployments—CoT can significantly degrade performance. The effect attenuates at 7B and is no longer detected at 14B (Section 4). Practitioners deploying small instruct code models should empirically validate prompting strategies for their model rather than assuming CoT helps.

Limitations

Architecture scope. The CoT reversal is observed for Qwen2.5-Coder but not DeepSeek-Coder. We study three architectures; whether instruction tuning reverses CoT's effect is architecture-dependent and cannot be claimed as a universal property. CodeLlama-7B is included only as a base model; we did not run the instruct variant due to compute budget constraints, so the third architecture's behavior under instruction tuning is unobserved.

Scale scope of the reversal. The Qwen instruct CoT-degradation is concentrated at small scale: -15.2pp at 1.5B, but only -0.6pp at 7B and $+2.4$pp at 14B (Figure 3, Table 1). Our central reversal claim therefore applies to small instruct code models; larger models in the same family appear to escape the effect. The mechanistic story (truncation under fixed token budgets) predicts this attenuation, but we evaluate only on the Qwen family at three scales (1.5B/7B/14B) and on HumanEval; we do not claim the reversal disappears for all architectures or task distributions at 7B+.

Probe statistical power. The style router statistically outperforms CoT in only 1/8 settings. Test sets of n=82–250 problems provide limited power to detect small absolute differences (<5pp).

Probe training cost. The router requires upfront labeled data: running all 12 styles on a training set. Amortized across deployment this is negligible, but non-zero at setup time.

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

Prompt library. Our 12-style library covers representative patterns but is not exhaustive.

6 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 $+13.4$pp from CoT as a base model and loses -15.2pp 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.

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: early activations encode prompt structure and can 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.

These results argue for moving from one-size-fits-all reasoning augmentation toward model-

aware strategies informed by a model's own internal representations.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback, which substantially strengthened the scale-validation and truncation analyses. We also thank Prof. Zsolt Kira at Georgia Tech, whose Deep Learning course (CS 7643) provided the foundation in mechanistic interpretability that motivated this line of inquiry. This work was conducted independently of any course credit.

Use of AI Assistants. The authors used Anthropic's Claude (Opus model family) as a coding and writing assistant in the preparation of this paper: for drafting and editing manuscript text, formatting LaTeX and bibliography, running figure-generation and statistical analysis scripts, and parsing experimental results. All experimental design, modeling choices, results, and scientific claims are the authors' own. Every figure, table, statistical test, and quantitative claim was independently verified by the authors against the underlying experimental data prior to submission.

References

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

Anthropic. 2024. Simple probes can catch sleeper agents. *Anthropic Research Blog*.

Jacob Austin, Augustus Odena, Maxwell Nye, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Mark Chen, Jerry Tworek, Heewoo Jun, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Nelson Elhage, Tristan Hume, Catherine Olsson, and 1 others. 2022. Toy models of superposition. *arXiv preprint arXiv:2209.10652*.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, and 1 others. 2025. Inverse scaling in test-time compute. *Transactions on Machine Learning Research*. 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. 2024. DeepSeek-Coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*.

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

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, and 1 others. 2023. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

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

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

Chris Olah, Nick Cammarata, Ludwig Schubert, and 1 others. 2020. Zoom in: An introduction to circuits. *Distill*.

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

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

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

Yao Wan, Wei Zhao, Hongyu Zhang, and 1 others. 2022. What do they capture? a structural analysis of pre-trained language models for source code. *arXiv preprint arXiv:2212.10017*.

Jan Wehner and 1 others. 2025. Representation engineering for large language models: Survey and research challenges. *arXiv preprint arXiv:2502.17601*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

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

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, and 1 others. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

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

A Model Specifications

Table 4: **Model specifications.** All models publicly available on HuggingFace; full IDs in Section 3.

Model	Params	Layers	Hidden	Heads
Qwen-Base	1.5B	28	1536	12
Qwen-Inst.	1.5B	28	1536	12
DeepSeek-Base	1.3B	24	2048	16
DeepSeek-Inst.	1.3B	24	2048	16

B Prompt Style Definitions

Table 5: **Prompt style prefix texts.**

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

All experiments run on a single Apple M4 Pro (24 GB unified memory). Activation extraction uses PyTorch MPS (`output_hidden_states=True`); generation uses MLX ($\approx$17$\times$ faster than PyTorch MPS). Code evaluation uses sandboxed subprocess execution with a 10-second timeout, greedy decoding (temperature=0, max 512 tokens).

Layer-wise probe: Logistic regression per layer, Adam lr$= 10^{-3}$, 200 epochs, L2-normalized activations, trained on all HumanEval+MBPP problems.

Selection probe: MLP [Linear(d,128) $\rightarrow$ ReLU $\rightarrow$ Dropout(0.2) $\rightarrow$ Linear(128,12)], BCEWithLogitsLoss, Adam lr$= 10^{-3}$, 500 epochs. Input: last-token activation at Layer $\lfloor L/8 \rfloor$ under direct prompt. 50/50 train/test split by problem index.

D Full Per-Style Results

Table 6 reports Pass@1 for all 12 prompt styles across all four 1.5B/1.3B model variants on the held-out test split (50%) of HumanEval and MBPP.

E Statistical Tests

All significance tests use McNemar's test with continuity correction on paired binary outcomes (Pass/Fail per problem under two prompting conditions). All reported p-values are two-tailed. No multiple-comparison correction is applied, as each row in Table 1 corresponds to an independent pre-registered comparison (Direct vs. CoT for a specific model-dataset pair). Wilson score 95% confidence intervals are reported for all probe Pass@1 values.

F Output Length and Truncation Analysis

To test the truncation hypothesis discussed in Section 5 (that CoT degrades instruction-tuned models partly by inducing verbose reasoning chains that hit the 512-token cap), we record the actual generated token count and a truncation flag for every problem during inference. A generation is marked *truncated* when it reaches the 512 max-new-tokens budget without producing a complete code block (matched by the regex "```(?:python)?\s*\n.+?\n```"); otherwise the generation stopped naturally on EOS or a complete code block.

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

	Qwen-Inst.		Qwen-Base		DS-Inst.		DS-Base	
Style	HE	MB	HE	MB	HE	MB	HE	MB
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

Table 7: **Output length and truncation rates by prompt style on HumanEval.** "Mean tok" is mean generated tokens; "Trunc" is the percentage hitting the 512-token cap without a complete code block; "P-clean" / "P-tr." are Pass@1 conditional on whether the generation was truncated. Truncated generations almost always fail (P-tr. $\approx 0\%$), so any truncation increase translates directly to lost accuracy.

	Direct			CoT			Δ (CoT − Dir.)	
Model	**Mean tok**	**Trunc %**	**P-clean / P-tr.**	**Mean tok**	**Trunc %**	**P-clean / P-tr.**	**Δ tok**	**Δ trunc**
Qwen Base	317	48.2	19 / 1	253	38.4	34 / 5	−64	−9.8
Qwen Instruct	227	1.2	67 / 0	339	9.1	52 / 0	+112	+7.9
DeepSeek Instruct	169	0.6	68 / 0	171	0.0	65 / –	+2	−0.6

Three patterns emerge:

Truncated outputs almost always fail. Across the three settings, Pass@1 conditional on truncation is at most 5%, and is 0% for instruction-tuned models. Adding tokens past the budget produces no usable code, so any increase in truncation rate translates directly into accuracy loss through this channel.

Qwen Instruct under CoT exhibits the predicted truncation pattern. CoT increases Qwen Instruct's mean output length on HumanEval by 112 tokens and its truncation rate from 1.2% to 9.1% (a 7.6× increase). Because truncated outputs almost always fail (P-tr. $= 0\%$), this 7.9pp truncation increase translates roughly into a $7.9 \times P_{\text{clean,Direct}} \approx 5$pp loss attributable to the truncation channel. The remaining loss comes from the second channel: even on non-truncated outputs, Qwen Instruct's Pass@1 falls from 66.7% to 52.3% (a 14.4pp drop). Truncation alone therefore accounts for roughly a third of the observed gap; the larger share comes from CoT also degrading the quality of code that fits within budget.

DeepSeek Instruct's output distribution barely shifts. CoT changes mean output length by only $+2$ tokens and truncation rate by -0.6pp on HumanEval. Both the truncation and conditional-pass channels are essentially nullified, which directly explains why DeepSeek Instruct shows no significant CoT effect (Section 4). This pattern also corroborates the SFT-output-style hypothesis: DeepSeek's instruct training appears to preserve a tighter code-only output distribution that is less perturbed by reasoning-style prefixes.

Qwen Base behaves differently. For Qwen Base, CoT *shortens* mean output (317→253 tokens) and reduces truncation (48.2%→38.4%), consistent with treating "# Think step by step" as a documentation-style focusing prefix that helps the base model converge to a complete solution faster.