PROBING SYSTEM-LEVEL PERFORMANCE REPRESENTATIONS IN CODE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Can a code language model represent a system-level property more reliably than it reports that property in its output? We study capacity bottlenecks in rendered request handlers, where the correct stage is derived exactly from visit counts, per-visit costs, and the runtime sharing regime. Across six Qwen2.5-Coder scales from 0.5B to 32B on 1,200 instances, linear probes predict the bottleneck with 67.9–87.1% held-out accuracy, while surface-feature and shuffled-label controls remain near chance. A teacher-forced output read-out on the same held-out instances reaches 22.1–45.0%, leaving a 37.5–58.8 point gap. At 0.5B and 1.5B, the rank of the correct stage under the output logits is indistinguishable from chance; from 3B upward, the rank is significantly better than chance, indicating that the output distribution carries some bottleneck information even though it remains substantially less accurate than the probe. Layerwise profiles distinguish lexically explicit properties, which become decodable early, from composed properties, which emerge later; bottleneck decodability peaks around mid-depth and then decreases. The probe result also holds for DeepSeek-Coder 1.3B and StarCoder2 3B. Finally, steering with fitted probe directions is unstable across data splits: refitting changes the selected layer and reverses intervention effects. These results separate linear decodability, output accessibility, and intervention stability when probing system-level reasoning in code models.

1 INTRODUCTION

Code language models are increasingly used for programming assistance and repository-level reasoning, but many practically important software properties are not local attributes of a token or expression. A model that recommends changes to a service may need to infer how several resources interact at runtime, and a correct internal representation is useful only insofar as it can be read reliably or expressed in the model's output.

We study one such property: which stage of a request handler becomes the capacity bottleneck as request rate grows. The answer is determined by visit counts, per-visit costs, and whether stages share a serialized runtime resource. All ingredients are present in the rendered handler, but the bottleneck itself is not stated explicitly and requires composing those ingredients. This makes the task suitable for asking three distinct questions: whether the property is linearly decodable from hidden states, when in the network it becomes decodable, and how that representation relates to the model's own answer distribution.

The task has two useful methodological features. First, its labels are derived from a capacity law rather than human annotation, so correctness is unambiguous on the generated instances. Second, the same generator yields companion properties with different compositional demands. In our data, selecting the resource with the largest per-visit cost is correct on only 27.9% of instances, providing a direct check that a successful probe is not merely reading a salient literal.

Contributions.

- **A controlled system-level probing task.** We generate 1200 rendered service handlers with exact labels for the bottleneck, runtime sharing regime, unindexed-query presence, and pool-binding status, together with explicit surface-feature and shuffled-label controls.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- **A representation and layerwise analysis across scales and families.** Bottleneck probes reach 67.9–87.1% held-out accuracy across six Qwen2.5-Coder scales, substantially above the controls. Lexically explicit properties become decodable in early layers, whereas composed properties emerge later; bottleneck decodability peaks near the middle of the network. The bottleneck result replicates on DeepSeek-Coder 1.3B and StarCoder2 3B.
- **A direct comparison between internal decodability and the model's output readout.** Teacher-forced candidate logits remain substantially less accurate than the best linear probe. At the two smallest scales the correct candidate's rank is statistically indistinguishable from chance, while from 3B upward the ranking becomes informative but still does not close the gap to the hidden-state probe.
- **A split-sensitivity test for probe-based steering.** Refitting the probe on different train/validation partitions changes the selected layer and the fitted direction, and the corresponding intervention effects reverse across runs. Probe accuracy alone therefore does not establish that a particular fitted direction is a stable intervention target.

2 Related work

Probing and probe controls. Linear probes are a standard tool for characterizing intermediate representations (Alain & Bengio, 2017). Their interpretation depends on appropriate controls: Hewitt & Liang (2019) emphasize comparing against tasks that remove the relevant structure, while subsequent work has proposed alternatives based on description length (Voita & Titov, 2020) and interventions that remove decodable information (Elazar et al., 2021). We use two controls that target different failure modes: a logistic model on surface features of the rendered code, and the same hidden-state probe trained on shuffled labels. Our steering analysis then asks a separate question about whether a fitted probe direction provides a stable intervention target.

Probing code models. Recent work has used hidden states to study code-model behavior and program properties. Di Cicco (2026) probe whether code that has not yet been generated will pass its tests, while Gorgone & Carcassi (2026) recover type information and Sandoval et al. (2026) localize security-relevant computation. Sahoo et al. (2026) show that apparently strong probe results can disappear after controlling for formatting and source-identity confounds, and they caution against interpreting decodability as a causal mechanism without additional tests. Our setting differs in that the target is an externally defined property of the rendered program rather than a property of the model's own future output, and the generator keeps format and instance distribution fixed across targets.

Internal representations and model outputs. Several lines of work show that information available in internal representations need not be reflected faithfully in generated outputs. Examples include latent-knowledge recovery (Burns et al., 2023), truth-related directions in hidden states (Azaria & Mitchell, 2023; Marks & Tegmark, 2024), and world-state representations in sequence models (Li et al., 2023a; Nanda et al., 2023; Gurnee & Tegmark, 2024). Separately, generated rationales and self-evaluations can fail to track the computation or quality of the underlying answer (Turpin et al., 2023; Lanham et al., 2023; Saunders et al., 2022; Huang et al., 2024; Stechly et al., 2024). We compare the two levels on identical inputs by pairing a hidden-state probe with a teacher-forced ranking of the same candidate answers.

Causal methods and steering. Activation patching and related interventions test causal relevance beyond decodability (Meng et al., 2022; Ghandeharioun et al., 2024). Activation addition (Turner et al., 2023) and inference-time intervention (Li et al., 2023b) show that directions obtained from contrasts or probes can alter model behavior. Our steering experiment is narrower: it asks whether the particular direction returned by a probe fit is stable enough across data partitions to support a consistent intervention conclusion.

System performance and architectural reasoning. The bottleneck and sharing regime are classical objects in operational analysis (Denning & Buzen, 1978; Lazowska et al., 1984; Little, 1961), and modern systems work learns related performance quantities from telemetry (Venkataraman et al.,

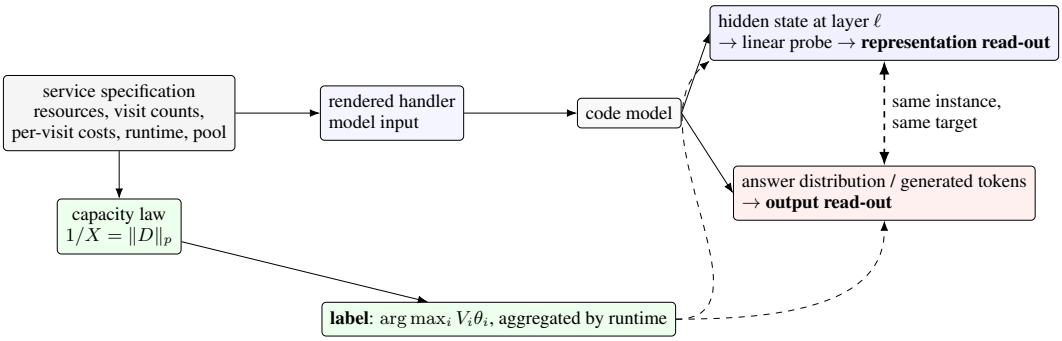

Figure 1: The representation and output read-outs are evaluated on the same generated service and against the same analytically derived bottleneck label. This separates whether the target is linearly decodable from hidden states from whether it is accessible through the model's answer distribution.

2016; Gan et al., 2019; Zhang et al., 2021; Park et al., 2021; Somashekar et al., 2024). Recent software benchmarks also evaluate model judgments about architecture and repository structure (Santilli et al., 2026; Adnan et al., 2026; Yang et al., 2026; Dinu et al., 2026). Our task uses the systems theory as a label generator, allowing representation-level and output-level measurements to be compared against the same exact target.

3 TASK

Instances. Each instance describes a service with three to five resources. Every resource has a type, visit count, and per-visit cost; the specification also states whether the runtime serializes stages through one interpreter and gives an admission-pool size. We render the specification as a Python request handler with helper functions, pool declarations, and per-stage cost comments.

Targets. We derive four properties from each instance: the bottleneck stage, whether the runtime serializes its stages, whether an unindexed aggregate is present, and whether the admission pool binds. The first requires computing per-resource demands and aggregating them according to the sharing regime. The last requires combining response time with the pool size. The two remaining properties are stated directly in the rendered code and serve as lexical calibration targets.

Controls. The surface-feature control uses the number of resources; the largest per-visit cost and its index; the largest visit count and its index; and the character and line counts of the rendered code. The shuffled-label control retrains the hidden-state probe at the selected layer after permuting labels. On the full instance distribution, the bottleneck matches the largest per-visit cost on 27.9% of examples and the majority class is 28.8%.

4 METHOD

For each model, we record the hidden state at the final token of a fixed prompt that presents the handler and asks the model to consider behavior as request rate grows. We retain every second layer, including the embedding output. A multinomial or binary logistic probe with ℓ_2 regularization is trained on 60% of the instances. The read-out layer is selected by accuracy on a 20% validation split, and all reported probe accuracies use the remaining 20% test split. The controls use the same partition. We additionally record the best test-layer accuracy only as a diagnostic for the effect of improper test-time layer selection.

For the output-level comparison, we ask the model for the bottleneck stage using a fixed answer format. To separate answer formatting from candidate preference, we also teacher-force the requested answer prefix and rank the valid stage numbers under the next-token distribution. This yields a forced-choice accuracy and a normalized rank for the gold stage on the same held-out instances used by the probe.

scale	bottleneck probe (layer)	pool binds probe (layer)	is serialized probe (layer)	has unindexed probe (layer)
0.5B	67.9 (L16)	86.2 (L24)	100.0 (L6)	100.0 (L2)
1.5B	80.8 (L14)	95.0 (L18)	100.0 (L2)	100.0 (L2)
3B	77.9 (L20)	99.2 (L14)	100.0 (L2)	100.0 (L2)
7B	86.2 (L20)	98.3 (L14)	100.0 (L2)	100.0 (L2)
14B	85.8 (L26)	99.2 (L30)	100.0 (L2)	100.0 (L2)
32B	87.1 (L38)	100.0 (L40)	100.0 (L4)	100.0 (L2)
majority	30.4	59.2	66.2	76.2
surface features	24.6	58.8	79.6	76.7
shuffled labels	20.0–26.7	45.0–52.5	56.7–63.7	73.3–76.7

Table 1: Held-out probe accuracy (%) across six Qwen2.5-Coder scales. Layers are selected on the validation split and evaluated on a 240-instance test split; binomial 95% intervals for the bottleneck probe are approximately ± 4.2 to ± 5.9 points. The two explicit targets are recovered in early layers, while the composed targets are best read later. Control rows use the same test split.

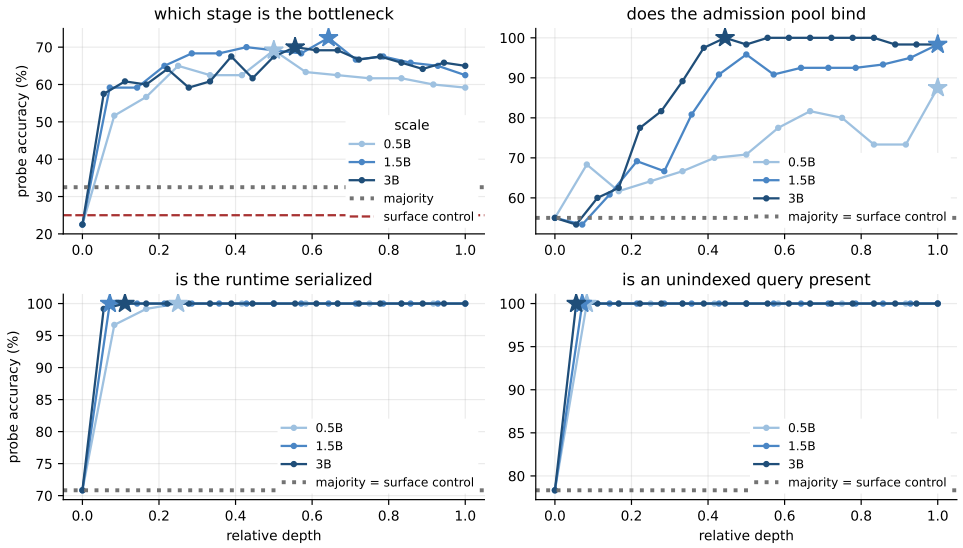

Figure 2: Probe accuracy as a function of relative depth for the four targets at the three scales used for the layer sweep. Dotted lines show the majority baseline; dashed lines show the surface-feature control where it differs from the majority. Lexically explicit targets reach ceiling performance early. The pool-binding target generally improves with depth, while bottleneck decodability peaks near mid-depth and then decreases toward the final layer.

5 Results

The bottleneck is linearly decodable well above both controls across all six scales. The sharing regime and unindexed-query indicator are recovered essentially perfectly, as expected for properties stated directly in the rendered input. The pool-binding target is also highly decodable but develops later in the network. Together, these targets provide a progression from lexical to composed properties under a common renderer and split.

Two instance sets are used in the analyses below. The six-scale probe and output-read-out results use all 1200 instances. The layer-profile, tolerance, and steering analyses were run on a 600-instance subset generated earlier from the same distribution; the bottleneck class proportions differ from the full set by at most 2.3 percentage points. We state the relevant sample size with each analysis.

5.1 LAYER PROFILES DISTINGUISH LEXICAL AND COMPOSED PROPERTIES

The four targets exhibit different layerwise profiles. The unindexed-query and sharing targets reach 100% accuracy within the early part of each network. Pool binding, which requires combining a response-time quantity with the pool size, generally improves with depth: its peak occurs at relative depths 1.00, 1.00, and 0.44 for the three scales, with final-layer decreases of 0.0, 0.0, and 1.7 percentage points. Bottleneck decodability peaks at relative depths 0.50, 0.64, and 0.56 and then falls by 14.5%, 13.8%, and 7.1% of the peak value over the remaining layers.

This profile is consistent with a distinction between properties that can be read directly from local input cues and properties that require composition across the handler. For the bottleneck in particular, the best validation-selected read-out is consistently in an intermediate layer rather than the final layer. The profile alone does not establish which internal computation produces the property, but it identifies where a linear read-out is most informative.

5.2 SURFACE AND LABEL CONTROLS RULE OUT SIMPLE SHORTCUTS

A probe is informative only relative to plausible alternatives. Table 2 summarizes the controls for the bottleneck target.

shortcut	accuracy	interpretation
majority class	30.4%	held-out class baseline
largest per-visit cost	27.9%	ignores visit count and sharing
all surface features jointly	24.6%	linear surface control
same probe, shuffled labels	20.0–26.7%	probe-capacity control
linear probe on hidden states	**67.9–87.1%**	

Table 2: Controls for bottleneck prediction. The target depends on a product of visit count and per-visit cost, followed by aggregation determined by the runtime sharing regime. Neither the strongest single literal nor the joint linear surface-feature model approaches the hidden-state probe.

The joint surface model reaches 24.6% on the held-out split, and shuffled-label probes range from 20.0% to 26.7% across scales. Both are far below the hidden-state probe. These controls do not rule out every possible correlate of the generated code, but they exclude the explicit surface statistics used to construct the most direct shortcut baselines.

An analytic task margin predicts probe success. The capacity law supplies a natural difficulty variable: the gap between the largest and second-largest relative demand. Pooled over the three scales used for this analysis, probe accuracy is higher in the upper two quartiles of this gap, and a logistic regression retains a positive gap coefficient after controlling for the number of resources and scale. Appendix A reports the regression, per-scale correlations, and the corresponding figure. The effect is modest and not monotone within every quartile, so we use it as a task-difficulty correlate rather than as a calibration guarantee.

5.3 THE OUTPUT READ-OUT REMAINS BELOW THE HIDDEN-STATE PROBE

The hidden-state probe and the teacher-forced output read-out answer the same bottleneck question on the same held-out instances. The forced-choice read-out removes free-form generation and formatting: after teacher-forcing the requested answer prefix, we compare the logits of the valid candidate stage numbers. We also measure the normalized rank of the gold stage, which retains information about relative preference even when the top-ranked stage is incorrect.

At 0.5B and 1.5B, the mean gold-stage ranks are 0.512 and 0.533, statistically indistinguishable from the 0.5 null under candidate-order exchangeability ($p = 0.69$ and $p = 0.91$). From 3B upward, the ranks are 0.393, 0.394, 0.413, and 0.332, all significantly better than chance ($p < 2 \times 10^{-4}$). Thus the particular output read-out tested here becomes informative at larger scales, but forced-choice accuracy remains substantially below probe accuracy. The average spread between candidate logits ranges from 0.41 to 6.20 nats across the ladder and is largest at 32B, so the result is not explained by uniformly tied candidate scores.

scale	probe	forced choice	majority	gap	gold rank	p
0.5B	0.679 ± 0.059	0.267	0.304	$+\mathbf{0.413}$	0.512	0.69
1.5B	0.808 ± 0.050	0.221	0.304	$+\mathbf{0.588}$	0.533	0.91
3B	0.779 ± 0.052	0.404	0.304	$+\mathbf{0.375}$	0.393	$< 10^{-4}$
7B	0.863 ± 0.044	0.333	0.304	$+\mathbf{0.529}$	0.394	$< 10^{-4}$
14B	0.858 ± 0.044	0.308	0.304	$+\mathbf{0.550}$	0.413	1.5×10^{-4}
32B	0.871 ± 0.042	0.450	0.304	$+\mathbf{0.421}$	0.332	$< 10^{-4}$

Table 3: Hidden-state probe accuracy and teacher-forced output read-out accuracy on the same held-out 240 instances at each scale. The majority column is computed on this held-out split. Gold rank is the normalized rank of the correct stage under the candidate logits (0 is best; the exchangeable null has mean 0.5). The probe-minus-output gap remains between 0.375 and 0.588 across the 64× scale range.

The probe-minus-forced-choice gap is 0.413, 0.588, 0.375, 0.529, 0.550, and 0.421 from 0.5B through 32B. It therefore remains large throughout the 64× scale range rather than shrinking steadily with model size. We interpret the rank result conservatively: at the two smallest scales this read-out does not detect bottleneck information in the output logits, whereas from 3B upward it does. This distinction does not establish that the property is wholly absent or present elsewhere in the output computation.

5.4 Probe and stated answers fail on different instances

scale	n	stated accuracy	majority	parsed	modal answer
0.5B	600	26.8±3.5	29.8	100.0	0 (83%)
1.5B	600	20.3±3.2	29.8	100.0	2 (34%)
3B	600	38.8±3.9	29.8	100.0	0 (39%)

Table 4: Accuracy (%) of the model's generated bottleneck answer on the 600-instance subset used for the stated-answer analysis.

scale	both right	probe only	stated only	neither	probe right where stated wrong	stated right where probe wrong	odds ratio	p
0.5B	52	102	3	83	**55.1%**	3.5%	14.10	$<10^{-8}$
1.5B	28	146	23	43	**77.2%**	34.8%	0.36	0.002
3B	61	103	21	55	**65.2%**	27.6%	1.55	0.188

Table 5: Instance-level overlap between probe correctness and stated-answer correctness on examples held out of probe training. The odds ratio and Fisher exact p-value test association between the two binary success indicators.

On the instances for which the model's stated answer is wrong, the hidden-state probe is correct on 55.1%, 77.2%, and 65.2% at 0.5B, 1.5B, and 3B. The joint contingency table further shows that the association between probe success and stated-answer success changes with scale. At 0.5B, stated success is concentrated on examples the probe also gets right (odds ratio 14.1, $p < 10^{-8}$). At 1.5B the association reverses (odds ratio 0.36, $p = 0.002$), and at 3B it is not statistically significant (odds ratio 1.55, $p = 0.188$).

These results show that probe accuracy and stated-answer accuracy are not interchangeable measurements. In particular, the hidden-state probe can recover the correct bottleneck on many examples for which the generated answer is wrong. The data do not by themselves determine whether the decoded feature is used by the computation that produces the answer; the steering analysis below addresses the narrower question of whether the fitted probe direction behaves as a stable intervention target.

5.5 PROBE-BASED STEERING IS NOT STABLE ACROSS PROBE FITS

The layer profile identifies where the bottleneck is most linearly decodable, but it does not imply that the fitted probe direction is a stable causal object. We test this directly at the validation-selected layer. At the final prompt position and each generated position, we add a unit direction scaled to α times the mean residual norm. We compare three norm-matched directions: the weight row associated with the probe's own per-instance prediction, a different-class weight row from the same probe, and a fixed random direction. The gold label is never used to choose the intervention direction.

On one partition, the three interventions produce sharply different outcomes. At $\alpha = 1.5$, stated accuracy is 0.215 for the probe direction, compared with an unsteered baseline of 0.210, 0.010 for a different-class direction, and 0.025 for a random direction; the paired contrast reported by the original run is $p = 2 \times 10^{-11}$. Considered alone, this run would suggest that the fitted probe direction is behaviorally distinguished from the controls.

			$\alpha = 1.0$			$\alpha = 1.5$		
split	layer	none	probe	wrong	random	probe	wrong	random
1	18	0.210	0.295	0.210	0.105	0.215	0.010	0.025
2	16	0.208	0.129	0.062	0.163	–	–	–
3	16	0.233	0.000	0.017	0.146	0.025	0.000	0.271
4	14	0.229	0.021	0.042	0.017	0.004	0.033	0.013

Table 6: Stated-answer accuracy at 1.5B under rank-one residual additions, evaluated on held-out instances for four independent data partitions. Each row refits the probe and reselects the layer. The qualitative ordering of the probe, different-class, and random directions is not stable across partitions.

Refitting the probe on three additional partitions changes that conclusion. The selected layer moves among 18, 16, and 14, and the ordering of the three intervention arms changes across runs. On partition 3, for example, the random direction reaches 0.271 at $\alpha = 1.5$ while the probe direction reaches 0.025, reversing the qualitative pattern of partition 1. Across the four partitions, no arm ordering is preserved.

The stable object in these runs is the predictive task, not the fitted intervention vector: held-out probe accuracies are similar across partitions (0.720, 0.767, 0.767, and 0.783), while the selected direction and its behavioral effect vary substantially. This is consistent with broader concerns about representation identifiability (Nelson et al., 2026). The practical implication is limited but important: wrong-class, random-direction, and paired controls evaluate a fixed fitted vector, whereas a refit tests whether the intervention conclusion survives another statistically comparable probe fit.

5.6 THE PROBE RESULT GENERALIZES ACROSS MODEL FAMILIES

The six-scale ladder holds pretraining family and instruction recipe fixed, which makes scale comparisons easier to interpret but leaves open whether the representation result is family-specific. We therefore repeat the same probing protocol on DeepSeek-Coder 1.3B and StarCoder2 3B using the same 1,200 instances, split, layer-selection rule, regularization, and controls.

Bottleneck probes reach 86.7% on DeepSeek-Coder 1.3B and 84.6% on StarCoder2 3B, compared with 80.8% for Qwen2.5-Coder 1.5B under the same protocol. In all three families, the bottleneck probe is well above the majority, surface-feature, and shuffled-label controls. The two lexically explicit targets are recovered in the early part of each network, while the selected bottleneck layer is at relative depth 0.50, 0.50, and 0.40. We therefore observe a similar mid-network bottleneck read-out across three independently trained code-model families.

For the two instruction-tuned families on which the teacher-forced answer read-out is defined, forced-choice bottleneck accuracy is 22.1% for Qwen2.5-Coder 1.5B and 27.1% for DeepSeek-Coder 1.3B, below the held-out majority baseline in both cases. Their gold-stage ranks are also indistinguishable from chance ($p = 0.91$ and $p = 0.60$). The corresponding probe-minus-read-out gaps are 58.8 and 59.6 percentage points. We do not report an analogous output read-out for StarCoder2 because the tested checkpoint does not provide the same instruction-following interface.

family and scale	target	maj.	surf.	shuf.	probe	depth
Qwen2.5-Coder 1.5B	bottleneck idx	30.4	24.6	21.2	$\mathbf{80.8} \pm 5.0$	0.50
	is serialized	66.2	79.6	57.1	$\mathbf{100.0} \pm 0.0$	0.07
	has unindexed	76.2	76.7	76.7	$\mathbf{100.0} \pm 0.0$	0.07
	pool binds	59.2	58.8	45.0	$\mathbf{95.0} \pm 2.8$	0.64
	bottleneck, read-out	30.4	–	–	22.1 ± 5.2	–
DeepSeek-Coder 1.3B	bottleneck idx	30.4	24.6	21.7	$\mathbf{86.7} \pm 4.3$	0.50
	is serialized	66.2	79.6	59.6	$\mathbf{100.0} \pm 0.0$	0.08
	has unindexed	76.2	76.7	73.8	$\mathbf{100.0} \pm 0.0$	0.08
	pool binds	59.2	58.8	50.8	$\mathbf{97.1} \pm 2.1$	0.42
	bottleneck, read-out	30.4	–	–	27.1 ± 5.6	–
StarCoder2 3B	bottleneck idx	30.4	24.6	20.8	$\mathbf{84.6} \pm 4.6$	0.40
	is serialized	66.2	79.6	57.5	$\mathbf{100.0} \pm 0.0$	0.10
	has unindexed	76.2	76.7	70.8	$\mathbf{100.0} \pm 0.0$	0.10
	pool binds	59.2	58.8	48.8	$\mathbf{95.8} \pm 2.5$	0.50

Table 7: The probing protocol applied to three code-model families using the same 1,200 instances, split, regularization, and controls. Depth is the selected layer divided by the number of layers. Bottleneck probes are well above all controls in each family. The teacher-forced output read-out is reported only for the two instruction-tuned checkpoints for which the same answer interface is available.

6 Discussion

The experiments support three conclusions about probing system-level reasoning in code models. First, a capacity bottleneck that must be composed from several quantities in the handler is linearly decodable from intermediate representations across scales and model families, substantially above explicit surface and shuffled-label controls. Second, decodability is not equivalent to output accessibility: the teacher-forced candidate distribution is much less accurate, and only at 3B and above do we detect a statistically informative ranking of the correct stage. Third, probe-based interventions require their own stability checks. In our steering experiment, refitting the probe across data partitions changes both the fitted direction and the qualitative intervention outcome even when held-out probe accuracy remains similar.

These findings argue for separating three questions that are often conflated in representation analyses: whether a property can be decoded, whether the model's output distribution exposes it, and whether a particular fitted direction supports a reproducible intervention. The present study is deliberately controlled: labels come from generated services with a known capacity law, and the strongest conclusions concern the tested model families, prompting format, and linear read-outs. Extending the analysis to real repositories and independently measured performance bottlenecks is a natural next step.

Reproducibility statement

The supplementary package contains the instance generator and renderer, activation-extraction and probing scripts, output-read-out and steering code, the exact generated instances, and cached analysis outputs sufficient to regenerate the reported tables and figures without the large activation tensors. The extraction scripts can regenerate those tensors from the released instances and public model weights. The paper specifies the data split, probe regularization, layer-selection rule, and statistical tests used in the reported analyses.

Ethics statement

The study uses generated service specifications and publicly available model weights. It contains no personal data and involves no human subjects.

AI USE STATEMENT

GPT was used only to polish the writing of this manuscript.

REFERENCES

Bassam Adnan, Aviral Gupta, Sreemaee Akshathala, and Karthik Vaidhyanathan. Archbench: Benchmarking generative-ai for software architecture tasks. *arXiv:2603.17833*, 2026.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. In *ICLR Workshop*, 2017.

Amos Azaria and Tom Mitchell. The internal state of an llm knows when it's lying. In *Findings of EMNLP*, 2023.

Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. In *ICLR*, 2023.

Peter J. Denning and Jeffrey P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, 1978.

Carlo Di Cicco. Code correctness signals in LLM hidden states: Pre-generation probing and repair geometry. *arXiv:2606.14530*, 2026.

Ion George Dinu, Marian Cristian Mihăescu, and Traian Rebedea. Smellbench: Evaluating llm agents on architectural code smell repair. *arXiv:2605.07001*, 2026.

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175, 2021.

Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *ASPLOS*, 2019.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. Patchscopes: A unifying framework for inspecting hidden representations of language models. In *ICML*, 2024.

Giuliano Gorgone and Fausto Carcassi. Typeprobe: Recovering type representations from hidden states of pre-trained code models. *arXiv:2607.08339*, 2026.

Wes Gurnee and Max Tegmark. Language models represent space and time. In *ICLR*, 2024.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *EMNLP*, 2019.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *arXiv:2307.13702*, 2023.

Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *ICLR*, 2023a.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Advances in Neural Information Processing Systems*, 2023b.

John D. C. Little. A proof for the queuing formula: $l = \lambda w$. *Operations Research*, 9(3):383–387, 1961.

Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *Conference on Language Modeling (COLM)*, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In *NeurIPS*, 2022.

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. In *BlackboxNLP Workshop*, 2023.

Walter Nelson, Marco Fumero, Theofanis Karaletsos, and Francesco Locatello. Statistical and structural identifiability in representation learning. *arXiv:2603.11970*, 2026.

Jinwoo Park, Byungkwon Choi, Chunghan Lee, and Dongsu Han. Graf: A graph neural network based proactive resource allocation framework for slo-oriented microservices. *ACM CoNEXT*, 2021.

Subramanyam Sahoo, Vinija Jain, Aman Chadha, and Divya Chaudhary. Linear probes detect task format, not reasoning mode in language model hidden states. *arXiv:2606.02907*, 2026.

Gustavo Sandoval, Brendan Dolan-Gavitt, and Siddharth Garg. Surgical repair of insecure code generation in llms. *arXiv:2604.16697*, 2026.

Tiziano Santilli, Francesco Daghero, and Mayhar Tourchi Moghaddam. Sake: Software architectural knowledge evaluation benchmark for large language models. *arXiv:2606.29520*, 2026.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv:2206.05802*, 2022.

Gagan Somashekar, Anurag Tandon, Anush Kini, Chien-Chun Chang, Petr Husain, and Anshul Gandhi. Gamma: Graph neural network-based multi-bottleneck localization for microservices applications. In *The Web Conference (WWW)*, 2024.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *arXiv:2402.08115*, 2024.

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

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

Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *USENIX NSDI*, 2016.

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 183–196, 2020.

Yuexi Yang, Alyssa Wu, Ji Luo, Richeng Xuan, Zhichao Hu, Yuhong Liu, et al. Repoprobe: Benchmarking architecture-aware repository comprehension with checklists. *arXiv:2608.04783*, 2026.

Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. Sinan: Ml-based and qos-aware resource management for cloud microservices. In *ASPLOS*, 2021.

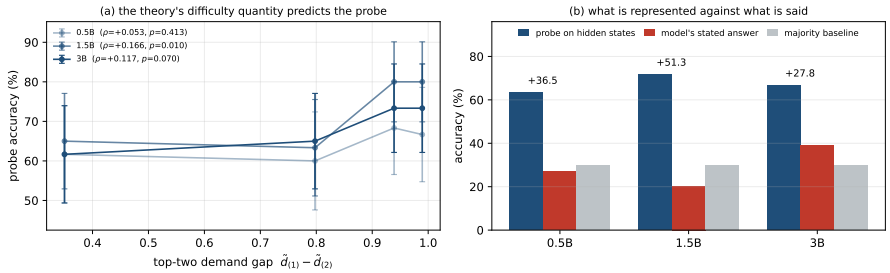

Figure 3: (a) Probe accuracy by quartile of the top-two demand gap, with binomial intervals, on held-out instances at the validation-selected layer. The legend gives per-scale Spearman correlations and p-values. (b) Hidden-state probe accuracy and the model's stated-answer accuracy, together with the majority baseline and their difference.

A THE TOLERANCE QUANTITY AND THE PROBE

The capacity model also provides a simple measure of identification difficulty. Let $\gamma = \tilde{d}_{(1)} - \tilde{d}_{(2)}$ be the gap between the largest and second-largest relative demands. If every estimated demand is within ϵ of its true value, the leader remains correctly ordered whenever $\gamma > 2\epsilon$: the estimated leader is at least $\tilde{d}_{(1)} - \epsilon$, while the estimated runner-up is at most $\tilde{d}_{(2)} + \epsilon$. Conversely, when the gap is below this threshold, an error pattern of the same magnitude can reverse the top two demands. Larger gaps therefore make bottleneck identification more tolerant to estimation error.

Pooled over the three scales used for this analysis, 720 held-out instance-scale observations give a Spearman correlation of $+0.111$ between the demand gap and per-instance probe correctness ($p = 0.003$). In a logistic regression of correctness on the gap, number of resources, and scale fixed effects, the gap coefficient is $\beta = +0.898$ (se $= 0.315$, $p = 0.004$), while the number of resources has $\beta = -0.392$ ($p = 10^{-4}$). Accuracy by pooled gap quartile is 0.628, 0.628, 0.739, and 0.733. Per-scale Spearman correlations are $+0.166$ ($p = 0.010$) at 1.5B, $+0.117$ ($p = 0.070$) at 3B, and $+0.053$ ($p = 0.413$) at 0.5B, each on 240 held-out examples.

The pooled association is statistically detectable but modest, and the quartile means are not strictly monotone. We therefore treat the demand gap as an interpretable task-difficulty correlate rather than a sufficient predictor of probe correctness.

B WHICH LAYER TO READ FROM

At the smallest scale, the selected layers for the four targets span early, middle, and late parts of the network. Across scales, the bottleneck probe is never best at the final layer, whereas the two lexically explicit targets become decodable near the input and the pool-binding target tends to improve later. This pattern motivates selecting the read-out layer on validation data rather than defaulting to the final layer.

C RENDERED INSTANCE

A rendered handler declares its pools, defines one helper per resource kind, and calls them in the specified order. The unindexed aggregate carries an explicit note, the runtime is identified by the pool constructor, and the admission-pool size is stated. Costs span roughly two orders of magnitude across resource kinds and visit counts span approximately one order of magnitude.

D SURFACE FEATURES

The surface control uses the number of resources, the maximum per-visit cost and its index, the maximum visit count and its index, and the character and line counts of the rendered file. Its bot-

tleneck accuracy remains near the held-out class baseline because the target depends on combining cost, visit count, and the runtime sharing regime rather than on any one of these literals.