STEERING LLM CODING BEHAVIORS WITH CONCEPT VECTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models follow coding-practice instructions, but prompt-only control neither reveals internal structure nor avoids repeating requirements. We test whether coding behaviors admit useful residual directions. Using factorized multilingual prompts, we generate and teacher-force code from six Qwen2.5-Coder-Instruct models (0.5B–32B) and estimate compliance-conditioned post-token mean differences for commenting, logging, detected defensive constructs, and programming language. The prompt-conditioned contrasts show coherent lexical and cross-language structure. Adding them after the first sampled token changes held-out outputs, but effects depend on model and layer, strong scaling harms validity and correctness, and axis specificity remains unresolved without matched-norm causal controls. Among endpoint-compliant tasks, explicit comment prompts shift outputs along the training-derived direction. Pre-summed vectors share one hook and one residual addition per steered decode call. In one correctness-screened 7B case, prompting and steering both reach 100% detected compliance; steering uses 17.6% fewer input but 5.1% more output tokens (9.0% fewer total), with 0.43 Python pass versus 0.44 for prompting. None of six tested positive strengths qualifies for a logging-plus-defensive profile. These empirical controls require model- and layer-specific calibration and executable evaluation.

1 INTRODUCTION

Code generation systems are routinely asked to control properties beyond functional intent: use a particular programming language, include comments, emit diagnostic logs, validate inputs, or avoid such behavior when concise code is preferred. Natural-language prompting is the standard interface for these requirements. It is effective, but it leaves two questions open. First, what internal representations support these controllable coding behaviors? Second, can those representations be used directly, providing graded and compositional control without restating every requirement in every prompt?

Activation steering offers a simple way to ask both questions. Prior work has found directions for sentiment, truthfulness, refusal, and in-context functions in general-purpose language models (Turner et al., 2023; Panickssery et al., 2024; Li et al., 2023; Arditi et al., 2024; Todd et al., 2024). However, code generation differs from ordinary text in several important respects. Outputs have executable semantics; surface changes can break tests even when they satisfy a style detector; programming languages induce large representational shifts; and useful engineering behaviors may be sparse, syntactically constrained, and entangled with output length. Prior work has begun to steer code-model language and library choices (Rahman et al., 2026), but it remains unclear whether broader coding behaviors form interpretable directions, how they interact with task and language, whether they recur across scale, and what tradeoffs joint steering has relative to prompting.

We address these questions with a controlled study of the six Qwen2.5-Coder-Instruct checkpoints from 0.5B to 32B parameters (Hui et al., 2024). For 100 training tasks, three programming languages, and a complete 2^3 grid over comments/docstrings, logging/observability, and defensive programming, we generate 2400 outputs per checkpoint. We then replay accepted outputs under each model's chat template with teacher forcing and mean-pool the post-block residual at generated code-body positions. Task-matched differences yield one compliance-conditioned residual direction per behavior and layer;

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.

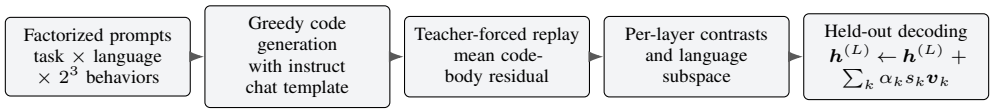


Figure 1: Study design. Prompts create controlled output contrasts; residual directions are estimated from teacher-forced generated-code activations. At inference, scaled vectors are pre-summed and one residual delta is applied on each steered one-token decode call after the first sampled token.

complete cross-language tuples yield a centered language subspace. The pooled positions exclude prompt tokens, although their causal states remain conditioned on the prompt.

We organize the evidence as four linked claims. First, controlled prompts expose prompt-conditioned coding-behavior contrasts throughout the residual stream. Second, the resulting directions have coherent descriptive structure: they project to recognizable code tokens, survive removal of literal comment tokens, decompose alongside task and language factors, and support an analogy-style relation across languages. Third, adding a selected direction during decoding changes held-out generations, although the present controls do not establish that strong interventions are axis-specific. The effect is signed and layer dependent, but strong coefficients can reduce parseability or functional correctness. Fourth, the directions support richer control. Vector sums can induce multiple attributes, compliant prompt-strength endpoints align with the training-derived direction, and a calibrated two-vector case study can replace two repeated textual requirements with a one-point observed pass-rate difference whose confidence interval does not establish equivalence.

Figure 1 summarizes the protocol. We deliberately separate descriptive geometry, causal intervention, and deployment-oriented calibration. A direction can be statistically visible without being useful at every layer; it can causally increase a detector while harming execution; and individually meaningful directions need not compose safely. This separation produces a more qualified conclusion than "steering works": prompt-conditioned behavior directions recur across the model family, but useful operating points are checkpoint- and profile-specific.

Our contributions are:

- a factorized, multilingual extraction protocol for per-layer coding-behavior directions across six instruct code models;
- convergent descriptive evidence from variance decomposition, cross-language alignment, analogy retrieval, vocabulary projection, and lexical random-direction controls;
- held-out interventions that report behavior and executable correctness together, plus signed coefficient and all-layer sweeps that expose both control and failure regimes while leaving strong-intervention specificity unresolved; and
- compositional and prompt-alignment experiments, including a correctness-screened cost comparison where multiple requirements share one residual intervention.

2 Related Work

Linear representations and activation steering. The linear representation hypothesis treats some high-level properties as directions or low-rank subspaces in model representations (Park et al., 2024). Activation Addition and Contrastive Activation Addition estimate such directions from contrasting examples and intervene by adding a scaled vector during inference (Turner et al., 2023; Panickssery et al., 2024); latent vectors have also been optimized directly for target generations (Subramani et al., 2022). Related methods use activation interventions to improve truthfulness (Li et al., 2023), control refusal (Arditi et al., 2024; Lee et al., 2025), and elicit functions represented by compact vectors (Todd et al., 2024). Representation engineering provides a broader framework for these methods (Zou et al., 2023). We use the established difference-in-means estimator; the contribution is the coding-specific experimental design and stress test, not a new steering algorithm.

Our study differs in both domain and evaluation. Coding behaviors must be assessed jointly with syntax and executable semantics, and language is a major nuisance factor rather than merely another topic label. The closest prior work steers global programming-language and library choice (Rahman

Table 1: Model and extraction setup. Each checkpoint uses 100 train, 100 validation, and 100 test tasks, three languages, and the same prompt grid. Validity is the training-grid syntax-validity rate.

Model	Layers	$L^\star$	Syntax valid
0.5B	24	16	0.960
1.5B	28	19	0.988
3B	36	24	0.998
7B	28	19	1.000
14B	48	32	1.000
32B	64	43	1.000

et al., 2026). We instead target local coding practices, factor task, language, and three behaviors, audit six model sizes, execute Python outputs, and expose sign reversals, degeneration, calibration failures, and conditional prompt alignment.

Interpreting intermediate representations. Vocabulary projection and logit-lens methods decode hidden states through the model's output head; the tuned lens documents both their usefulness and their limitations (Belrose et al., 2023). We use a training-free, final-RMSNorm-folded projection only as one source of evidence, and compare against matched random directions because a large vocabulary produces nontrivial extreme cosine values by chance. Analogy-style vector arithmetic follows the geometric intuition of classic embedding work (Mikolov et al., 2013), but our test is performed on task-residualized contextual code representations and is framed as alignment rather than exact word-embedding algebra.

Controllable code generation. MBPP and multilingual extensions provide compact, executable program-synthesis tasks (Austin et al., 2021; Cassano et al., 2022); HumanEval established a complementary execution-based evaluation protocol (Chen et al., 2021). Prompt and prefix tuning move control information into learned continuous parameters (Li & Liang, 2021; Lester et al., 2021), whereas our vectors require no weight updates and are applied directly to one residual stream during decoding. Prompting remains the default interface: the primary sweep has no matched prompt-only arm, and only the calibrated case study in Section 7 compares them directly. We do not claim universal replacement.

3 Experimental Design and Concept Extraction

3.1 Models, tasks, and factorized prompts

We study all six instruction-tuned checkpoints in the Qwen2.5-Coder family (Hui et al., 2024). Table 1 lists layer counts and the pre-specified layer $L^\star = \mathrm{round}(2n_{\mathrm{layers}}/3)$ fixed before test-split evaluation. Appendix B details the prompts and detectors.

We start from the Python, Java, and C++ subsets of `allenai/multilingual_mbpp`. After deduplicating task IDs, we retain tasks present in all three languages and select the 300 with the longest minimum reference-solution length across languages. This avoids a benchmark dominated by one-line programs while preserving parallel task identity. A seeded hash split assigns 100 tasks each to train, validation, and test. Original MBPP Python assertions are joined by task ID and used only for steering evaluation, never to filter extraction examples (Austin et al., 2021; Cassano et al., 2022).

For every training task and language, the prompt crosses three binary factors: comments/docstrings, logging/observability, and defensive programming, operationalized as detected defensive constructs. The positive and negative clauses are explicit and symmetric in intent (e.g., include versus omit comments). This gives $100 \times 3 \times 2^3 = 2400$ prompts per checkpoint. We apply the checkpoint's native chat template and greedily decode up to 1,200 new tokens. Static language-aware detectors record comments/docstrings, diagnostic output, and a union of defensive constructs (exceptions, null checks, assertions, explicit failures, validation, and guard clauses). Syntax validity uses an AST parse for Python and balanced-delimiter checks for Java/C++; Python functional correctness is evaluated later in an isolated subprocess with an 8-second and 1-GB limit.

3.2 TEACHER-FORCED CODE REPRESENTATIONS

For each generated program y, we render a two-turn chat containing the original user prompt and the parsed assistant code, with `add_generation_prompt=False`. We run one teacher-forced forward pass and capture the output residual of every decoder block. Let $\boldsymbol{h}_i^{(L)}$ denote the post-block residual at layer L and token position i, and let $C(y)$ contain only tokens overlapping the generated code body, excluding prompt positions, chat control tokens, Markdown fences, EOS, and padding. These post-token states have observed token i and still attend to the original prompt; without neutral-context re-encoding, they may retain clause effects and are not demonstrated antecedent control states. The representation used for extraction is

$$\bar{\boldsymbol{h}}^{(L)}(y) = \frac{1}{|C(y)|} \sum_{i \in C(y)} \boldsymbol{h}_i^{(L)}. \tag{1}$$

Right padding, explicit position IDs, and per-row attention masks make the pooled value invariant to batch mates up to numerical precision. Only pooled vectors are persisted; no token-by-layer activation cache is stored.

We accept generations that parse and whose detector outcome agrees with the requested side of the target contrast. This compliance-conditioned estimand is not the intention-to-treat effect of prompt assignment and can inherit outcome-selection differences. Compliance is evaluated separately by behavior, so a row may contribute to one estimate and not another. Functional correctness is not an extraction criterion, which avoids directly selecting on test passage. Throughout, "concept vector" is shorthand for this operational contrast, not a claim of an isolated latent variable.

For behavior b, we match positive and negative rows on task, language, template, sample index, and the two non-target behavior settings. The raw direction is the average within-pair difference,

$$\tilde{\boldsymbol{v}}_b^{(L)} = \frac{1}{|\mathcal{P}_b|} \sum_{p \in \mathcal{P}_b} \left(\bar{\boldsymbol{h}}_{p,b=1}^{(L)} - \bar{\boldsymbol{h}}_{p,b=0}^{(L)} \right), \qquad \boldsymbol{v}_b^{(L)} = \frac{\tilde{\boldsymbol{v}}_b^{(L)}}{\|\tilde{\boldsymbol{v}}_b^{(L)}\|_2}. \tag{2}$$

At least 16 positive examples, negative examples, and matched pairs are required; otherwise the implementation falls back to a class-mean difference when sample counts permit, or marks the vector unavailable. This threshold establishes sample support, not held-out linear separability. A second variant removes comment and docstring token spans before pooling. Its cosine with the full vector is 0.45–0.66 (Appendix Table 4), evidence that the contrast is not solely literal comment tokens, although lexical content remains part of the signal.

Language vectors use complete tuples that hold task and all behavior settings fixed. If $\boldsymbol{\mu}_\ell^{(L)}$ is the mean for language ℓ, we define

$$\boldsymbol{v}_\ell^{(L)} = \boldsymbol{\mu}_\ell^{(L)} - \frac{1}{3} \sum_{j \in \{\mathrm{Py,Java,C{+}{+}}\}} \boldsymbol{\mu}_j^{(L)}. \tag{3}$$

The three centered vectors span a subspace of rank at most two.

3.3 INTERVENTION AND EVALUATION

At generation time we add a scaled delta to the output residual of one decoder block:

$$\boldsymbol{h}^{(L)} \leftarrow \boldsymbol{h}^{(L)} + \sum_{k=1}^{K} \alpha_k s_k^{(L)} \boldsymbol{v}_k^{(L)}. \tag{4}$$

Here $s_k^{(L)}$ is the unnormalized mean-difference norm (or centered class-mean norm for language), so $\alpha = 1$ corresponds to one natural contrast unit. Steering is applied to every continuation decode call after the first token has been sampled, but not to prompt prefill or the first-token logits. For $K > 1$, all deltas are summed once and one hook performs one residual addition per steered decode call.

The main causal sweep varies $\alpha \in \{-8, -4, -2, -1, -0.5, 0, 0.5, 1, 2, 4, 8\}$ at quarter, half, and three-quarter depth, a grid that excludes the shared $L^\star$ used elsewhere. For each behavior, validation

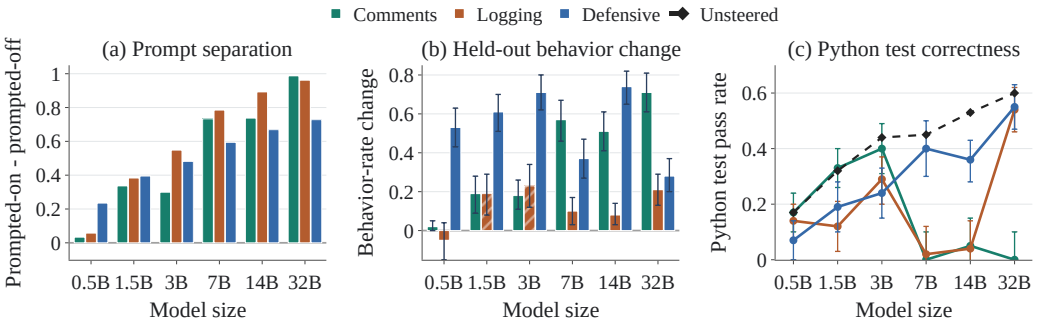


Figure 2: Cross-scale evidence. (a) Factor prompts separate requested-on from requested-off generations. (b) On 100 held-out tasks, directions are applied at validation-selected layer/coefficient settings; error bars are 95% task-bootstrap intervals and hatching marks opposite-sign directional failures. (c) Python test pass rate; bars show paired change intervals centered on each observed rate. Maximizing detector change can be destructive; Section 7 introduces correctness-screened calibration.

selects the layer and coefficient with maximum detector induction, breaking ties toward smaller $|\alpha|$; test tasks then provide a frozen-setting confirmation. This selection deliberately measures perturbation capacity rather than imposing a correctness constraint, so behavior and correctness are always reported together. Follow-up experiments use a dense signed grid at $L^\star$, sweep every layer, and test pre-specified vector sums. Advanced sweeps use 1,000 percentile task-bootstrap resamples; held-out cross-scale intervals and the paired cost comparison use 10,000.

4 Prompt-Conditioned Coding-Behavior Contrasts Recur Across Scale

The extraction procedure requires the prompting intervention to create observable positive and negative examples. Figure 2a reports the requested-on minus requested-off detection rate in the training grid. Separation is weak but nonzero in the 0.5B checkpoint (0.03 comments, 0.06 logging, 0.24 defensive) and substantially larger in the 32B checkpoint (0.99, 0.96, 0.73). Intermediate checkpoints do not follow a simple monotone scaling curve: for example, 3B comment separation is below 1.5B, while logging rises sharply at 3B. Parse rates increase from 96.0% at 0.5B to essentially 100% from 7B onward (Table 1). Estimability does not imply directional validity: 0.5B is a low-compliance boundary with only 39 comment pairs (Appendix Table 5) and negligible held-out comment induction.

The same factor definitions yield available behavior directions at every layer and checkpoint under the fixed sample thresholds. This by itself is not causal evidence: any two classes can have a mean difference in a high-dimensional space. We therefore treat the manipulation check as the first link in a longer chain, and evaluate geometry, held-out intervention, and prompt alignment separately. Figure 2b–c previews the causal result: behavior changes recur across all six sizes, but their magnitude and correctness cost are highly non-monotone.

5 The Contrasts Have Interpretable Descriptive Structure

We next ask whether the contrasts have reproducible descriptive structure rather than being arbitrary separating hyperplanes. Three analyses examine factorial variance decomposition, cross-language alignment, and token-level projection.

5.1 A sequential decomposition separates task and behavior terms

For each layer, we fit a multivariate factorial model to the pooled training activations with task, language, comment condition, and their interactions. Logging, defensive programming, and their comment interactions are entered first as nuisance covariates because compliance dropout is correlated

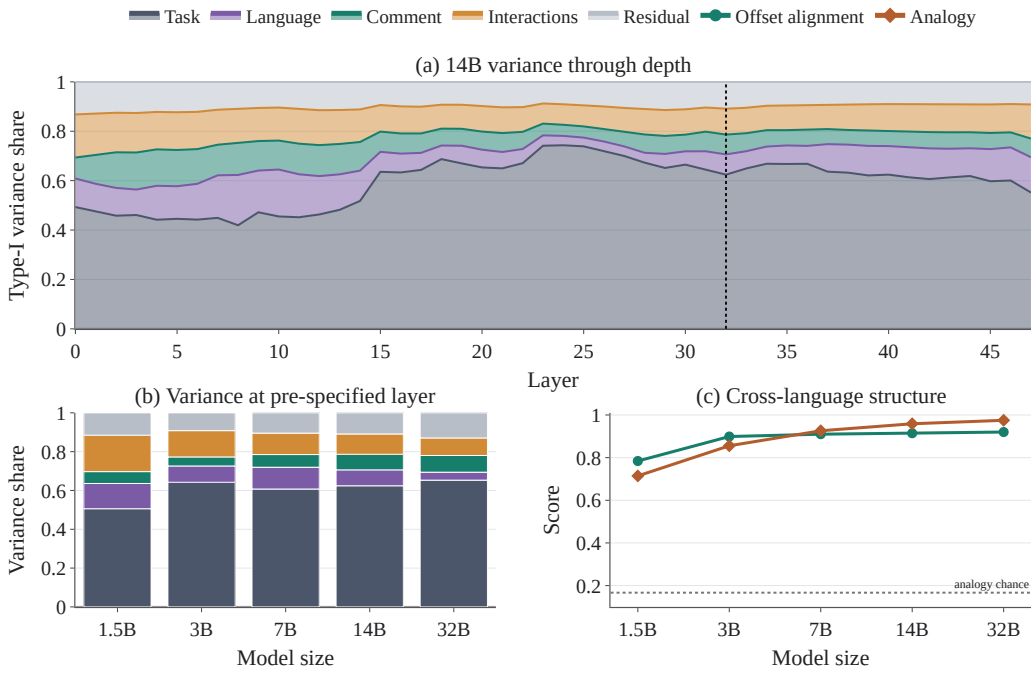


Figure 3: Representational structure. (a) Type-I decomposition for 14B; the dotted line is $L^\star = 32$. (b) Decomposition at each checkpoint's $L^\star$ (0.5B is unavailable because no task satisfies complete-cell coverage; the other cohorts contain 56/45/99/97/100 tasks). (c) Mean pairwise cosine among comment offsets and analogy top-1; the dotted line is six-class chance.

with those factors. We report Type-I sums of squares over the remaining adjusted total; the components sum to one. To avoid imputing missing cells, a task enters only if all language-by-comment cells are represented. This leaves 45–100 tasks for 1.5B–32B and none for 0.5B, whose result is therefore marked unavailable rather than estimated under a relaxed rule.

Figure 3a shows the 14B decomposition through depth. Under the pre-specified term ordering, task identity has the largest sequential share and peaks in the middle layers, while language and commenting remain visible. At the shared $L^\star$ across estimable models, task explains 50.6–65.3% of adjusted variance, language 4.1–13.0%, comments 4.6–8.6%, and the direct language-by-comment term only 0.5–1.6%. The direct interaction is descriptively small under this order-sensitive Type-I decomposition, but it does not establish language independence. Type-III marginal shares preserve the ordering; per-df shares need not (Appendix Tables 10–11). Thus this is design-specific variance allocation, not intrinsic factor importance.

5.2 Comment offsets align across programming languages

Let $\boldsymbol{d}_\ell$ be the adjusted comment-on minus comment-off offset within language ℓ. At $L^\star$, the mean pairwise cosine among $\boldsymbol{d}_{\text{Python}}$, $\boldsymbol{d}_{\text{Java}}$, and $\boldsymbol{d}_{\text{C++}}$ ranges from 0.78 to 0.92 across the five estimable checkpoints (Figure 3c). We also test an analogy-style relation:

$$\bar{\boldsymbol{h}}(t, \text{C++}, \text{plain}) + [\bar{\boldsymbol{h}}(t, \text{Python}, \text{commented}) - \bar{\boldsymbol{h}}(t, \text{Python}, \text{plain})] \approx \bar{\boldsymbol{h}}(t, \text{C++}, \text{commented}). \tag{5}$$

Before retrieval, each cell is residualized by the mean of the other cells for the same task, and the three source cells are excluded from the candidate bank. Global top-1 retrieval of the target language/comment class is 0.71–0.98 at $L^\star$ (six-class chance is $1/6$). Because this is a transductive, in-sample diagnostic whose within-task centering uses all other cells, it is descriptive evidence of an analogy-style relation, not held-out generalization or exact word-embedding algebra (Mikolov et al., 2013).

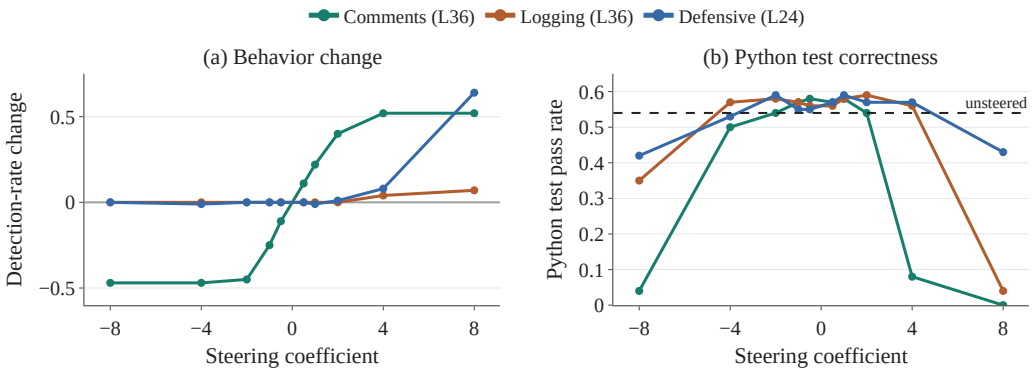


Figure 4: Signed 14B validation sweep. Each behavior is shown at the layer selected for maximum induction. The same coefficient can be nearly inert, useful, or destructive depending on the axis. Correctness remains near the unsteered rate for moderate comment coefficients but collapses at strong comment and logging settings.

5.3 Vocabulary projections provide lexical diagnostics

We fold the final RMSNorm gain into the unembedding matrix and rank vocabulary rows by cosine with each 14B direction. Appendix Table 13 reports representative tokens and their ranks among the top ten. Logging retrieves variants of `print`, `debug`, and Java `println`; defensive programming retrieves `valid`, exception classes, and `ValueError`; language directions retrieve characteristic syntax. Comment tokens are less purely lexical but include `TODO` and `NOTE`. Aligned code-window projections likewise place high scores on comments, logging calls, and guard/exception regions. These are lexical associations with post-token states, not next-token logit attributions.

The magnitude of the maximum cosine is not a sufficient control: random directions also obtain extreme top-vocabulary cosines around 0.06–0.08 because of vocabulary size. Their ranked identities, however, are incoherent mixtures of rare and multilingual fragments; none of the 30 random-control rankings evaluated for the 14B analysis recovers the corresponding behavior-specific token set; one example is shown in the appendix. We therefore interpret token identity and rank together with geometry and intervention, rather than treating a large maximum cosine as mechanistic proof.

6 Selected Direction Injections Alter Held-Out Code

We intervene on neutral prompts that contain the task and, for Python, the required function signature, but no instruction about comments, logging, or defensive programming. Every task has an $\alpha = 0$ baseline generated in the same run. Figure 4 shows the signed 14B sweep at each behavior's validation-selected layer. Commenting changes bidirectionally: at $\alpha = -4$, detection falls 0.47 while pass is 0.50 versus 0.54 unsteered; $\alpha = -8$ gives the same suppression but only 0.04 pass. Detected defensive constructs have a later onset and then rise sharply. Logging is detectable but much weaker. These differences show why "a steering vector" is not a complete operating specification; layer and scale are part of the intervention.

6.1 Held-out effects recur across scale, but not monotonically

Table 2 evaluates the exact validation-selected layer and coefficient on 100 disjoint test tasks. Comment induction ranges from +0.02 to +0.71, defensive-construct induction from +0.28 to +0.74, and logging from -0.05 to +0.23. Validation selects $\alpha = -4$ for logging at 1.5B and 3B, opposite the positive-minus-negative extraction sign; we count these as directional failures rather than positive induction. The point estimates are nonmonotone, but separately selected settings and compliance-conditioned cohorts differ across checkpoints, so this is not a controlled scaling-law comparison. Every checkpoint nevertheless has at least one substantial held-out effect.

Table 2: Held-out behavior-rate changes at validation-selected settings, with 95% task-bootstrap intervals. A dagger marks an opposite-sign selection, treated as a directional failure. Python test pass rates at these settings are in Figure 2c.

Model	Comments Δ	Logging Δ	Defensive Δ
0.5B	+0.02 [+0.00,+0.05]	-0.05 [-0.15,+0.04]	+0.53 [+0.43,+0.63]
1.5B	+0.19 [+0.09,+0.28]	+0.19 [+0.08,+0.29]†	+0.61 [+0.51,+0.70]
3B	+0.18 [+0.11,+0.26]	+0.23 [+0.12,+0.34]†	+0.71 [+0.62,+0.80]
7B	+0.57 [+0.46,+0.67]	+0.10 [+0.03,+0.17]	+0.37 [+0.27,+0.47]
14B	+0.51 [+0.41,+0.61]	+0.08 [+0.03,+0.14]	+0.74 [+0.65,+0.82]
32B	+0.71 [+0.61,+0.81]	+0.21 [+0.13,+0.29]	+0.28 [+0.20,+0.37]

Figure 2c makes correctness equally important. Settings selected solely for maximum detector effect can be unusable: 7B and 32B comment steering reaches 0% pass rate, and 14B logging reaches 4%, even though the corresponding detectors move in the intended direction. Conversely, 32B logging and defensive steering retain 0.54 and 0.55 pass rates against a 0.60 baseline. These results reject the interpretation that any detector increase is automatically a useful coding intervention.

The target detector has the largest absolute change in 14 of 18 cells (13 by signed change), but specificity is uneven. At 7B all axes use (L21, $\alpha = 8$), yet comment and logging vectors each change comments by +0.57; at 32B all use (L48, $\alpha = 4$), yet logging changes comments more than logging (+0.24 versus +0.21). Natural norms differ, so this is only a partial control. No matched-norm random/shuffled steering or prompt-only arm is run; 14 selections use $|\alpha| \geq 4$. These results establish intervention effects, not axis specificity or superiority to prompting (Appendix Tables 7–8).

Programming-language directions are a positive control because language is global and easy to recognize. A Qwen2.5-Coder-14B-Instruct judge gives validation-selected C++/Java flips of 0.92–0.99/0.82–1.00, versus 0.56–1.00/0.57–1.00 from conservative syntax rules. The judge is one evaluated checkpoint and self-judges its 14B row; neither detector is ground truth. We retain both views and report aggregate agreement in Appendix B.

6.2 Steerability is localized, but there is no universal depth

The three-layer sweep could miss narrow regions, so we repeat comment and C++ interventions at every layer using two fixed coefficients. Strong effects often occur late, but overlap sets can be broad or disconnected and the maximizing location varies by checkpoint and target. We report a descriptive pointwise-CI-overlap set, not a simultaneous confidence set: for 32B it is 53–55 for rule-based C++, $\{48, 53\}$ under the judge, and broad and disconnected for comments around a layer-61 maximum. Because natural and residual norms vary through depth, this sweep localizes effective perturbations rather than intrinsic direction quality. Strong coefficients also cause layer-specific extraction failure or very short code (Appendix Figure 6). The layer-61 comment maximum is degeneration-guarded; it need not match the unguarded three-layer selection at layer 48 in Table 7.

7 Continuous, Compositional, and Prompt-Related Control

7.1 Continuous control is bounded by degeneration

At the shared layer $L^\star$, dense signed sweeps measure behavior count per logical code line. Control is graded but not globally linear: effects can saturate, reverse, or become undefined. At the smallest coefficient reaching a +0.10 validation target, 14B held-out density changes per logical line are $+0.155$ comments ($n = 100$; pass -0.07) and $+0.274$ defensive constructs ($n = 74$; pass -0.15); intervals are in Appendix Table 6. Logging never reaches the target and leaves one nondegenerate test pair. Intensity requires coverage, validity, and correctness.

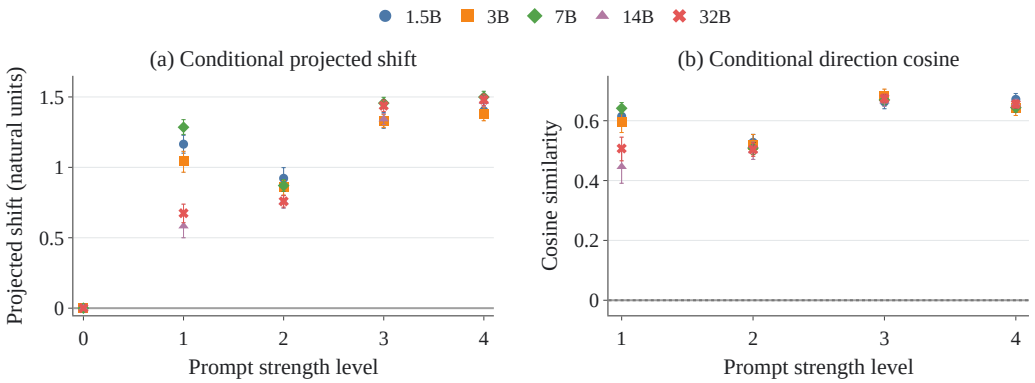


Figure 5: Conditional prompt-vector alignment on held-out Python tasks. Error bars are 95% task-bootstrap intervals. Compliant level-4 pairs shift by roughly 1.4 natural units at cosine ≈ 0.65; level 0 is the definitional origin, and endpoint/cohort sizes are in Appendix Table 12.

7.2 Vector sums induce multiple attributes; additivity remains uncertain

For K requirements, Equation 4 pre-sums the scaled vectors into one delta and uses one hook. On held-out 14B *tutorial-C++*, the joint intervention increases C++, comments, and defensive programming (Appendix Table 3), but only 68 behavior cases are extractable and the intervals include meaningful subadditivity. Other profiles lose malformed cases, especially Python *debug-mode*. Linear vector addition does not imply additive output behavior.

7.3 Compliant prompted outputs align with the extracted direction

For held-out Python outputs under a five-level comment-instruction ladder, we project each task-paired shift from level 0 onto the training direction at $L^\star$ (Appendix F). Only comment-free baselines and compliant targets enter, so cohorts differ. Endpoint projections are 1.38–1.50 natural units, versus $|\alpha| \geq 4$ for 14 of 18 primary selections at other possible layers. Random-direction projections remain near zero, but Figure 5 is not a dose response or evidence of behavioral equivalence; 0.5B lacks baseline pairs.

7.4 A calibrated case study reduces repeated requirement tokens

On 100 held-out 7B Python tasks, a validation-screened *concise-Python* profile compares prompted requirements with a neutral prompt plus one pre-summed residual addition per decode call (Appendix G). Both reach 100% compliance; steering saves 19 input tokens/request (17.6%) but emits 3.37 more (5.1%), for 9.0% fewer total tokens. Python pass is 0.43 for steering versus 0.44 for prompting ($\Delta = -0.01$, 95% CI [-0.08, +0.06]), not evidence of equivalence. The neutral arm reaches 61%; no positive *debug-mode* strength qualifies.

8 Discussion

Across six model sizes, concept vectors reveal structure in coding behaviors and alter held-out outputs, supporting their use as empirical controls. Extraction remains conditioned on prompts and observed compliance, however, and behavioral specificity requires matched-norm causal controls. Stronger steering can impair correctness, so detector gains alone do not establish useful control. Evidence from one model family, short programs, detectors without human validation, and Python execution tests limits generalization. Prompting remains the practical baseline; the main sweep lacks a matched comparison, and the single-profile cost result establishes neither speedup nor correctness equivalence. Reusable prompt prefixes could also change its token advantage. Reliable deployment therefore requires task-specific calibration and joint evaluation of behavior and executable correctness.

AI USE STATEMENT

OpenAI Codex coding agents assisted with software implementation and debugging, experiment orchestration, analysis and visualization code, and manuscript drafting and editing. Retained AI-assisted code, figures, and prose were reviewed by the authors. Numerical claims were checked against frozen experiment artifacts, generated tables and figures were audited, and the implementation was tested before inclusion. Generative models used as experimental subjects or evaluators are described separately in the paper. The authors take responsibility for the final text, claims, code, and artifacts.

ETHICS STATEMENT

This work uses public programming benchmarks and does not involve human subjects or private user data. Activation steering can suppress as well as induce coding behaviors; in particular, reducing defensive checks or changing language against user intent could be harmful if deployed without validation. We therefore report correctness and degeneration alongside behavioral control and do not present the extracted directions as guarantees of software quality or security.

REPRODUCIBILITY STATEMENT

Section 3 specifies the extraction and intervention protocol. The appendices in this manuscript provide prompt templates, detector definitions, sweep grids, validation-selected settings, additional numerical results, and the cost protocol.

REFERENCES

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. *arXiv preprint arXiv:2406.11717*, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and extensible approach to benchmarking neural code generation. *arXiv preprint arXiv:2208.08227*, 2022.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. In *International Conference on Learning Representations*, 2025.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in Neural Information Processing Systems*, 36, 2023.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, 2021.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, 2013.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Md Mahbubur Rahman, Arjun Guha, and Harshitha Menon. Steering code LLMs with activation directions for language and library control. *arXiv preprint arXiv:2603.23629*, 2026.

Nishant Subramani, Nivedita Suresh, and Matthew E. Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, 2022.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *International Conference on Learning Representations*, 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 preprint arXiv:2308.10248*, 2023.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Composition Details

Table 3: Held-out 14B *tutorial-C++* composition. Joint and interaction columns show mean change with 95% task-bootstrap intervals. "Sum singles" is the sum of the separately measured component effects. C++ is a binary detection-rate change over all 100 tasks; comments and defensive programming are density changes over 68 jointly extractable outputs, so rows are not directly comparable.

Outcome	Joint Δ	Sum singles	Interaction	n
C++	+0.190 [+0.110,+0.270]	+0.150	+0.040 [-0.030,+0.110]	100
Comments	+0.226 [+0.122,+0.326]	+0.301	-0.075 [-0.195,+0.039]	68
Defensive	+0.289 [+0.217,+0.369]	+0.288	+0.001 [-0.056,+0.059]	68

B Prompt and Detector Details

B.1 Factorial extraction prompt

Every prompt uses the following frame after substituting the task and language:

```
You are an expert programmer. Write a [language] solution
to the following task.
Task: [natural-language task]
Requirements: Write the solution in [language]. [comments
clause] [logging clause] [defensive clause]
Return only the solution code inside a single [language]
fenced code block.
```

The binary clauses are:

Axis	Positive	Negative
Comments	Include a docstring/header comment and explanatory inline comments.	Do not include comments or docstrings.
Logging	Emit diagnostic logging/print statements at key steps.	Do not add logging or prints.
Defensive	Apply appropriate input validation and error handling where it is genuinely warranted.	Write the direct solution without extra validation.

The behavior-steering prompt omits all three behavior clauses and fixes Python plus the expected function signature. The language-steering prompt omits both language and behavior requirements. The model's own instruct chat template is applied in all generation and teacher-forcing stages. Steering begins only after the first sampled token; prefill and first-token logits are untouched, and this token-position convention is not ablated.

B.2 Static behavior detectors

Python comments are found with `tokenize`; docstrings and logical code lines use the AST. Logging includes `print` and calls such as `logger.debug`. Defensive programming is one binary axis formed by the union of try/except, assertions, comparisons with `None`, explicit raises, recognized validation exceptions, and guard clauses. Java/C++ use regular expressions over extracted code for comment markers and conservative syntax patterns for logging and defensive constructs; markers inside string literals can therefore be false positives. These detectors measure presence or volume, not quality.

B.3 Programming-language judge

The secondary language detector is Qwen2.5-Coder-14B-Instruct, so it also judges outputs from that same checkpoint in the 14B row. It sees only generated code (truncated to 4,000 characters), not the requested target or source model, and is asked to reply with exactly one of `python`, `java`, `cpp`, or `unknown`. Decoding is greedy for at most four tokens; the first whole-word language mention is normalized, otherwise the verdict is `unknown`. A language-experiment row is one generated output for a language-axis steering cell. Across 27600 such rows from all six checkpoints, judge/rule agreement is 84.6% and the judge returns `unknown` on 8.1%. This is a robustness view, not human ground truth.

Table 4: Cosine between the full and comment-token-masked comment vectors at $L^\star$. Literal comment/docstring spans are removed before pooling for the masked variant.

Model	$L^\star$	Cosine
0.5B	16	0.454
1.5B	19	0.663
3B	24	0.561
7B	19	0.656
14B	32	0.644
32B	43	0.508

C Additional Numerical Results

Layer indices are zero-based: layer 0 is the first decoder block. The dense continuous-control sweeps at $L^\star$ use $\alpha \in \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6, \pm 8\}$ for the three coding behaviors; language sweeps use only the nonnegative values of this grid.

At 7B all three axes use (L21, $\alpha = 8$), yet comment and logging vectors both change comments by $+0.57$; at 32B all use (L48, $\alpha = 4$), where logging changes comments by $+0.24$ versus logging by $+0.21$. These same-setting contrasts are informative but not norm-matched because natural vector

Table 5: Matched-pair counts for full behavior vectors at each shared layer. Every displayed vector uses the paired estimator; no class-mean fallback is used in the headline analyses.

Model	Comments	Logging	Defensive
0.5B	39	93	287
1.5B	403	462	494
3B	357	658	610
7B	880	941	717
14B	885	1071	806
32B	1185	1154	886

Table 6: Held-out 14B continuous-control operating points selected on validation. Behavior change is density per logical code line; pass change is paired against the unsteered output over all 100 tasks. Behavior n counts nondegenerate extracted pairs, making the logging estimate unusable.

Axis	α	Behavior Δ [95% CI]	Behavior n	Pass Δ [95% CI]
Comments	1	+0.155 [+0.110,+0.208]	100	-0.07 [-0.12,-0.03]
Defensive	6	+0.274 [+0.207,+0.338]	74	-0.15 [-0.24,-0.06]
Logging	8	+0.250 [+0.250,+0.250]	1	-0.53 [-0.64,-0.43]

norms differ. At 14B, logging steering changes comments by $+0.36$ versus logging by $+0.08$. This motivates the specificity limitation in Section 8.

The maximum task variance share over all layers ranges from 66.3% to 74.9% across the five estimable checkpoints; Table 9 instead reports each checkpoint's shared layer. At 14B, the shared-layer direct language-by-comment interaction is 0.005, while mean alignment cosine is 0.914 and global analogy top-1 is 0.959. Analogy evaluation includes 582 ordered source-target cases at this checkpoint.

D ALL-LAYER STEERING

Every decoder block is tested, using $\alpha \in \{1, 4\}$ for comments and $\alpha \in \{4, 8\}$ for C++ at each checkpoint.

At each layer, the procedure retains the larger mean effect among the two tested coefficients after excluding settings with degeneration above 0.5; the shaded set contains layers whose pointwise interval overlaps the selected maximum's interval. It is therefore post-selection and descriptive. For 32B comments, this rule selects layer 61 at $\alpha = 4$: behavior density changes by $+0.364$ [0.269, 0.465] with 0.18 degenerate outputs. The primary three-layer sweep instead selects layer 48 at $\alpha = 4$, where the binary detector changes by $+0.71$ and Python pass is zero; the outcomes and selection rules are intentionally different.

Across checkpoints, best layers do not collapse to a single normalized depth. Strong comment steering frequently becomes degenerate over broad layer ranges, while the weaker coefficient can still induce a modest change. For language steering, the judge and rule detector usually identify the same late-layer region but can choose different maxima within it.

Table 7: Validation-selected behavior-steering settings, reported as layer and α. These settings optimize binary detection; $\alpha = 8$ is the largest tested positive coefficient. Settings are frozen before test evaluation in Table 2.

Model	Comments	Logging	Defensive
0.5B	L12, 0.5	L18, 2	L12, 8
1.5B	L21, 2	L14, -4	L14, 8
3B	L27, 2	L18, -4	L18, 8
7B	L21, 8	L21, 8	L21, 8
14B	L36, 4	L36, 8	L24, 8
32B	L48, 4	L48, 4	L48, 4

Table 8: Held-out target-by-detector changes at selected settings. Entries are mean binary-rate changes versus each task's baseline; bold marks the target. It has the largest absolute change in 14 of 18 rows and the largest signed change in 13; exceptions preclude a general specificity claim.

Model	Steered axis	Comments	Logging	Defensive
0.5B	Comments	**+0.02**	+0.04	+0.06
	Logging	+0.02	**-0.05**	+0.01
	Defensive	+0.01	-0.26	**+0.53**
1.5B	Comments	**+0.19**	+0.09	+0.02
	Logging	-0.01	**+0.19**	+0.00
	Defensive	+0.33	+0.48	**+0.61**
3B	Comments	**+0.18**	+0.09	+0.00
	Logging	+0.06	**+0.23**	-0.02
	Defensive	+0.10	+0.10	**+0.71**
7B	Comments	**+0.57**	-0.02	-0.02
	Logging	+0.57	**+0.10**	-0.02
	Defensive	+0.19	+0.03	**+0.37**
14B	Comments	**+0.51**	+0.00	-0.02
	Logging	+0.36	**+0.08**	-0.04
	Defensive	+0.35	+0.00	**+0.74**
32B	Comments	**+0.71**	+0.00	-0.05
	Logging	+0.24	**+0.21**	+0.01
	Defensive	+0.19	+0.00	**+0.28**

Table 9: Structure analysis at each shared layer. Variance entries are Type-I shares of the nuisance-adjusted total; L×C is the direct language-by-comment term. Alignment is mean pairwise cosine among language-specific comment offsets.

Model	Tasks	Task	Lang.	Comment	L×C	Alignment	Analogy
0.5B	0	–	–	–	–	–	–
1.5B	56	0.506	0.130	0.061	0.016	0.784	0.714
3B	45	0.642	0.084	0.046	0.006	0.899	0.856
7B	99	0.607	0.112	0.065	0.006	0.910	0.926
14B	97	0.624	0.082	0.080	0.005	0.914	0.959
32B	100	0.653	0.041	0.086	0.005	0.920	0.975

Table 10: Type-III marginal shares from the same pre-specified-layer factorial fits. Unlike the Type-I partition in Table 9, these terms need not sum to one. Neither version divides by factor degrees of freedom.

Model	Task	Language	Comment	L×C
0.5B	–	–	–	–
1.5B	0.427	0.096	0.044	0.014
3B	0.548	0.065	0.034	0.006
7B	0.578	0.104	0.060	0.006
14B	0.598	0.075	0.074	0.005
32B	0.652	0.041	0.086	0.005

Table 11: Shared-layer Type-I share per factor degree of freedom. Task df equals complete tasks minus one; language and L×C have 2 df and comment has 1. This diagnostic reverses the raw-share ordering at 14B and shows why Figure 3 is a design decomposition, not a claim that one task degree of freedom is stronger than one behavior degree of freedom.

Model	Task df	Task/df	Lang./df	Comment/df	L×C/df
0.5B	–	–	–	–	–
1.5B	55	0.0092	0.0650	0.0612	0.0078
3B	44	0.0146	0.0420	0.0465	0.0032
7B	98	0.0062	0.0561	0.0654	0.0030
14B	96	0.0065	0.0410	0.0803	0.0024
32B	99	0.0066	0.0205	0.0859	0.0024

Table 12: Held-out prompt alignment. Level-0 compliance is the fraction of no-comment outputs that actually contain no comments/docstrings. Endpoint columns report level 4 with 95% task-bootstrap intervals. Projected shift is measured in extraction-direction natural units; it is not a behaviorally equivalent steering coefficient. L4 n is the endpoint sample size; pair range covers levels 1–4. Cohorts differ by level, and plotted level 0 is a definitional origin without an interval.

Model	L0 compliance	Projected shift	Cosine	L4 n	Pair n range
0.5B	0.00	–	–	–	–
1.5B	0.85	1.401 [1.350,1.452]	0.671 [0.652,0.691]	85	80–85
3B	0.60	1.379 [1.330,1.427]	0.642 [0.617,0.667]	60	58–60
7B	1.00	1.499 [1.459,1.540]	0.644 [0.629,0.658]	100	95–100
14B	0.97	1.467 [1.430,1.504]	0.658 [0.643,0.672]	97	17–97
32B	1.00	1.480 [1.439,1.520]	0.655 [0.642,0.668]	100	31–100

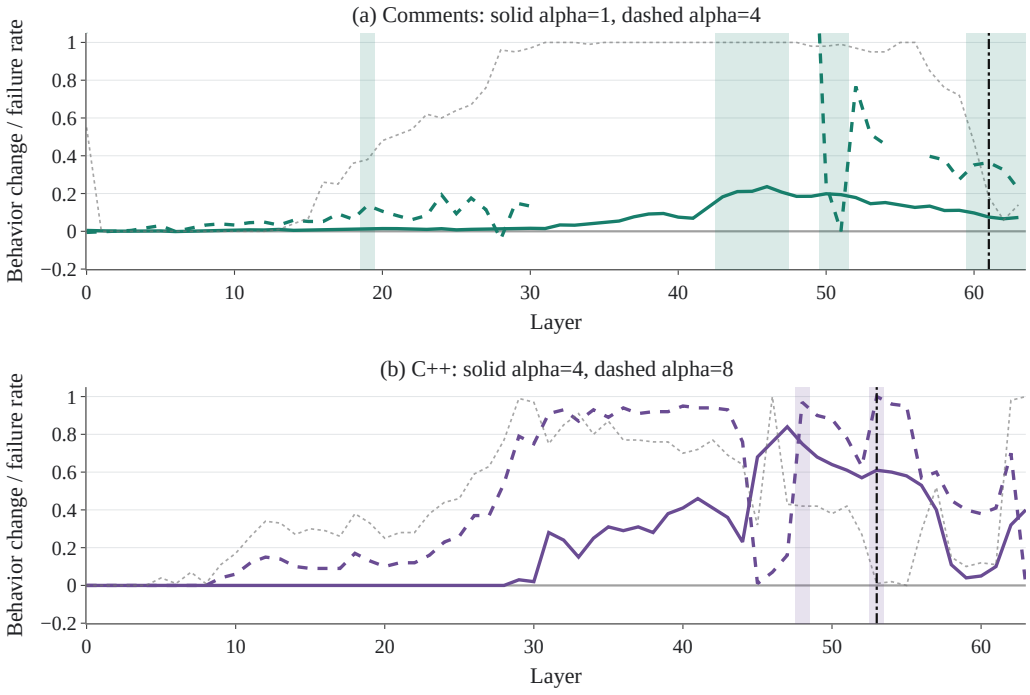


Figure 6: Every-layer sweep for 32B. The dash-dotted vertical line marks the degeneration-guarded empirical maximum, and lightly shaded regions mark the pointwise-CI-overlap set. Dashed colored curves use the stronger coefficient; dotted gray curves show its degenerate-output rate. Comment control concentrates late but has a broad, disconnected set. C++ uses the 14B language judge and peaks at layer 53 with overlap set $\{48, 53\}$. This is descriptive, not a simultaneous confidence set.

E Token and Snippet Controls

Table 13: Representative tokens among each 14B vector's top ten final-RMSNorm-folded unembedding projections; parenthesized values are ranks. Whitespace is stripped for display.

Direction	Representative top-ranked tokens
Comments	`the` (1), `TODO` (3), `NOTE` (4), `The` (6)
Logging	`print` (1), `.print` (3), `.debug` (4), `.Println` (6)
Defensive	`valid` (1), `IllegalArgumentException` (2), `validate` (4), `stdexcept` (5), `ValueError` (7)
Python	`isinstance` (1), `ValueError` (2), `elif` (4), `(None` (5)
Java	`.java` (1), `java` (2), `boolean` (5), `=null` (6)
C++	`(std` (1), `std` (3), `(nullptr` (6), `#include` (9)

At 14B, the top cosine of the 30 seeded random controls lies in the same rough numerical range as some concept-vector maxima, but the token lists are semantically incoherent. One example begins with `azi`, `excludes`, an Arabic fragment, `_VERTEX`, and a Chinese character. This motivates our rank/identity criterion rather than a maximum-cosine threshold.

For each real direction, the snippet analysis ranks code-body windows by token-level cosine and compares the real p95 score with its random counterpart. Across all model/vector cells, the ratio ranges from 0.37 to 13.67. Most behavior vectors clearly exceed random, but 0.5B defensive, 3B C++, and 32B C++ do not. These windows are extracted from the same training generations used to estimate the vectors and are therefore qualitative, in-sample evidence only.

F Prompt-Alignment Protocol

The five comment instructions are:

1. Do not write comments or docstrings.
2. Only write comments or docstrings if absolutely necessary for non-obvious code.
3. Write concise comments for non-obvious logic, but avoid straightforward-line explanations.
4. Write helpful comments and docstrings explaining the approach and important choices.
5. Write detailed comments and docstrings explaining the solution, edge cases, and reasoning.

Each level contains all 100 validation tasks. For level $q > 0$, a pair is included only when its level-0 output has zero detected comments/docstrings and its level-q output has at least one. The mean prompt-induced shift Δ_q is summarized in natural units by $\langle \Delta_q, \boldsymbol{v}_{\text{comment}} \rangle / s$, where s is the natural norm from extraction. Five seeded random directions provide a cosine control. Level 0 is the definitional origin rather than an estimated point, and no behavioral equivalence to direct steering is tested.

G Cost Experiment Details

Profile names are fixed shorthand, not additional learned objects. The 14B composition run at layer 32 defines *tutorial-C++* as $(+2\boldsymbol{v}_{\text{C++}}, +1\boldsymbol{v}_{\text{comments}}, +6\boldsymbol{v}_{\text{defensive}})$ and *debug-mode* as $(+8\boldsymbol{v}_{\text{logging}}, +6\boldsymbol{v}_{\text{defensive}})$. The separate 7B cost calibration at layer 19 starts from concise-Python $(-1, -6)$ on comments/logging and debug-mode $(+6, +6)$ on logging/defensive, then multiplies both coefficients by the candidate β.

The prompt-only and steering arms share the same tasks, decoding parameters, and chat template. The prompt arm appends the literal requirements. For concise-Python, these are "Do not include comments or docstrings." and "Do not add logging or prints." For debug-mode, they are the positive logging and defensive clauses listed in Appendix B. The steering arm uses the neutral task prompt and one pre-summed delta. The neutral arm uses that same prompt without a residual intervention.

Table 14: Held-out 7B concise-Python comparison ($n = 100$). Every rate uses all 100 requests; extraction is 1.00 in every arm, so compliance can remain 1.00 when the separate syntax parse rate is 0.99. Input counts are exact after chat templating; total/request is input plus generated tokens.

Arm	Input/request	Output/request	Total/request	Compliance	Parse	Pass
Neutral	88.76	96.49	185.25	0.61	1.00	0.44
Prompt	107.76	66.31	174.07	1.00	1.00	0.44
Joint steering	88.76	69.68	158.44	1.00	0.99	0.43

The concise-Python base profile is $-1\boldsymbol{v}_{\text{comments}} - 6\boldsymbol{v}_{\text{logging}}$ at layer 19; multiplying by the selected $\beta = 0.5$ gives injected coefficients $(-0.5, -3)$. Input tokens are counted after chat templating. The fixed probe generates exactly eight tokens and repeats each batch five times; medians are summed across batches. Full generation runs to EOS and records prompt/output tokens, CUDA and wall time, peak incremental memory, parse and extraction rates, detector compliance, and Python tests. Validation is capped at 600 new tokens; held-out confirmation is capped at 1,200.

Validation evaluates $\beta \in \{0, 0.125, 0.25, 0.375, 0.5, 0.75, 1\}$ for each profile. A candidate is eligible when parse and extraction rates are at least 0.95 and pass rate is no more than 0.05 below the stronger of prompt and neutral. The eligible point with highest joint compliance is frozen for test. Concise-Python selects $\beta = 0.5$; debug-mode selects $\beta = 0$, meaning no positive steering strength among the six tested values meets the pre-specified point-estimate constraints on validation. The 0.05 correctness screen is a point-estimate rule rather than an equivalence or noninferiority test.

Across two profiles and 200 requests, the fixed-eight-token probe reports 20.2% fewer input tokens and 5.6% lower aggregate CUDA time under steering. Per-batch reductions range from 2.2% to 8.8%; hook overhead against the identical neutral prompt ranges from -0.9% to 1.2%. These timings are hardware-specific and do not establish an end-to-end speedup; raw full-generation timings favored steering (17.26 versus 23.51 CUDA s) but were single runs with different output lengths and batch schedules, so we do not interpret them. Because requirements follow task-specific text, ordinary prefix caching does not reuse them across distinct tasks; moving them into a shared system prefix would change that comparison. For debug-mode at full strength, pass rate falls to zero and 80% of outputs hit the length cap or fail extraction.