TASK IDENTIFICATION FEATURES CAUSALLY DRIVE EXECUTION FEATURES IN IN-CONTEXT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models learn patterns from demonstrations in context, but how information at the demonstrations comes to drive the output has not been established. An identification feature for a pattern forms where the pattern is demonstrated (we call these "label tokens") and drives an execution feature at subsequent locations where the model must continue the pattern ("cue tokens"). We establish two results: 1) an identification and an execution feature together mediate this effect on behavior, and 2) a single affine map shared across patterns relates each identification feature to its execution feature. In simple patterns generated by input-output tasks in GPT-J and Qwen2.5-7B, where the execution feature is the task's function vector, steering the identification feature raises cue-token alignment with the execution feature more than with a generic one, ablating it lowers that alignment, and ablating the execution feature removes the accuracy gained from identification steering. A single affine map fitted on training tasks predicts held-out tasks' execution features from their identification features (R^2 of 0.60 and 0.53, versus below zero for a mean predictor). The same pathway appears in a naturalistic setting, in-context learning of coding conventions, with no explicit labels, features at evidence and cue tokens steer and ablate generated style, and identification steering moves the cue-token state toward the execution feature, although the linear map is weaker.

1 INTRODUCTION

Large language models can adapt their behavior from a few demonstrations without updating their parameters, a capability known as in-context learning (ICL) (Brown et al., 2020). A prompt containing "hot : cold" and "big : small" can induce a model to complete "fast :" with "slow." Successful completion requires both identifying a pattern in the preceding text and executing that pattern in generated text. We study whether these two steps correspond to distinct, causally connected representations inside the model.

A simple example is a pattern of input-output pairs that represent simple tasks (e.g. antonyms in the example above). In this setting, Todd et al. (2024) show the existence of function vectors (FVs), extracted from attention-head outputs, that act as execution features and drive the correct pattern for a new input without any demonstrations in context. However, how information in the demonstrations gives rise to these execution features remains understudied. Wang et al. (2023b) show that label words (output words) aggregate information in shallow layers, and later computation reads from them for prediction. Taking this label-site information as our starting point, we additionally show two things: the execution feature mediates the behavioral effect of label-site identification, and one linear map shared across tasks links the two.

Figure 1 illustrates the identification-execution mechanism. Label tokens provide evidence about the task but the final cue token (the token immediately before the model has to continue the pattern) is where the model must apply it. We study patterns from simple input-output tasks, then extend the analysis to more realistic patterns found in conventions or styles of code.

Our findings are:

- **Identification drives execution.** Steering along the identification feature increases residual stream alignment with the execution feature at the cue token and ablating it decreases it. The accuracy (on pattern completion) gained from identification steering depends on both features

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.

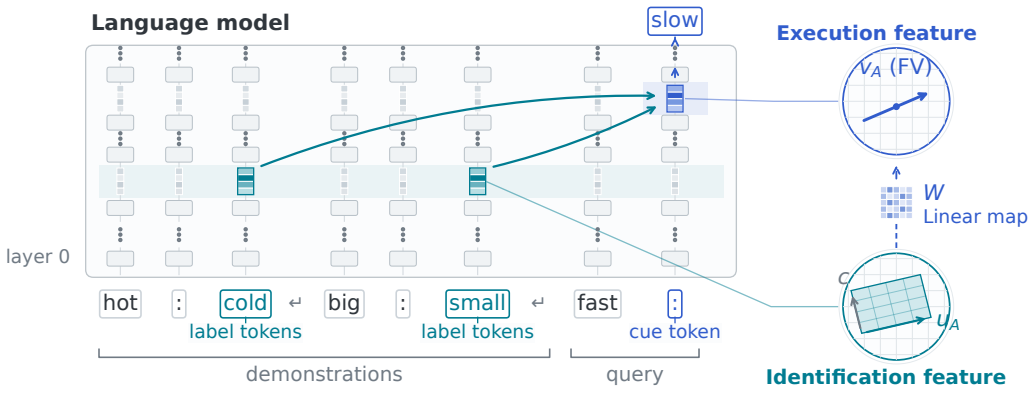

Figure 1: **Shared representations for task identification and execution.** At the demonstration label tokens, the residual stream carries a task-specific *identification feature* u_A (teal), alongside a label direction c shared across tasks. This task identity is carried to the cue token (solid arrows), where an *execution feature* (blue), the function vector v_A in simple tasks, drives the answer. A single map W, fitted across tasks, predicts v_A from the identification feature (dashed). The arrows show causal dependencies tested by steering and ablation. All symbols are defined in Section 4.

together: identification steering produces the gain, and ablating the execution feature removes essentially all of it (100

- **A shared map links the two across tasks.** A single affine map fitted on a training set predicts held-out pattern execution features from their identification features (Section 4.2.2).
- **Both features bidirectionally control behavior.** In simple input-output patterns, we identify two features that each exert bidirectional control: an identification feature at the demonstration label tokens in earlier layers, and an execution feature, the FV (Todd et al., 2024), at the cue token in later layers. Steering along either feature induces the model to follow the pattern, and ablating it suppresses the pattern, while controls leave it largely intact (Sections 4.1.1 and 4.1.3).
- **The pathway extends to coding conventions.** We show the existence of similar identification and execution features on Qwen2.5-7B base imitating patterns in coding conventions from example code. Steering or ablating either feature changes the generated style, and identification steering moves the residual stream alignment with the execution feature (Section 5). The linear map between the two sites, however, is less predictive here (Section 5.3).

The core contribution of this work is a causal pathway for in-context learning, from identification features at demonstration tokens to execution features at the cue token, that is approximately linear across tasks and also appears in coding-style imitation.

2 Related Work

From demonstration labels to pattern or task representations. Label words aggregate demonstration information for later prediction (Wang et al., 2023b) and can act as local task vectors (Zheng et al., 2026), and task or state vectors summarize demonstrations for the query (Hendel et al., 2023; Li et al., 2024; Yang et al., 2025c). These works do not test whether label-site information drives the function vector at the query, whether this mapping is shared across tasks, or whether it holds outside `Q:/A:` prompts; we take their findings as our premise and test these three questions. Encoding–decoding, geometric, multi-phase, and sparse-feature-circuit accounts describe staged processing (Han et al., 2025; Yang et al., 2025b; Cho et al., 2025; Sia et al., 2024; Kharlapenko et al., 2025; Hu et al., 2025). Our contribution is these two results, mediation by the execution feature and a linear map shared across tasks, with function vectors as one instance of the execution feature.

Function vectors and attention heads. Function vectors induce task execution (Todd et al., 2024; Wang et al., 2026; Hendel et al., 2023) and, with induction heads, drive few-shot in-context learning (Olsson et al., 2022; Yin & Steinhardt, 2025). We use FVs as the execution feature for patterns generated by simple tasks.

Stages of ICL and behavioral steering. Task recognition versus learning (Min et al., 2022; Pan et al., 2023; Yang et al., 2025a) and compression versus expression (Jiang et al., 2025) are other distinctions of in-context learning, while ours concerns token sites. Activation steering (Turner et al., 2023; Rimsky et al., 2024; Brumley et al., 2024), including of style (Liu et al., 2024) and code (Rahman et al., 2026), motivates our coding extension.

3 METHODOLOGY

We study input-output patterns that are generated by simple tasks from the function-vector framework of Todd et al. (2024) (e.g., antonyms, translation, morphology) in GPT-J-6B (Wang & Komatsuzaki, 2021) and Qwen2.5-7B-Instruct (Qwen Team, 2024), followed by a naturalistic coding-style setting in Qwen2.5-7B base.

3.1 SIMPLE-TASK SETUP AND DEFINITIONS

Tasks and prompts. Following Todd et al. (2024), prompts are formed by demonstrations of pairs of input lines (`Q: hot`) with output or label lines (`A: cold`) and end with a query input (`Q: fast`) followed by a blank answer line (`A:`), see Appendix B for a full example. The colon of final `A:` is the *cue token* in this setting (the token before which the model must complete the pattern). Both models receive plain-text prompts and we sample generations using a temperature of 1. Since patterns are generated by tasks, we will use the words "tasks" and "patterns" interchangeably in the results of this section.

Task selection. From 117 candidate tasks (Appendix A), we only use ones that the models can successfully learn in context (at least 30% completion accuracy given 6 demonstrations in context). This leaves us with 69 tasks for GPT-J and 96 for Qwen. We divide these tasks into train and test sets where appropriate to show generalization of claims.

Features. A *feature* is a direction s in the residual stream, with unit direction $\hat{s} = s/\|s\|$. Each task's execution feature is its FV, v_A (Section 4.1.1), and its identification feature is a task-specific direction u_A at the label tokens, which sits alongside a label direction c shared across tasks (see Section 4.1.3 for the detailed definitions). We reserve *feature* for these task-specific directions and call shared components such as *c directions*. By alignment with a feature we mean the signed cosine of the residual stream (at a certain layer) with it.

Steering and ablation. Let $z_{\ell,t} \in \mathbb{R}^{d_{\text{model}}}$ be a residual-stream activation at the output of (zero-indexed) transformer layer ℓ at token t. Steering adds a vector s with strength α, $z_{\ell,t} \leftarrow z_{\ell,t} + \alpha s$ (Turner et al., 2023). Ablating a feature s replaces the activation's component along it: $z_{\ell,t} \leftarrow z_{\ell,t} - (\hat{s}^\top z_{\ell,t})\hat{s} + (\hat{s}^\top z^{\text{ref}}_{\ell,t})\hat{s}$, with $z^{\text{ref}}_{\ell,t} = 0$ (zero ablation) or a reference mean activation (mean ablation) (Geiger et al., 2024; Zhang & Nanda, 2024; Arditi et al., 2024).

3.2 CODING-STYLE SETUP AND FEATURE CONSTRUCTION

As a second setting, we turn to more realistic patterns in context, namely coding styles.

Coding conventions and families. A *coding convention* (Allamanis et al., 2014) is a stylistic choice that does not change what a program does, such as double versus single quotes around strings, `camelCase` versus `snake_case` identifiers, or four- versus two-space indentation. A pair of such choices together is called a *family* (e.g. the quote family is composed of the choice between single vs double quotes). For each family we build 200 programming tasks, and each task is written as a pair of documents, complete programs that implement the solution to the same task but differ in the convention used. We study 60 candidate families across nine languages (Table 4).

Prompts. Within a document, every place where the convention must be chosen is a *scored choice* (e.g. in the quote family, every string literal is one). A prompt is composed of a programming-task description, a language header, and the document's code, which is cut off immediately before a scored choice. The last prompt token is the cue token, and k is the number of scored choices already visible

in the code (i.e. k in-context demonstrations). As a 3-shot example, this JavaScript prefix shows three double-quoted strings and ends at the cue token before the fourth:

```
const PENDING = "pending";
const RUNNING = "running";
const SUSPENDED = "suspended";
const ERROR =
```

Its paired prompt is identical except that it uses single quotes (Figure 19 shows both prompts in full). Note that the prompts don't explicitly request any coding style; the model can only follow the convention by inferring it from the code provided as context.

Success and family selection. The model continues the prompt, and its choice at the cue token is scored. *Target-style success* requires the continuation to use the target convention and to be plausible, valid code for the task (determined by a judge model). Success grows with the number of visible choices, from 29.6% at $k = 0$ to 76.3% at $k = 4$ across the 53 retained families (Figure 18 in Appendix E). As for the simple tasks, we filter families the model can learn in context (at least 30% success at $k = 4$ for both conventions, which retains 54 of the 60; one further family is excluded because its alternative rendering cannot be produced reliably, leaving 53). See Appendix E for details on scoring, filtering, and document splits.

Identification and execution features. We look for the execution feature at the cue tokens and the identification feature at *evidence tokens*, which are tokens in the code that express the convention (e.g. the quote tokens in the example above). For family f and site $q \in \{\mathrm{id}, \mathrm{exec}\}$ (site meaning layer and token type), let $\mu^q_{f,\mathrm{A}}$ and $\mu^q_{f,\mathrm{B}}$ be the mean activations at those sites on prompts demonstrating convention A or B (where the mean is computed over successfully completed prompts). We find suitable identification ($q = \mathrm{id}$) or execution ($q = \mathrm{exec}$) features using *mean activation differences* $d^q_f = \mu^q_{f,\mathrm{A}} - \mu^q_{f,\mathrm{B}}$. Adding $+d^q_f$ or $-d^q_f$ pushes toward convention A or B, an approach related to in-context vectors and contrastive activation addition (Liu et al., 2024; Rimsky et al., 2024).

4 Results: simple input-output tasks

4.1 Identification and execution features in simple tasks

4.1.1 Function vectors are execution features

Todd et al. (2024) show that function vectors (FVs) induce task execution (pattern completion) when added to a model's activations at the token just before the model must complete the pattern, so in this setting, we use the FV as the execution feature. For task A, let $\bar{h}_{A,h} \in \mathbb{R}^{d_{\mathrm{model}}}$ be attention head h's mean output at the final cue token of ten-shot prompts (prompts with ten demonstrations before the final query). Given a selected set of heads H, we construct the FV as follows:

$$v_A = \sum_{h \in H} \bar{h}_{A,h}.$$

Adapting the sparse head-selection method of Hu et al. (2025), we optimize H jointly over training tasks, obtaining 37 heads for GPT-J and 140 for Qwen. Within each model, this head set is shared across tasks. The head means are specific to the task, and hence v_A is also task-specific. The final head sets are fitted and then evaluated using 80/20 train-test splits to check these heads generalize. This is mostly reproducing prior results and more details are provided in Appendix B.

We add v_A at the final cue token of zero-shot prompts, at the output of each of the 28 layers, and score accuracy on 50 queries per task. Function-vector steering improves zero-shot execution across a broad range of injection depths (Appendix B, Figure 7). At the layer selected on training tasks, layer 1 in GPT-J, held-out accuracy is 62.9%, compared with an unsteered baseline of 9.0%, and at layer 9 in Qwen it is 64.4%, compared with 5.8%. After a certain depth, steering effectiveness decreases, as it becomes "too late" to induce the task.

Thus attention heads for FVs are generalizable and the steering ability of FVs at the cue tokens establishes them as execution features.

4.1.2 IDENTIFICATION INFORMATION IS AVAILABLE AT DEMONSTRATION LABELS

To find an identification feature, we investigate candidate locations (i.e. token type and layer depth) in GPT-J by asking where the residual stream contains linearly decodable information about the execution feature, v_A. We define per-prompt FVs as the activations of the attention heads in H at the final cue token of a single prompt. We fit ridge regressions from residual stream at each location (i.e. token and layer) to their respective per-prompt FVs. Regularization is done by cross-validation over training tasks and evaluating on held-out tasks. We find that FV-relevant information is available at label tokens before it even becomes comparably decodable at the cue tokens (Figure 2), which suggests that we should look here for identification features. For example, at layer 6, the first demonstration's label token yields a mean task-FV R^2 of 0.46, versus -0.02 at its cue; similarly, the sixth label and cue yield 0.58 and 0.53, respectively. More details are given in Appendix C.

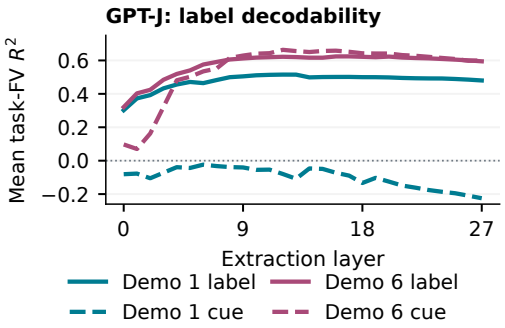

Figure 2: **Function vectors are decodable at preceding label-tokens.** R^2 when linearly decoding FV at various positions on 14 held-out tasks (GPT-J). Solid: labels. dashed: cues. (See Appendix C for more details)

Decodability alone does not establish causality though, so we next construct features from label-token activations and intervene on them.

4.1.3 BIDIRECTIONAL CAUSAL CONTROL USING IDENTIFICATION AND EXECUTION FEATURES

The identification feature. For task A, let $m_A(\ell)$ be the mean residual activation at the final label token in ten-shot prompts at layer ℓ. We generate dummy n-shot prompts, which are prompts with all the demonstration labels replaced with a filler token like _ (see Appendix B for an example), so the model can no longer learn the right task in context. We steer using $m_A(\ell)$ at the dummy labels (_), sweeping over ℓ, to see at which layers task completion peaks. We find the best results at layers 5–7 for GPT-J and 11–13 for Qwen (Appendix C), which we define as the best layers, B. Since these activations share a significant common component (Appendix C), we split them into a label direction shared across tasks and a task-specific component. We define

$$g_\ell = \tfrac{1}{|\mathcal{T}|}\sum_{A\in\mathcal{T}} m_A(\ell), \qquad \hat{g}_\ell = g_\ell/\|g_\ell\|, \qquad c = \tfrac{1}{|B|}\sum_{\ell\in B} g_\ell,$$
$$u_A = \tfrac{1}{|B|}\sum_{\ell\in B}\left[m_A(\ell) - \hat{g}_\ell\hat{g}_\ell^\top m_A(\ell)\right].$$

Here $\mathcal{T}$ is each model's task pool from training tasks only. We call c the label direction and u_A the identification feature of task A. After removing the label direction, identification features of different tasks are nearly orthogonal (mean pairwise cosine -0.00 and -0.01, against 0.73 and 0.50 for raw label means; Appendix C). The identification features of all tasks span the identification subspace $S^{\text{id}} = \text{span}\{u_A : A \in \mathcal{T}\}$. As we show below, inducing the model to do a task works best with the identification feature together with the label direction, while stopping the model from completing the task requires only the identification feature.

Steering along the identification feature. Using dummy six-shot prompts, we steer at the dummy labels, _, with $\alpha(c+u_A)$ at layer 6 in GPT-J and layer 12 in Qwen (chosen by one-shot layer sweeps, Appendix C) and measure model accuracy. At $\alpha = 2$, accuracy rises from approximately zero to 38.1% and 54.3% in GPT-J and Qwen respectively; the biggest improvement is at $\alpha = 2$ for Qwen and at $\alpha = 4$ for GPT-J (44.2%), recovering most of the accuracy on real six-shot prompts (Figure 4). Similarly for one-shot prompts, accuracy rises at $\alpha = 2$ from approximately zero to 11.2% and 54.0%, against real one-shot accuracy of 20.8% and 47.6% in GPT-J and Qwen respectively (Appendix C).

We have now shown that both features can steer the model (FV steering in Figure 7 in Appendix B and identification steering in Figure 4), but to conclude they are the true bottlenecks for models to learn these task patterns in context, we must show ablation works.

Ablation of the identification feature. Taking six-shot prompts, we remove projections along the identification feature (zero ablation of $\hat{u}_A$) at every label token at all layers and see if the model can still learn the task. Accuracy falls from 62.9% to 12.0% in GPT-J and from 79.8% to 21.1% in Qwen, while the same operation with another task's identification feature, $\hat{u}_{A'}$, leaves 61.4% and 74.8% (Figure 3, identification groups). We also study mean-ablation to get similar conclusions (exact results can be found in Appendix C).

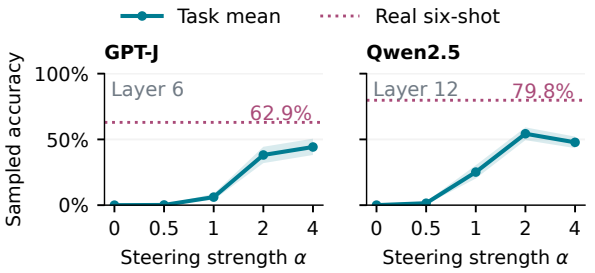

Figure 4: **Steering along the identification feature at label tokens drives pattern completion.** Dotted: real six-shot baseline. Shading: 95% task-bootstrap intervals.

Execution ablation. Again taking six-shot prompts, we ablate projections along the execution feature, v_A, at the final cue token (layers 9 to 27 in both models). Zero ablation reduces six-shot accuracy from 63.9% to 1.3% for GPT-J and from 79.8% to 5.6% for Qwen. Removing another randomly chosen task's FV ($v_{A'}$) also impairs performance, but less severely (47.6% and 54.3%, Figure 3). We hypothesize that other-task zero ablation also removes a shared component and separating this shared direction isolates task-specific execution information in both models (Appendix C). Mean ablation, which avoids the off-distribution effects of zero ablation (Zhang & Nanda, 2024), is weaker for Qwen (73.9%) because its FVs are dominated by this shared direction; mean-ablating only the task-specific part lowers accuracy to 16.8% and 57.9%.

One-shot controls and more details appear in Appendix C.

4.2 From identification to execution

Having established bidirectional control using the identification and execution features, we next test whether identification steering and ablation respectively increase and decrease downstream residual stream alignment with the execution feature, whether the relationship between the two features is shared across tasks, and whether alignment with the execution feature tracks model behavior.

4.2.1 Identification feature steering drives downstream execution features

Taking the dummy six-shot prompts again, we steer at the label tokens with the identification feature plus the shared label direction ($\alpha(c + u_A)$) at layer 6 in GPT-J and layer 12 in Qwen, and measure alignment at the final cue token, $\cos(z_{\ell,\text{cue}}, v_A)$, where $z_{\ell,\text{cue}}$ is the residual activation taken at layer 13 in GPT-J and layer 24 in Qwen. From $\alpha = 0$ to 2, mean alignment rises from 0.183 to 0.365 in GPT-J and from 0.388 to 0.502 in Qwen (Figure 5). Alignment with the generic FV, the mean $\bar{v} = \frac{1}{|\mathcal{T}|}\sum_{A\in\mathcal{T}} v_A$, also increases, but less so (rise from $\alpha = 0$ to 2: $\Delta\cos = 0.089$ in GPT-J and 0.038 in Qwen). The shift is therefore task-specific. More variations can be found in Appendix D.

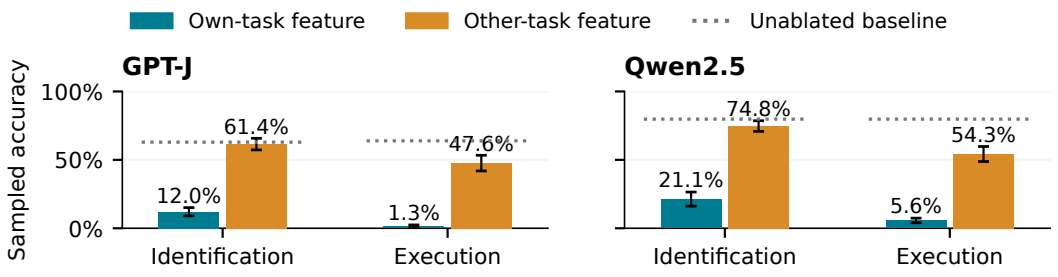

Figure 3: **Zero ablation impairs ICL.** Own-task and other-task interventions, with dotted unablated baselines and 95% task-bootstrap intervals. Mean-ablation comparisons appear in Appendix C.

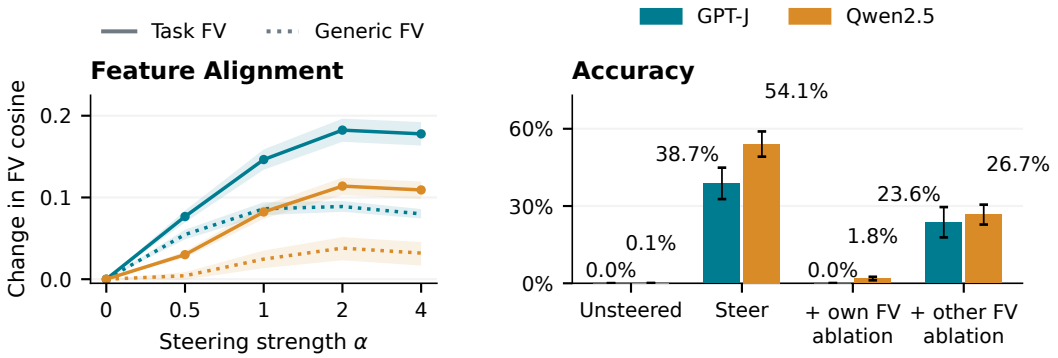

Figure 5: **Identification to execution.** Left: alignment change with steering strength. Right: FV ablation at the cue token removes the steering gain. Shading and error bars: 95% task-bootstrap intervals (Appendix D).

Ablation reduces downstream alignment. On six-shot prompts, we ablate the identification feature at all label tokens at all layers (same as before). As shown in Figure 14, mean ablation reduces the cosine between the cue token and the FV from 0.424 to 0.312 in GPT-J and from 0.498 to 0.451 in Qwen (the other-task control leaves it at 0.423 and 0.498). Zero ablation gives similar effects (Appendix D, Figure 14). While the ablation effects are a bit smaller in Qwen, we see these steering and ablation results as evidence that task-specific execution alignment at the cue token causally depends in part on the identification feature at the labels.

Identification steering acts through the execution feature. We take six-shot dummy prompts and combine steering along the identification feature at the label tokens with ablation of the execution feature (v_A) at the cue token. Steering at the labels alone raises accuracy from near zero to 38.7% in GPT-J and 54.1% in Qwen, reproducing Figure 4. But then ablating the execution feature at the cue token on top reduces this to 0.0% and 1.8%, removing the gains. As a control, we ablate other-task execution features, to leave accuracy at 23.6% and 26.7% (Figure 5). The other-task control's effect scales with the overlap between the execution features (Figure 15, Appendix D). Together with the alignment readouts, these results show that the identification and the execution feature together mediate the behavioral effect (under the tested interventions).

4.2.2 A SHARED MAP RELATES TASK IDENTIFICATION AND EXECUTION FEATURES

We have established the causal relationship between the identification and execution features, so now we turn to the exact relationship between the two and whether that relationship is shared across tasks. To test this, we train a ridge regression from the identification feature u_A to the execution feature, v_A:

$$\widehat{v}_A = \bar{v}_{\text{train}} + W(u_A - \bar{u}_{\text{train}}),$$

where the bars denote means over training tasks. Everything is fitted on training tasks only, including the label direction used to construct u_A and the regularization strength, chosen by leave-one-out validation. Across ten random 80/20 train-test task splits, $R^2 = 0.602 \pm 0.060$ and 0.534 ± 0.070 (mean $\pm$ sample standard deviation) for GPT-J and Qwen respectively. Predicting every held-out v_A as the average training v_A instead gives $R^2 = -0.097 \pm 0.050$ and -0.072 ± 0.022 over the same splits. The ridge map also outperforms a pure rotation (R^2 0.46 on the main split in both models; Appendix D), and the two features are nearly orthogonal (mean cosine 0.05 and 0.07, close to unrelated task pairs at 0.00), so the map transforms one into the other rather than copying it. For more variations of the mapping and training details, see Appendix D. Within each model, identification features therefore linearly predict the execution features of unseen tasks.

4.2.3 EXECUTION ALIGNMENT TRACKS ACCURACY WITHIN TASKS

For each task, we vary the number of in-context demonstrations from zero to six. At each shot count we measure temperature-1 sampled exact-match accuracy and the *execution alignment*, $\cos(z_{\ell,\text{cue}}, v_A)$ (layer 13 in GPT-J, layer 24 in Qwen), averaged over prompts. Across the seven shot counts, the

within-task Spearman correlation between alignment and accuracy has median $\rho = 0.96$ in GPT-J and $\rho = 0.85$ in Qwen. The per-task distributions appear in Figure 17 in Appendix D. Thus execution alignment tracks model accuracy within tasks.

5 RESULTS: IMITATION OF CODING-STYLE

The identification–execution account, derived on `Q:/A:` tasks, predicts the same two sites in code, where there are no explicit labels; the steering, ablation and cue-shift results below confirm this, while the linear map between the sites is weaker.

5.1 STEERING AT IDENTIFICATION AND EXECUTION SITES

We find suitable identification and execution features using mean activation differences, d_f^q. We add $+\alpha d_f^q$ or $-\alpha d_f^q$ to steer toward either convention, evaluating on 40 held-out prompts per family, which were not used to estimate d_f^q. Similar to the simple functions setting, layers the features are extracted from are shared across families and we test for steering ability at the relevant token positions to determine which layer each feature is extracted from. The layer sweep selected layer 7 for identification features (with strength $\alpha = 4$), and layer 23 for execution features (with strength $\alpha = 2$). Details for these choices can be found in Appendix E. Qwen2.5-7B base is the only model used for this extension.

Identification steering. We steer with d_f^{id} at all evidence tokens at layer 7 in a $k = 3$ example prompt demonstrating the *opposite* convention to the desired output. Mean target-style success rises from 4.4% to 60.0% (as a comparison 75.8% was the accuracy for $k = 3$ shot prompts demonstrating the desired convention). Thus, evidence-site steering can substantially override the convention expressed in the visible code.

Execution steering. We steer with d_f^{exec} at the final cue token at layer 23 of a $k = 0$ prompt and see if we can make the model complete the code with the desired style. Mean target-style success rises from 30.1% to 56.0% (for comparison $k = 4$ shot prompts achieve 76.3%). Figure 6 shows both experiments.

Together, they establish that interventions at both sites can steer the generated style.

5.2 ABLATION AT IDENTIFICATION AND EXECUTION SITES

On $k = 4$ prompts demonstrating convention B for each family of styles (see Table 4 for which convention is A and B for each family), we run ablation tests to see if the model can stop learning convention B in context. Again we ablate across all layers (Appendix E). From an unablated 73.0%, success falls to 31.0% for the identification feature only ablation and to 40.5% for the execution feature only ablation (other-family controls retain 73.8% and 71.8%, see Figure 6). Thus both sites are causally necessary for performance.

5.3 RELATIONSHIP BETWEEN IDENTIFICATION AND EXECUTION FEATURES

Identification steering moves the execution feature. On $k = 3$ prompts demonstrating one style, we steer using the identification feature at the evidence tokens toward the alternative style (adding $-\alpha d_f^{\text{id}}$) and measure the resulting shift $\Delta z_\alpha = z_\alpha - z_0$ of the residual stream at the cue token at layer 23, where z_0 is the unsteered activation. At $\alpha = 2$, Δz_α points in the same direction as the execution feature $-d_f^{exec}$: $\cos(\Delta z_\alpha, -d_f^{exec}) = 0.628$ (95% interval $[0.572, 0.682]$), versus 0.100 for other-family steering ($-d_{f'}^{id}$) and -0.006 for equal-norm random steering. See Appendix E for more details.

Identification and execution feature pairs are linearly related. We want to see if there is a general linear map that can go from the identification direction to the execution direction, $\hat{d}_f^{\text{id}}$ to $\hat{d}_f^{\text{exec}}$. We find the best results if we train an affine ridge map on the directions at a prompt level from $\hat{d}_{fj}^{\text{id}}$

to $\hat{d}_{fj}^{\text{exec}}$, where d_{fj}^{q} is the activation difference at site q between document j of family f rendered in conventions A and B. Regularization is selected by leave-one-family-out validation on training families and then evaluated on held-out family centroids ($\hat{d}_{f}^{\text{id}}$ to $\hat{d}_{f}^{\text{exec}}$). Across five nominal 66/34 family splits, $R^2 = 0.238 \pm 0.031$ (mean and sample standard deviation across splits). Appendix E gives the training objective, centroid normalization, score definition, and per-split results.

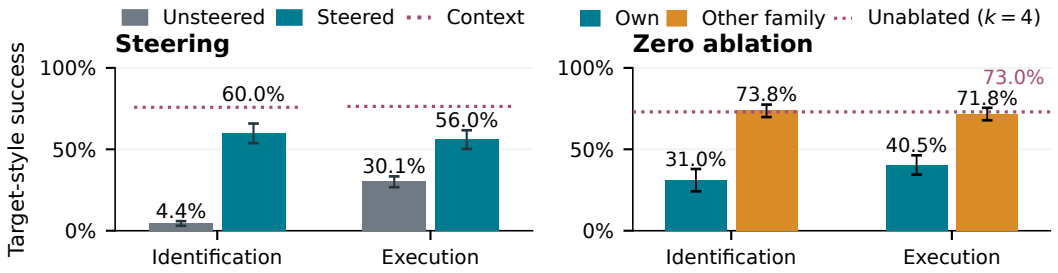

Figure 6: **Coding-style steering and ablation.** Left: steering with dotted context references. Right: zero ablation under alternative-convention demonstrations; the dotted line is the unablated baseline. Error bars: 95% family-bootstrap intervals.

6 Limitations

Our simple tasks are short input-output mappings in a fixed `Q:`/`A:` format, studied in two 6–7B-parameter models. Whether identification and execution separate in the same way for multi-step reasoning, more varied or longer contexts, or larger models is untested. In the coding-style setting, the relationship between features is weaker: a shared linear map predicts held-out families' execution features only partially ($R^2 = 0.238$). This could be resolved with more diverse families. The coding experiments also use a single model, lack a combined steering-plus-ablation test, and score success with a style detector and model-judge acceptance rather than functional correctness. More broadly, our features are constructed using single directions at reasonable intervention targets rather than exhaustive search of task information (in the coding styles setting) and do not address the possibility that multiple features for the same pattern lie in a low-dimensional subspace.

7 Discussion and Conclusion

We show that in-context learning (Brown et al., 2020) runs through a causal pathway from identification features at demonstration tokens to execution features at the cue token. At tokens that exhibit certain patterns, an identification step occurs, where the model encodes the patterns seen, which is followed by an execution step to drive the output, at tokens where the model needs to continue the pattern (cue tokens). We find features in the residual stream (Elhage et al., 2021; Park et al., 2024) to characterize each step in multiple contexts. For patterns generated by simple input-output tasks, the identification feature is a task-specific direction alongside a label direction shared across tasks, and the execution feature is the task's function vector (Todd et al., 2024). For patterns of coding conventions (Allamanis et al., 2014), both features are mean activation difference vectors (Rimsky et al., 2024). For both features, steering induces behavior and ablation impairs it (Turner et al., 2023; Arditi et al., 2024) and the two steps are linked. Steering and ablating the identification feature raises and lowers residual stream alignment with the execution feature. In simple patterns, the behavioral effect requires the execution feature. Shared affine maps partially predict held-out execution features from identification features, strongly for simple tasks and somewhat for coding styles. These results support a general mechanistic account of ICL across two model families and two kinds of in-context learning: identification features are first encoded at label tokens, and they are carried to cue tokens (Wang et al., 2023b; Hendel et al., 2023), where an execution feature forms and applies it. Tracing the attention pathways that carry identity to the query (Olsson et al., 2022; Wang et al., 2023a), and testing the account in larger models and multi-step tasks, are the natural next steps.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data sets, implement methods, design and provide feedback on experiments, clean and reformat data sets, support data analysis, and interpret results. The 100 new input-output tasks (Appendix A) were ideated and specified with LLM agents; their examples come from deterministic generator scripts and curated fact lists, checked by automated structural validation. The coding-style documents (Appendix E) were written by LLMs (Claude Opus and GPT-5); alternative renderings come from exact rewrite rules where available, and documents were screened by an LLM reviewer and automated static checks. Gemini 2.5 Flash is part of the coding-style success metric as a plausibility judge. AI coding agents (Claude Code and OpenAI Codex) implemented experiment, analysis and plotting code, ran experiments, and proposed experimental designs and controls. We have not used generative AI tools to formulate mathematical claims or proofs, and translation and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to create and modify figures, search and summarize the literature, and draft and edit the paper. We have reviewed all AI-assisted work: the authors checked generated data sets and code, reproduced key results from saved intermediates, and verified reported numbers against the underlying result files. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work analyzes the internal representations of publicly available language models (GPT-J-6B, Qwen2.5-7B-Instruct and Qwen2.5-7B base) on synthetic input-output tasks and generated code documents. It involves no human subjects, personal data or crowdsourced annotation. The coding-style documents were written by language models and screened by an LLM reviewer and automated static checks to make sure paired documents differ only in stylistic conventions. The steering and ablation interventions we study change model behavior at inference time, but they are applied only to benign pattern-completion and coding-style tasks, and a better understanding of how demonstrations shape model computation may help detect or prevent unwanted in-context behavior.

REPRODUCIBILITY STATEMENT

All models we study are publicly available (GPT-J-6B, Qwen2.5-7B-Instruct and Qwen2.5-7B base). Appendix A lists every task with examples, and Appendix B gives the prompt format, the dummy prompts, and the head-selection and function-vector steering protocol. Appendix C specifies how identification features are extracted, the layer sweeps, and the steering and ablation interventions; Appendix D gives the downstream readouts, the mediation test and the fitting and scoring of the linear maps, including regularization and task splits; and Appendix E describes the coding-style corpus, success scoring, feature estimation, intervention settings and mapping analysis. Generations are sampled at temperature 1, and random splits and bootstrap intervals use the fixed seeds stated with each result.

REFERENCES

Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 281–293, 2014. doi: 10.1145/2635868.2635883. URL https://arxiv.org/abs/1402.4182.

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. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/f545448535dfde4f9786555403ab7c49-Abstract-Conference.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language Models are Few-Shot Learners, 2020. URL https://arxiv.org/abs/2005.14165.

Madeline Brumley, Joe Kwon, David Krueger, Dmitrii Krasheninnikov, and Usman Anwar. Comparing bottom-up and top-down steering approaches on in-context learning tasks. In *NeurIPS 2024 Workshop on Foundation Model Interventions (MINT)*, 2024. URL https://arxiv.org/abs/2411.07213.

Hakaze Cho, Mariko Kato, Yoshihiro Sakai, and Naoya Inoue. Revisiting in-context learning inference circuit in large language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.04468.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In *Proceedings of the Third Conference on Causal Learning and Reasoning*, volume 236 of *PMLR*, pp. 160–187, 2024. URL https://proceedings.mlr.press/v236/geiger24a.html.

Seungwook Han, Jinyeop Song, Jeff Gore, and Pulkit Agrawal. Emergence and effectiveness of task vectors in in-context learning: An encoder decoder perspective. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 21871–21897, 2025. URL https://proceedings.mlr.press/v267/han25h.html.

Roee Hendel, Mor Geva, and Amir Globerson. In-Context Learning Creates Task Vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. URL https://aclanthology.org/2023.findings-emnlp.624/.

Xinyan Hu, Kayo Yin, Michael I. Jordan, Jacob Steinhardt, and Lijie Chen. Understanding In-context Learning of Addition via Activation Subspaces, 2025. URL https://arxiv.org/abs/2505.05145.

Jiachen Jiang, Yuxin Dong, Jinxin Zhou, and Zhihui Zhu. From Compression to Expression: A Layerwise Analysis of In-Context Learning, 2025. URL https://arxiv.org/abs/2505.17322.

Dmitrii Kharlapenko, Stepan Shabalin, Arthur Conmy, and Neel Nanda. Scaling sparse feature circuits for studying in-context learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 29819–29851, 2025. URL https://proceedings.mlr.press/v267/kharlapenko25a.html.

Dongfang Li, Zhenyu Liu, Xinshuo Hu, Zetian Sun, Baotian Hu, and Min Zhang. In-context learning state vector with inner and momentum optimization. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0ed52d7f6f641f228405d48a611e0684-Abstract-Conference.html.

Sheng Liu, Haotian Ye, Lei Xing, and James Y. Zou. In-context Vectors: Making In Context Learning More Effective and Controllable Through Latent Space Steering. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32287–32307. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bx.html.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064, 2022. URL https://aclanthology.org/2022.emnlp-main.759/.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli,

Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. URL https://arxiv.org/abs/2209.11895. arXiv:2209.11895.

Jane Pan, Tianyu Gao, Howard Chen, and Danqi Chen. What In-Context Learning "Learns" In-Context: Disentangling Task Recognition and Task Learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023. URL https://aclanthology.org/2023.findings-acl.527/.

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*, volume 235 of *Proceedings of Machine Learning Research*, 2024. URL https://arxiv.org/abs/2311.03658.

Qwen Team. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/, September 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. URL https://arxiv.org/abs/2603.23629.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, 2024. URL https://aclanthology.org/2024.acl-long.828/.

Suzanna Sia, David Mueller, and Kevin Duh. Where does in-context learning happen in large language models? In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/3979818cdc7bc8dbeec87170c11ee340-Abstract-Conference.html.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function Vectors in Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.15213.

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. URL https://arxiv.org/abs/2308.10248.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Entang Wang, Yiwei Wang, Aleksandra Bakalova, and Michael Hahn. How few-shot examples add up: A causal decomposition of function vectors in in-context learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.16591.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *International Conference on Learning Representations*, 2023a. URL https://arxiv.org/abs/2211.00593.

Lean Wang, Lei Li, Damai Dai, Deli Chen, Hao Zhou, Fandong Meng, Jie Zhou, and Xu Sun. Label Words are Anchors: An Information Flow Perspective for Understanding In-Context Learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9840–9855. Association for Computational Linguistics, 2023b. doi: 10.18653/v1/2023.emnlp-main.609. URL https://aclanthology.org/2023.emnlp-main.609/.

Haolin Yang, Hakaze Cho, and Naoya Inoue. Localizing Task Recognition and Task Learning in In-Context Learning via Attention Head Analysis, 2025a. URL https://arxiv.org/abs/2509.24164.

Haolin Yang, Hakaze Cho, Yiqiao Zhong, and Naoya Inoue. Unifying attention heads and task vectors via hidden state geometry in in-context learning. In *Advances in Neural Information Processing Systems*, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1f3e9dddea1df5b3049192172f2e163f-Abstract-Conference.html.

Liu Yang, Ziqian Lin, Kangwook Lee, Dimitris Papailiopoulos, and Robert D. Nowak. Task vectors in in-context learning: Emergence, formation, and benefits. In *Conference on Language Modeling*, 2025c. URL https://arxiv.org/abs/2501.09240.

Kayo Yin and Jacob Steinhardt. Which Attention Heads Matter for In-Context Learning? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72428–72461. PMLR, 2025. URL https://proceedings.mlr.press/v267/yin25e.html.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.16042.

Bowen Zheng, Ming Ma, Zhongqiao Lin, and Tianming Yang. Label words as local task vectors in in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6319–6332, 2026. URL https://aclanthology.org/2026.findings-acl.314/.

Notation. As in the main text, a hat on a direction denotes normalization to unit length ($\hat{u}_A$, $\hat{g}_\ell$); on a predicted quantity ($\widehat{v}_A$, $\widehat{y}$) it denotes a prediction.

A Full Task Inventory

Table 1 lists the 117 candidate input-output tasks before filtering for sufficient accuracy. Tasks marked * (29) come from Todd et al. (2024); the other 88 were built for this work: 80 of the 100 generated tasks described in the statement on generative AI (the remaining 20 were set aside before task selection) and 8 earlier additions in the same format.

Each row gives the dataset identifier, the input-output transformation, and two illustrative examples. In the experiments, pairs are rendered as `Q: <input>` followed by `A: <output>`, the final query ends at `A:`. Arrows in the table are only for visualization purposes. Capitalization, accents, and output-label conventions are preserved.

Table 1: Full inventory of 117 candidate FV tasks

Task	Description	Input → output examples
adjective_to_adverb	Convert an adjective to an adverb.	inextricable → inextricably usual → usually
adjective_to_noun	Return the noun associated with an adjective.	strategic → strategy cyclical → cycle
adverb_to_adjective	Convert an adverb to its base adjective.	finely → fine confidently → confident
ag_news*	Classify a news passage as World, Sports, Business, or Science.	Socially Responsible Funds on a Tear Don't be too impressed – great returns don't always mean much. → Business Dell Debuts Color Laser Printers Three new models are designed for businesses and home office users. → Science
agent_noun_to_verb	Convert an agent noun to its associated verb.	crier → cry headliner → headline
alphabet_position_first_letter	Return the alphabet position of the first letter (a–j).	hydra → 8 fooling → 6
animal_class	Return an animal's biological class.	junebug → insect bullmastiff → mammal
animal_plant_object	Classify a word as animal, plant, or object.	boardgame → object scorpion → animal
antonym*	Return an antonym.	flawed → perfect orthodox → unorthodox
article_choice	Choose the indefinite article (a or an).	estimation → an khan → a
capitalize*	Capitalize a word's initial letter; return the whole word.	without → Without emu → Emu
capitalize_first_letter*	Return only the first letter, in uppercase.	whose → W deliver → D
capitalize_last_letter*	Return only the last letter, in uppercase.	throughout → T skunk → K
capitalize_second_letter*	Return only the second letter, in uppercase.	chinchilla → H lively → I

Continued on next page

Table 1 (continued)

Task	Description	Input → output examples
city-country	Return the country containing a city.	Saint John's → Antigua and Barbuda Lille → France
commonsense_qa*	Answer a multiple-choice commonsense question with its option letter.	What is a hand? a: body part b: foot c: handglide d: feet e: help → a What blocks sunshine? a: summer b: park c: desktop d: sea e: moon → e
compound_first	Return the first component of a compound word.	cartwheel → cart volleyball → volley
compound_head	Return the final component of a compound word.	cartwheel → wheel volleyball → ball
concrete_abstract	Classify a noun as concrete or abstract.	goodwill → abstract whisk → concrete
contains_letter_e	Determine whether a word contains the letter e (yes/no).	aiming → no qualified → yes
count_consonants	Count consonants in a word.	pale → 2 author → 3
count_digits	Count digits in an integer.	394782 → 6 1540 → 4
count_letter_e	Count occurrences of the letter e.	likewise → 2 semitic → 1
count_words	Count words in a phrase.	acting generic ritual whatever → 4 closed hate → 2
count_zeros	Count zeros in a digit string.	200709 → 3 30063 → 2
country-capital*	Return a country's capital.	Afghanistan → Kabul Albania → Tirana
country-currency*	Return the currency associated with a country.	Afghanistan → Afghani (AFN) Albania → Albanian Lek (ALL)
day_after_textual_date	Return the next calendar day as month and day, omitting the year.	July 13, 2052 → July 14 March 19, 2011 → March 20
double_last_letter	Repeat a word's final letter.	nine → ninee august → augustt
double_no_carry	Double an integer whose digits are at most four.	13223 → 26446 31344 → 62688
double_number	Double an integer.	786 → 1572 517 → 1034
east_neighbor	Return the dataset-listed eastern neighbor of a country or territory.	Afghanistan → Pakistan Aland Islands → Finland
ends_with_ing	Determine whether a word ends in ing (yes/no).	vertical → no caroling → yes
english-french*	Translate an English word into French.	dependent → dépendant those → ceux

Continued on next page

Table 1 (continued)

Task	Description	Input → output examples
english-german*	Translate an English word into German.	seminary → seminar contributed → beitragen
english-italian	Translate an English word into Italian.	interesting → interessante bracelet → braccialetto
english-portuguese	Translate an English word into Portuguese.	shoe → sapato argument → argumento
english-spanish*	Translate an English word into Spanish.	proposal → propuesta journal → diario
first_digit	Return the first digit of a number.	721851 → 7 58339 → 5
first_letter_second_word	Return the first letter of the second word, in lowercase.	maximum order → o unusual judge → j
first_three_letters	Return the first three letters of a word.	coffee → cof potential → pot
first_two_letters	Return the first two letters of a word.	beginning → be usually → us
first_vowel	Return the first vowel in a word.	resurgence → e interruption → i
french-english	Translate a French word into English.	onze → eleven honte → shame
french_noun_gender	Return a French noun's gender (masculine/feminine).	fédération → feminine centre → masculine
german-english	Translate a German word into English.	Schinken → ham vorbereiten → prepare
german_noun_gender	Return a German noun's gender-marking article (der/die/das).	Stadion → das Förderung → die
gerund_to_base	Convert an ing-form verb to its base form.	describing → describe priming → prime
gerund_to_past	Convert an ing-form verb to the past tense.	cabling → cabled gifting → gifted
halve_even_number	Halve an even integer.	1724 → 862 1122 → 561
hypernym_category	Return a word's broader semantic category.	hickory → tree tin → metal
initials_two_words	Return the uppercase initials of two words.	healthy regular → HR indirect protein → IP
iso_date_to_month	Return the month name from a YYYY-MM-DD date.	1935-04-13 → April 2082-12-05 → December
iso_date_year_plus_one	Return only the year following the year in an ISO date.	1803-07-05 → 1804 1935-10-16 → 1936
landmark-country*	Return the country associated with a landmark or named location.	Autonomous University of Madrid → Spain Pochepsky District → Russia

Continued on next page

Table 1 (continued)

Task	Description	Input → output examples
language_identification	Identify the language of a word.	bisogno → Italian Stadt → German
larger_of_pair	Return the larger of two space-separated integers.	151 685 → 685 922 348 → 922
larger_than_1000	Determine whether a number exceeds 1000 (yes/no).	302 → no 20377 → yes
largest_digit	Return the largest digit of an integer.	28064 → 8 50789 → 9
last_digit	Return the final digit of an integer.	27 → 7 475 → 5
last_two_letters	Return the last two letters of a word.	beginning → ng usually → ly
last_vowel	Return the last vowel in a word.	year → a between → e
lowercase_first_letter*	Return only the first letter, in lowercase.	WITTY → w SPICY → s
lowercase_last_letter*	Return only the last letter, in lowercase.	PILLOW → w GAZELLE → e
lowercase_word	Convert the entire word to lowercase.	RISK → risk TOTAL → total
national_parks*	Return the US state or district associated with a national park or other listed site.	Greenbelt Park → Maryland Rock Creek Park → Washington DC
nato_first_letter	Return the NATO alphabet codeword for the first letter.	moving → Mike training → Tango
natural_manmade	Classify an entity as natural or manmade.	pulley → manmade pumice → natural
next_capital_letter*	Return the alphabet successor of the first letter, in uppercase.	hammock → I build → C
next_in_group	Return the next element in the same periodic-table group.	Hydrogen → Lithium Lithium → Sodium
next_in_period	Return the next element in the same periodic-table period.	Hydrogen → Helium Lithium → Beryllium
next_item*	Return the next item in an ordered sequence.	zero → one one → two
next_letter_of_last_letter	Return the alphabet successor of the last letter, in uppercase.	risk → L total → M
next_month_of_date	Return the month following the supplied month, omitting the year.	May 2033 → June October 1970 → November
next_number_digits	Add one to an integer written in digits.	1 → 2 2 → 3
number_word_to_digits	Convert an English number expression to digits.	forty → 40 eighty → 80

Continued on next page

Table 1 (continued)

Task	Description	Input → output examples
park-country*	Return the country containing a national park.	Band-e Amir National Park → Afghanistan Butrint National Park → Albania
past_to_base	Convert a past-tense verb to its base form.	hid → hide led → lead
person-instrument*	Return the instrument associated with a musician.	Toko Yasuda → guitar Mark Isham → trumpet
person-occupation*	Return the occupation associated with a person.	Billy Roche → actor Arun Nehru → politician
person-sport*	Return the sport associated with a person.	Andreas Ivanschitz → soccer Roberto Clemente → baseball
person_place_thing	Classify a noun as person, place, or thing.	wallet → thing moor → place
plural_to_singular	Convert a plural noun to singular.	condemnations → condemnation inheritances → inheritance
plus_hundred	Add 100 to an integer.	77228 → 77328 49044 → 49144
pos_label	Return a word's part-of-speech label.	historically → adverb costly → adjective
present-past*	Convert a base-form verb to the past tense.	accelerate → accelerated adapt → adapted
prev_capital_letter	Return the alphabet predecessor of the first letter, in uppercase.	funny → E wish → V
prev_item*	Return the previous item in an ordered sequence.	one → zero two → one
prev_number_digits	Subtract one from an integer written in digits.	1 → 0 2 → 1
product-company*	Return the company or producer associated with a product.	Platform Controller Hub → Intel Shahab-2 → Iran
remove_first_letter	Remove the first letter of a word.	coach → oach fall → all
remove_last_letter	Remove the last letter of a word.	coach → coac fall → fal
rhyme	Return the dataset-listed rhyming partner of a word.	ability → anti able → disable
round_up_to_ten	Round an integer up to the next multiple of ten.	7833 → 7840 282 → 290
second_digit	Return the second digit from the left.	27 → 7 457 → 5
sentiment*	Classify the sentiment of a text as positive or negative.	unpretentious, charming, quirky, original → positive Disney again ransacks its archives for a quick-buck sequel. → negative

Continued on next page

Table 1 (continued)

Task	Description	Input → output examples
singular-plural*	Convert a singular noun to plural.	wallet → wallets keychain → keychains
singular_or_plural	Classify a noun's number as singular or plural.	reviews → plural end → singular
smaller_of_pair	Return the smaller of two space-separated integers.	931 156 → 156 862 614 → 614
spanish-english	Translate a Spanish word into English.	rayo → lightning temperatura → temperature
spanish_noun_gender	Return a Spanish noun's gender (masculine/feminine).	ausencia → feminine chile → masculine
starts_with_vowel	Classify the initial letter as vowel or consonant.	large → consonant anything → vowel
strip_prefix	Remove a derivational prefix from a word.	unwell → well preselection → selection
syllable_count	Return the number of syllables in a word.	properly → 3 except → 2
synonym*	Return a synonym.	curse → swear notorious → infamous
third_letter	Return the third letter of a word.	beginning → g usually → u
third_person_to_base	Convert a third-person singular verb to its base form.	excuses → excuse rings → ring
time_to_minutes	Convert a 24-hour clock time to minutes since midnight.	06:43 → 403 00:58 → 58
times_five	Multiply an integer by five.	929 → 4645 616 → 3080
titlecase_phrase	Capitalize the first letter of each word in a phrase.	mat name → Mat Name cursed contact → Cursed Contact
uppercase_word	Convert the entire word to uppercase.	risk → RISK total → TOTAL
us-city-state	Return the US state associated with a city or listed locality.	Auburn University → Alabama Iowa City → Iowa
verb_tense_label	Classify a verb form as present, past, or gerund.	stiffs → present braiding → gerund
verb_to_third_person	Convert a base-form verb to third-person singular.	crown → crowns apply → applies
west_neighbor	Return the dataset-listed western neighbor of a country or territory.	Afghanistan → Turkmenistan Aland Islands → Sweden
word_length*	Return the number of letters in a word.	along → 5 ant → 3
word_polarity	Classify a word's polarity as positive or negative.	deceptive → negative healthy → positive

B FUNCTION-VECTOR STEERING PROTOCOL

Illustrative prompt. Here is an example of what a two-shot prompt looks like for the antonym task:

```
Q: hot
A: cold

Q: big
A: small

Q: fast
A:
```

Here, `hot` and `big` are input tokens. `cold` and `small` are their respective label tokens. The `:` at the end of the final `A:` is the cue token, and `slow` is the intended completion. Line breaks are part of the prompt, and demonstrations are separated by a blank line.

Dummy prompts. Dummy prompts keep each demonstration's input but replace its label with the filler token _, so the context shows the prompt format but not the task. An illustrative dummy six-shot prompt for the antonym task is:

```
Q: hot
A: _

Q: big
A: _

Q: early
A: _

Q: loud
A: _

Q: happy
A: _

Q: full
A: _

Q: fast
A:
```

Identification steering adds its vector at each _ slot; the one-shot variant keeps a single demonstration.

Following the sparse head-selection approach of Hu et al. (2025), we learn gates $\gamma_h \in [0, 1]$ using a pooled zero-shot negative log-likelihood objective with an ℓ_1 penalty, injecting the weighted sum at zero-indexed layer 9 (roughly 1/3 of the depth into the model, following the same procedure as Hu et al. (2025)). Heads with $\gamma_h > 0.8$ are retained, and evaluation sums their mean projected outputs without gate weights. GPT-J uses penalty coefficient 0.005, selected by five-fold task cross-validation on its training tasks. Qwen uses 0.001, selected by five-fold task cross-validation on its training tasks.

Function vectors are extracted from separate clean ten-shot source prompts than the ones used to fit the sparse head selection. All layer numbers in this steering analysis denote zero-indexed transformer layers, and injection is at the layer output at the final cue token. We report both a shared injection layer chosen by mean training-task evaluation accuracy and the maximum for each task over all 28 layers. The former uses no held-out-task scores to choose its injection layer while the latter is optimized on each task's evaluation queries. Neither is an independent estimate of the training-task layer-selection procedure's performance.

To test steering without choosing a separate injection layer for each held-out task, we select the layer maximizing mean accuracy on the training tasks and apply it to all tasks. This selects zero-indexed layer 1 for GPT-J and layer 9 for Qwen. Held-out accuracy rises from 9.0% to 62.9% for GPT-J and from 5.8% to 64.4% for Qwen. The corresponding training-task accuracies rise from 2.8% to 65.0% and from 2.4% to 77.3%. Allowing each task its own best layer gives held-out accuracies of 73.4% and 75.8%, but those values use evaluation performance to choose the layer and should be interpreted as task-wise sweep maxima.

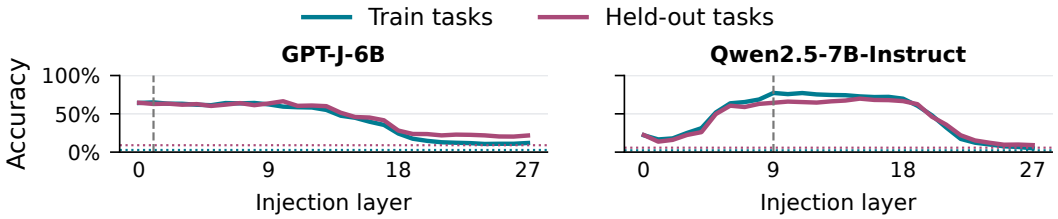

Figure 7: **Zero-shot FV steering.** Solid lines show accuracy when steering at a given layer, dotted lines show the unsteered baselines for 0-shot prompts. Dashed vertical lines mark best performing injection layers selected on training tasks.

C Identification probes and causal interventions

Decodability. For GPT-J we fit one ridge regression per layer and token position, mapping a single residual activation to that prompt's FV, which is the activation of the selected attention heads for that prompt at the final cue token (averaging all the per-prompt FVs gives the overall task-mean FV, v_A). Probes are trained on the 55 training tasks (150 clean ten-shot prompts each) and evaluated on the 14 held-out tasks. Regularization is chosen from 19 log-spaced values between 10^{-1} and 10^{8} by five-fold cross-validation over training tasks, minimizing pooled squared error. Inputs and targets are centered on training means and each map is affine in the original coordinates. We score predictions against the task-mean FV v_A and report the mean over held-out tasks of

$$R^2_A = 1 - \frac{\sum_j \|\hat{v}^j_A - v_A\|^2}{n_A \|v_A - \bar{v}_{\text{heldout}}\|^2},$$

where $\hat{v}^j_A$ is the probe's prediction of the FV of prompt j of task A, n_A is the number of evaluation prompts of task A, and $\bar{v}_{\text{heldout}}$ is the mean of v_A over held-out tasks. The held-out mean in the denominator is used only for scoring. The cue positions in the figure are those preceding demonstrations 1 and 6, not the final cue token of a six-shot prompt.

For Qwen we ran an auxiliary probe at the last token of the tenth demonstration label, using the same regularization grid and cross-validation with 77 training and 19 held-out tasks (prompt indices are matched to the per-prompt FVs after checking token positions). It scores per-prompt FV predictions with a coordinate-averaged R^2 over coordinates with nonzero held-out variance:

$$R^2_{\text{coord}} = \frac{1}{|D_+|} \sum_{d \in D_+} \left[1 - \frac{\sum_j (\hat{v}_{j,d} - v_{j,d})^2}{\sum_j (v_{j,d} - \bar{v}_d)^2} \right],$$

where D_+ is the set of FV coordinates d with nonzero held-out variance, $v_{j,d}$ is coordinate d of held-out prompt j's FV, $\hat{v}_{j,d}$ its prediction, and $\bar{v}_d$ its held-out mean. Because this target and averaging differ from the GPT-J metric, and from the task-centroid score used in the map analyses, we do not plot it with the GPT-J results. It reaches $R^2 = 0.2775$ at layer 12 and peaks at $R^2 = 0.2899$ at layer 20.

Spanning-vector extraction and steering. We follow the mean-activation addition of Todd et al. (2024), but extract and inject at demonstration labels rather than at the final cue token; this differs from replacing a cue activation with a task vector (Hendel et al., 2023). Task means $m_A(\ell)$ are layer-output residuals at the last token of the final demonstration label in clean ten-shot prompts.

Layer sweep. The label-mean sweep adds $\alpha m_A(\ell)$ to a single dummy-label slot, at the same layer it was extracted from, on prompts of the form `Q: <demo input> / A: _ / Q: <query> / A:` (newlines in place of the slashes). We test $\alpha \in \{0.5, 1, 2, 4\}$; Figure 8 fixes $\alpha = 2$ for every task. At this strength, label-mean accuracy peaks at layer 6 in GPT-J and layer 12 in Qwen. One-slot $c + u_A$ steering at these layers, the one-shot result in the main text, reaches 11.2% and 54.0%, against real one-shot baselines of 20.8% and 47.6%. The extraction band B is the peak layer and its two neighbors (layers 5–7 in GPT-J and 11–13 in Qwen). A shared-mean control, which averages label means over training tasks at the same layer, gives only 0.9% and 4.7%; like the label direction c, it uses training tasks only, but unlike c it is not averaged over the extraction band. The six-slot

experiment in the main text steers with $c + u_A$ at the same layers (6 and 12), reaching 38.1% and 54.3% at $\alpha = 2$, against 11.2% and 54.0% with one slot; these comparisons across stored scaffold evaluations are descriptive.

Shared component of label means. Label-token task means are strongly aligned across tasks, which motivates separating a label direction shared across tasks from each task's identification feature. Figure 9 shows pairwise cosines between tasks, over all 69 GPT-J and 96 Qwen tasks, of the label means $m_A(\ell)$ at the identification layer (6 in GPT-J, 12 in Qwen), of the task FVs v_A, and of the identification features u_A (label direction estimated from training tasks). Label means have mean pairwise cosine 0.73 in GPT-J and 0.50 in Qwen, far from the magnitude of about 0.02 expected for unrelated directions in these dimensions, and removing the label direction brings the mean to -0.00 and -0.01. In GPT-J, label means are also much more similar across tasks than FVs are (0.39). Qwen's FVs share an even larger common component (0.62), consistent with their dominant shared FV direction (shared and task-specific execution components).

Label direction plus identification feature steering. For $s_A = c + u_A$, every label of six in-distribution demonstrations is replaced by _, and all six slots receive the same vector, injected at layers 6 (GPT-J) and 12 (Qwen), the center of the extraction band B, as in the main text. A separate one-dummy-label sweep of $c + u_A$ over all 28 layers and $\alpha \in \{0.5, 1, 2, 4\}$, taking the layer with the highest task-averaged accuracy at each task's best strength, peaks at layer 0 for GPT-J (19.4%) and layer 12 for Qwen (55.8%). With six slots at layer 0, GPT-J reaches 57.0% at $\alpha = 2$, and Qwen 20.7% at $\alpha = 1$ and 17.3% at $\alpha = 2$. Figure 4 shows six-slot $c + u_A$ steering.

Identification feature alone. In GPT-J, setting the component along $\hat{u}_A$ to $\alpha\|u_A\|$ at the layer 6 output of all six dummy slots peaks at 30.9% (at $\alpha = 4$), below $c + u_A$ at the same layer (38.1%). The capture statistic, the fraction of $\|m_A(\ell)\|^2$ that lies in $\mathrm{span}\{c, u_A\}$ (computed per task), measures how much of the label mean these two directions account for. At the steering layer it has median 96.8% in GPT-J (minimum 95.4%) and 96.3% in Qwen (minimum 94.2%).

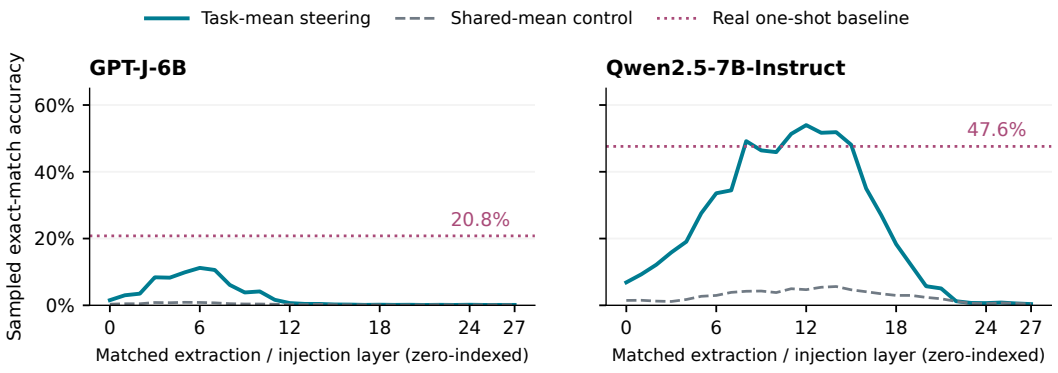

Figure 8: **Identification steering with one dummy demonstration.** Task-mean and shared-mean vectors are extracted and injected at the same layer, with fixed $\alpha = 2$. Dotted lines show the real one-shot baselines. The shared-mean control averages training-task vectors and contains no task-specific choice. Task means cover all 69 GPT-J or 96 Qwen tasks; the shared mean uses training tasks only.

Ablation. Our projection ablations apply the interventions of Hu et al. (2025) to residual-stream directions. Mean replacement parallels the head-output ablations of Yin & Steinhardt (2025), but removes one residual component rather than a whole head output. Identification ablations project out $\hat{u}_A$ at every demonstration-label token, at the outputs of all layers 0–27, and mean replacement restores the layer-specific mean activation over demonstration-label tokens in the task pool. Execution ablations act only at the final cue token, at the outputs of layers 9–27, and mean replacement uses the equal-task-weighted mean cue-token activation. Both act only while the prompt is processed. Control directions come from a randomly chosen other task (seeded, from another semantic family); own-task and control features are unit vectors.

All ablations reuse the fixed prompt banks, truncated to one or six demonstrations: 150 prompts per task, except two small-pool Qwen tasks with 52. Completions are sampled at temperature 1

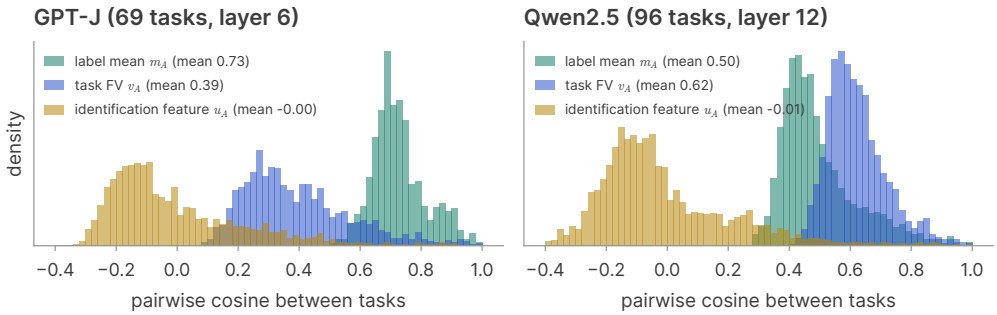

Figure 9: **Shared component of label means.** Distributions of pairwise cosines between tasks for label-token means $m_A(\ell)$ at the identification layer, task FVs v_A, and identification features u_A (label direction removed).

with unrestricted top-k/top-p, cut at the first newline, stripped, and compared with the stored gold string; baselines come from the matching prompt scaffolds. The label direction c and $\hat{g}_\ell$ use training tasks only, as in the main text, and the mean-ablation reference means are pooled over all tasks; the ablations test within-task effects, so they do not require held-out tasks. The GPT-J identification and execution ablations were sampled separately (baselines 62.9% and 63.9%), so each figure uses its own baseline.

Mean and zero ablation comparisons. Mean ablation of the identification feature lowers six-shot accuracy from 62.9% to 13.2% in GPT-J and from 79.8% to 20.5% in Qwen, while other-task mean ablation leaves 63.2% and 75.4%. Zero ablation gives 12.0% and 21.1% for own-task features and 61.4% and 74.8% for other-task features. Figure 10 shows both operators at both sites; the weaker effect of execution mean ablation in Qwen is analyzed below.

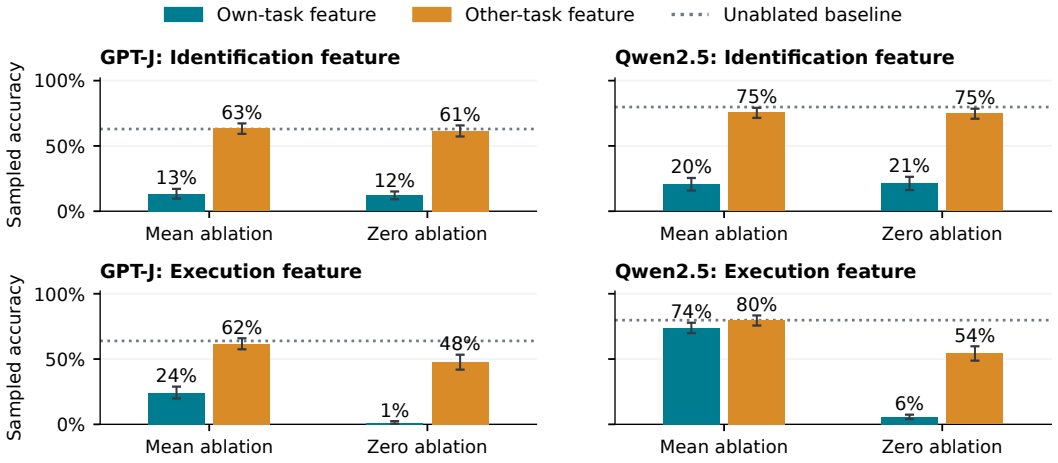

Figure 10: **Mean and zero ablation at both sites.** Six-shot sampled accuracy after ablating own-task or other-task directions. Identification interventions act at all demonstration-label tokens at the outputs of layers 0–27; execution interventions act at the final cue token at the outputs of layers 9–27. Dotted lines mark separately sampled baselines. Error bars are 95% task-bootstrap intervals.

Shared and task-specific execution components. Full-FV mean ablation lowers accuracy to 24.2% in GPT-J but only to 73.9% in Qwen (other-task controls 61.6% and 79.7%). To understand this gap, we split each FV into a shared FV direction and a task-unique part, much as Hu et al. (2025) separate task-specific from shared signals by replacing head outputs with cross-task means. The shared FV

direction is estimated from training tasks only:

$$c^{\text{exec}} = \frac{1}{|\mathcal{T}|} \sum_{A \in \mathcal{T}} v_A, \qquad \hat{c}^{\text{exec}} = c^{\text{exec}} / \|c^{\text{exec}}\|, \qquad u_A^{\text{exec}} = v_A - (v_A^\top \hat{c}^{\text{exec}}) \hat{c}^{\text{exec}}.$$

These are components of v_A, distinct from the label direction and identification features. The execution feature is still v_A; the rank-two span used below is only an auxiliary intervention space.

Qwen's FVs are dominated by the shared FV direction. Their mean cosine with it is 0.79 (GPT-J: 0.63), and the task-unique part holds only 0.37 of the squared FV norm (GPT-J: 0.60). Replacing the FV projection by its cross-task mean therefore changes the activation much less than zeroing it: the mean-ablation shift is 0.18 of the zero-ablation shift in Qwen and 0.48 in GPT-J (the median ratio over the outputs of layers 9–27 of each task's mean cue-token activation, averaged over tasks).

We then ablate the unit shared FV direction, the unit task-unique direction, or an orthonormal basis of their rank-two span at the final cue token at the outputs of layers 9–27, on clean six-shot prompts. Mean replacement uses the projected layer-specific mean cue-token activation, and generation and scoring follow the full-FV experiment. Baselines, sampled separately, are 63.4% for GPT-J and 80.1% for Qwen (Figure 11).

Intervention	GPT-J	Qwen
No ablation (split run)	63.4%	80.1%
Task-unique own, mean	16.8%	57.9%
Task-unique own, zero	20.3%	55.8%
Task-unique other-task, mean	62.5%	80.1%
Shared FV direction, mean	62.3%	79.2%
Shared FV direction, zero	13.8%	34.1%
Shared + own task-unique span, mean	14.3%	57.3%
Shared + other-task task-unique span, mean	60.7%	79.3%

Table 2: **Accuracy after ablating FV components.** The rank-two conditions remove the span of the shared FV direction and the selected task-unique component. All entries use the split run.

Mean-ablating the task-unique direction lowers accuracy by 0.457 more than the other-task control in GPT-J ($[0.411, 0.504]$) and by 0.222 more in Qwen ($[0.183, 0.260]$), with larger own-task drops in 69/69 and 86/96 tasks (95% paired task-bootstrap intervals, 10,000 resamples, seed 20260923). Different tasks' task-unique directions are nearly orthogonal (mean cosine -0.02 and -0.06), and once the shared FV direction is removed, mean and zero ablation behave alike: mean replacement moves the cue-token activation about as far as zero ablation (shift ratios 1.07 and 1.06). The two components therefore play different roles. The shared FV direction carries a shared execution signal whose magnitude matters: zeroing it is damaging, while mean replacement largely preserves accuracy. The task-unique component carries task-specific execution information in both models, strongly in GPT-J and partially in Qwen, where 57.9% accuracy remains.

One-shot controls and uncertainty. Figure 12 repeats the ablations on one-shot prompts. All plotted intervals resample tasks within a model 10,000 times, keep conditions paired by task, and report the 2.5th and 97.5th percentiles of task-mean accuracy. They hold the vectors and intervention settings fixed, so they do not include uncertainty from choosing those settings or from resampling prompts. Label-mean steering and decodability curves are stored aggregates without intervals.

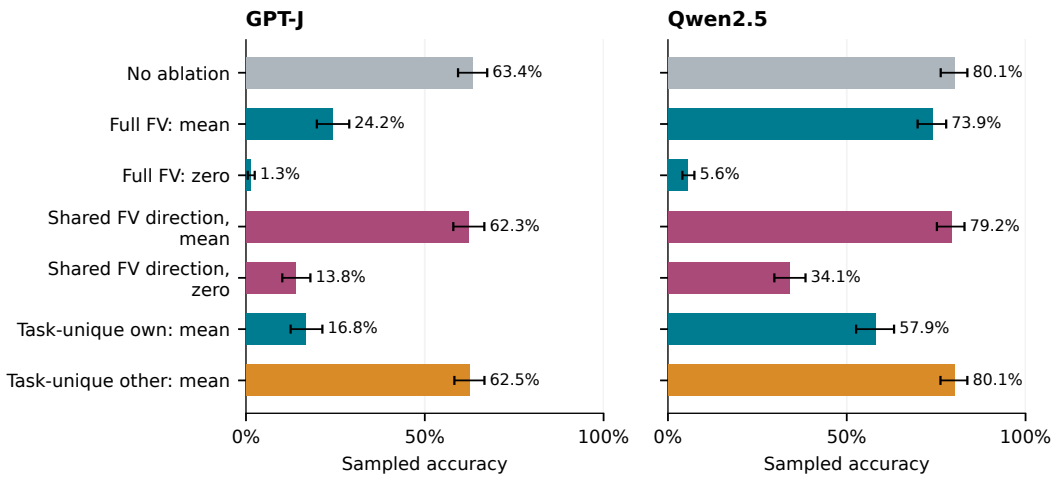

Figure 11: **Shared and task-specific components explain different ablation effects.** Bars show task-mean six-shot accuracy with 95% task-bootstrap intervals. "No ablation" is the split-run baseline (63.4%/80.1%); full-FV bars come from the separately sampled full-FV run on the same prompts, whose baselines are 63.9%/79.8%. Shared-direction and task-unique ablations use training-task mean FVs to define the shared FV direction.

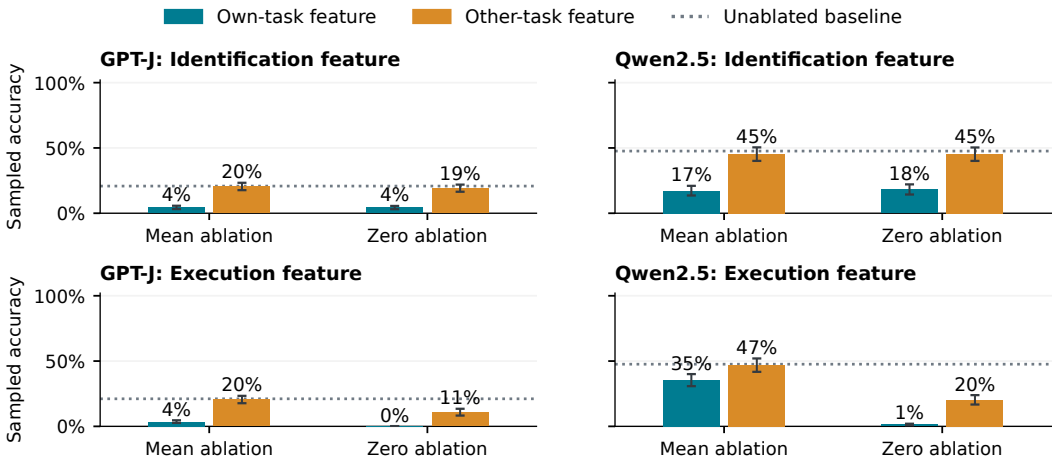

Figure 12: **One-shot ablation controls.** The interventions and plotting conventions match Figure 3, using one real demonstration. Identification features are ablated at label tokens at the outputs of all layers; execution features are ablated at the final cue token at the outputs of layers 9–27.

D IDENTIFICATION-TO-EXECUTION PROTOCOLS

Downstream readouts and controls. These readouts reuse the steering vector $c + u_A$ and dummy-label scaffolds of the identification steering experiment. Each forward pass adds $\alpha(c + u_A)$ at layer 6 in GPT-J or layer 12 in Qwen at every dummy-label slot, for $\alpha \in \{0, 0.5, 1, 2, 4\}$, and reads the cue-token activation at each layer output without generating an answer. We compare two references: the task FV v_A and the generic FV $\bar{v}$, the mean of v_A over the training tasks $\mathcal{T}$. For each reference, we subtract the cosine measured on the same prompt at $\alpha = 0$, then average over prompts and equally over tasks. Intervals resample tasks 10,000 times and do not include uncertainty from vector extraction or layer choice. Task FVs and the generic FV are not orthogonal, so comparing their cosine changes is a descriptive check of specificity, not an orthogonal decomposition of the cue-token activation.

Layers are indexed from zero. GPT-J's readout layer 13 follows the original experiment, near where the selected heads concentrate. Qwen's layer 24 had the largest mean six-shot FV presence and lies

in a broad high-presence region at layers 20–26. Neither layer was chosen to maximize the reported within-task accuracy correlation.

Figure 13 adds two variants: label-mean steering at a single slot, and six-slot $s_A = c + u_A$ steering at layer 0. At $\alpha = 2$, the latter raises execution alignment from 0.183 to 0.424 in GPT-J and from 0.388 to 0.475 in Qwen. At layer 0, six-slot accuracy is 57.0% in GPT-J and 17.3% in Qwen (54.3% at layer 12 in the main text) (Appendix C). These steering readouts complement the ablation readouts below.

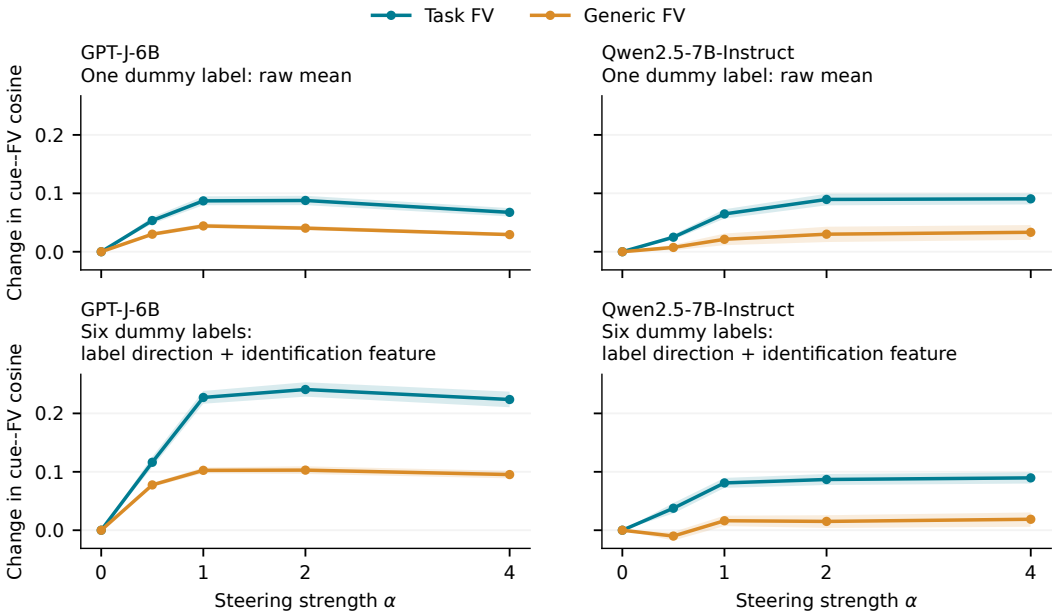

Figure 13: **Additional identification-to-execution interventions.** Top: label-mean steering at one dummy label. Bottom: label-direction-plus-identification-feature steering at six dummy labels, injected at layer 0. Readouts, generic-FV controls, and uncertainty conventions match Figure 5.

Identification-ablation readouts. This experiment reuses the clean six-shot prompts and identification-ablation hook of Appendix C: 150 prompts for each of 69 GPT-J tasks, and for Qwen, 150 prompts for 94 tasks and 52 for two. Each prompt is run under five conditions: no ablation, own-task mean and zero ablation, and paired other-task mean and zero ablation. The unit identification feature is removed at every demonstration-label token, including all tokens of multi-token labels, at the outputs of layers 0–27. Mean ablation replaces its component with the projected layer-specific label-token mean, and zero ablation removes it. The other-task control uses the same fixed different-family pairing as the behavioral ablation, and the final cue token itself is never ablated.

Each forward pass records the cue-token residual activation at every layer output, without sampling a completion; we report the same readout layers as above, 13 for GPT-J and 24 for Qwen. The task reference is the mean per-prompt FV v_A, and the generic reference averages these FVs equally across the training tasks. Cosines are averaged within each task and then across tasks. Intervals resample tasks 10,000 times, keeping conditions paired and holding extraction, control pairing, and readout layers fixed. Figure 14 shows the absolute cosines. The paired contrast subtracts the other-task change from the own-task change, so the unablated baseline cancels.

Own-task ablation lowers alignment more than its other-task control in 69/69 GPT-J tasks and 87/96 Qwen tasks, under either operator. Mean and zero ablation have similar pooled effects but can differ for individual tasks. Under own-task mean ablation, cosine with the generic FV rises slightly, by 0.010 in GPT-J and 0.018 in Qwen, so the drop is specific to the task FV rather than a general loss of alignment. Because the two references are not orthogonal, this comparison does not isolate an orthogonal task-specific component.

Model	Ablation	Own change	Other change	Paired contrast [95% interval]
GPT-J	Mean	-0.112	-0.001	-0.111 $[-0.121, -0.100]$
GPT-J	Zero	-0.112	-0.002	-0.110 $[-0.120, -0.100]$
Qwen	Mean	-0.047	-0.000	-0.047 $[-0.056, -0.039]$
Qwen	Zero	-0.048	$+0.000$	-0.048 $[-0.056, -0.039]$

Table 3: **Identification ablation reduces downstream execution alignment.** Changes are relative to no ablation on the same prompts. Values are rounded to three decimals; near-zero control changes retain their signs.

For scale, adding demonstrations from zero to six raises mean execution alignment by 0.248 in GPT-J and 0.085 in Qwen, in a separately saved sweep. The mean-ablation drops are about 45% and 56% of these increases. This ratio compares pooled changes across experiments; it is not the fraction of accuracy mediated by the FV.

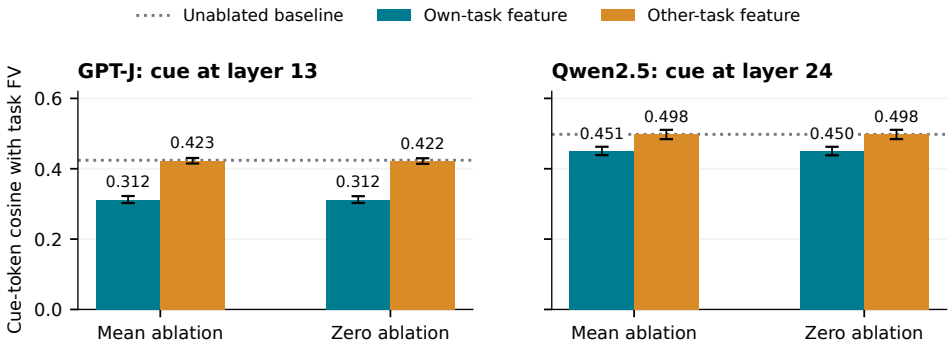

Figure 14: **Identification ablation lowers downstream execution alignment.** Own-task and other-task identification features are ablated at all demonstration-label tokens at the outputs of layers 0–27. Bars show mean cue-token cosine with the task FV at layer 13 in GPT-J and layer 24 in Qwen; dotted lines show no ablation. Error bars are 95% task-bootstrap intervals. Means weight tasks equally.

Identification steering with execution ablation. Each prompt has six in-distribution demonstration inputs whose labels are replaced by _. Every dummy slot receives $2(c + u_A)$ at the output of layer 6 in GPT-J or 12 in Qwen. In the same prompt pass, the unit task FV is zero-ablated at the final cue token at the outputs of layers 9–27; the control ablates the seeded other-family task's FV instead. The four conditions are unsteered, steering alone, steering with own-FV ablation, and steering with other-task FV ablation. We use 150 prompts per task (52 for two Qwen tasks). Completions are sampled at temperature 1 with unrestricted top-k/top-p, cut at the first newline, stripped, and scored by exact match. Each condition is sampled separately, with seeds set by a CRC32 hash of task, condition, and batch, and interventions act only while the prompt is processed.

Accuracy is averaged equally across tasks, and the pooled fraction of the steering gain that an ablation removes is

$$F_\kappa = \frac{\overline{a}_{\text{steer}} - \overline{a}_{\text{steer}+\kappa}}{\overline{a}_{\text{steer}} - \overline{a}_{\text{unsteered}}}, \qquad \kappa \in \{\text{own}, \text{other}\},$$

where $\overline{a}$ is task-averaged accuracy under steering alone, steering plus own- or other-task FV ablation, and no steering. It is 0.9995 and 0.9677 for own-task ablation in GPT-J and Qwen, and 0.3925 and 0.5075 for other-task ablation. Bootstrap intervals resample tasks 10,000 times (seed 20260923), keeping condition accuracies paired by task. The separately sampled steering-only accuracies (38.7% and 54.1%) closely reproduce the identification steering results of Figure 4 (38.1% and 54.3%).

Other-task FV overlap. Other-task ablation also removes part of the steering gain, and we test whether this reflects overlap between the two FVs. For tasks with a steering gain above 0.1 (59 GPT-J, 91 Qwen), we relate the fraction of gain removed by the other-task ablation, f_A, to the overlap $o_A = \cos(v_A, v_{A'})$, where A' is the control task paired with A. The Spearman correlation is 0.608 in GPT-J and 0.573 in Qwen, with mean overlaps of 0.379 and 0.609. An ordinary least-squares

fit $f_A = \beta_0 + \beta_1 o_A^2 + \epsilon_A$, bootstrapped over tasks, extrapolates to a removed fraction of 0.192 ($[0.080, 0.303]$) in GPT-J and -0.100 ($[-0.260, 0.060]$) in Qwen at zero overlap; both values lie outside the observed overlap range. The nine GPT-J pairs with cosine below 0.2 lose 9.3% of their steering gain on average, and no Qwen pair in this set falls below that cutoff. The other-task control's effect is therefore largely explained by shared FV components.

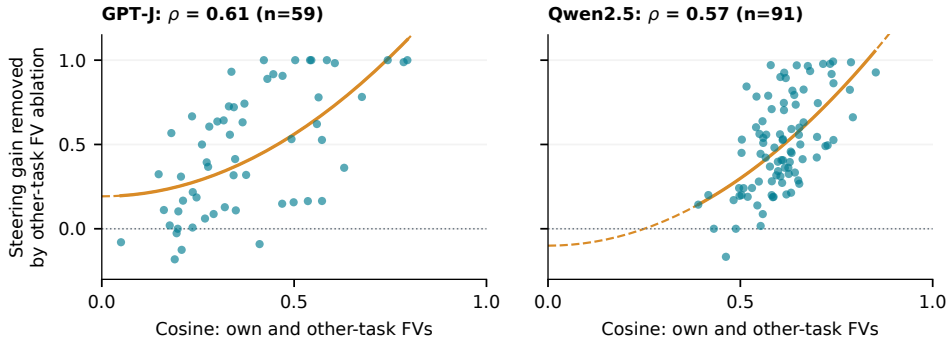

Figure 15: **Other-task ablation effects increase with FV overlap.** Each point is a task with steering gain above 0.1. The ordinate is the task's gain fraction removed by other-task FV zero ablation. The curve fits a linear model in squared cosine; dashed portions extrapolate outside the observed cosine range.

Identification-feature regression. The identification and execution features are nearly orthogonal: $\cos(u_A, v_A)$ has mean 0.05 in GPT-J and 0.07 in Qwen (range -0.06 to 0.20), against 0.00 for unrelated task pairs. The input is the task's identification feature u_A (label direction removed), averaged over prompts and over the extraction band (layers 5–7 for GPT-J, 11–13 for Qwen); the target is the task's mean FV. We fit on 55 GPT-J or 77 Qwen training tasks and evaluate on 14 or 19 held-out tasks, centering inputs and targets on training means. The per-layer label directions $\hat{g}_\ell$ are estimated from training tasks only; re-estimating them over the full task pool instead changes held-out R^2 by less than 0.001 in either model. A dual ridge solver chooses the regularization from 17 log-spaced values between 10^{-2} and 10^6 by a leave-one-out residual criterion on the training-centered matrices, selecting 0.01 in both models; centering and label-direction estimation are not recomputed inside each fold. We report the pooled score

$$R^2 = 1 - \frac{\sum_{A \in \mathcal{T}_{\text{test}}} \|v_A - \hat{v}_A\|^2}{\sum_{A \in \mathcal{T}_{\text{test}}} \|v_A - \bar{v}_{\text{test}}\|^2},$$

where $\mathcal{T}_{\text{test}}$ is the set of held-out tasks and $\bar{v}_{\text{test}}$ the mean of their FVs. To measure split sensitivity, we repeat the full fit, including the training-only label direction and the regularization choice, on ten random task partitions of the canonical sizes (seeds 0–9). Held-out R^2 ranges from 0.502 to 0.683 in GPT-J (mean 0.602, sample standard deviation 0.060) and from 0.443 to 0.630 in Qwen (0.534, 0.070); the training-mean predictor stays at or below -0.053 on every split. The canonical splits (0.641 and 0.588) lie within these ranges. With the training-mean FV as reference instead, it is 0.669 for GPT-J and 0.608 for Qwen. For comparison (Figure 16), the training-mean predictor, a fitted Procrustes partial isometry, and that isometry with a fitted global scale score -0.084, 0.457, and 0.588 in GPT-J and -0.053, 0.455, and 0.544 in Qwen, all with the test-mean reference. The partial isometry acts only on the span of the training inputs, so it does not establish a full-space rotation between the two representations.

Per-prompt regression. Here the inputs are individual residual activations at the last token of the tenth demonstration label, paired by prompt with their per-prompt FVs. For each extraction layer, we fit an affine ridge map on each model's training tasks. The regularization is chosen from 19 values between 10^{-1} and 10^8 by five-fold task-level cross-validation (seed 42) on summed squared error, and the map is then refitted on all training rows. The selected value is 10^4 for GPT-J at layer 6 and about 3.16×10^4 for Qwen at layer 12. To score task centroids, we average predictions within each held-out task and compare them with the task FV. Pooled R^2 (defined above) is 0.633 for GPT-J and 0.577 for Qwen. Coordinate-mean R^2, which averages coordinate-wise scores with the held-out

mean as reference, is 0.577 and 0.527, against 0.578 and 0.526 for the identification-feature map on the same metric. Predicting individual prompt FVs gives coordinate-mean R^2 of 0.384 and 0.278. Across extraction layers, the centroid score peaks at 0.636 (layer 15) in GPT-J and 0.554 (layer 22) in Qwen; these maxima are descriptive, not independently validated layer choices. Maps are always fitted within a single model.

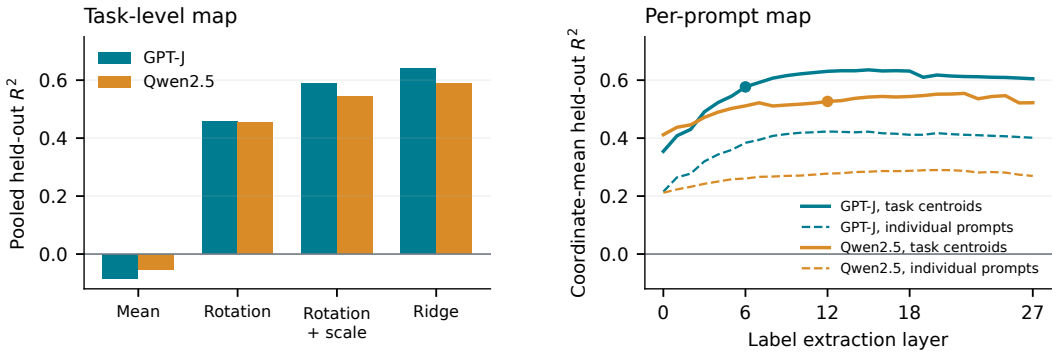

Figure 16: **Matched linear-prediction experiments.** Left: task-level prediction of held-out FVs from identification features (training-only label direction), pooled R^2, for the training-mean predictor, a Procrustes partial isometry, the isometry with a global scale, and ridge. Right: per-prompt affine regression across extraction layers, scored on held-out task centroids (solid) or individual prompt FVs (dashed) with coordinate-mean R^2; dots mark the identification layers (6 for GPT-J, 12 for Qwen).

Association with task accuracy. For each task and shot count $n \in \{0, \ldots, 6\}$, we average execution alignment $\cos(z_{\ell,\mathrm{cue}}, v_A)$ over prompts and compare it with sampled exact-match accuracy in the same setting, giving one Spearman correlation per task across the seven settings. Figure 17 uses layer 13 for GPT-J and layer 24 for Qwen. Averaging alignment over layers 9–20 instead gives between-task correlations at six shots of -0.36 and -0.24, so across tasks, higher alignment does not go with higher accuracy at this readout. Because demonstration count changes both context and accuracy, these correlations alone do not show that higher alignment causes higher accuracy.

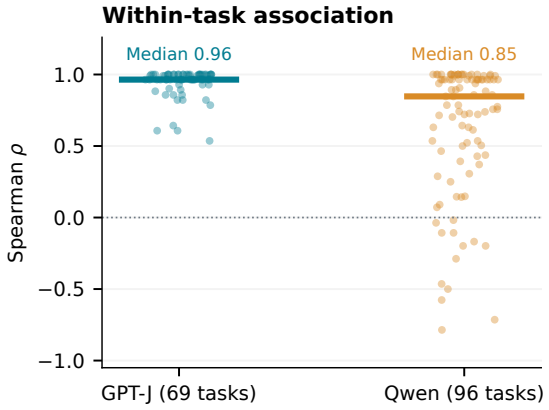

Figure 17: **Within-task association between execution alignment and accuracy.** Each dot represents one task's Spearman rank correlation across zero to six real demonstrations, using mean execution alignment $\cos(z_{\ell,\mathrm{cue}}, v_A)$ and sampled exact-match accuracy at each shot count. Bold horizontal lines mark medians (0.96 for GPT-J and 0.85 for Qwen). Correlations are positive for 69/69 GPT-J tasks and 83/96 Qwen tasks. Horizontal jitter separates overlapping dots and has no quantitative meaning.

E CODING-STYLE SETUP AND STEERING PROTOCOLS

Corpus and prompt construction. Of the 60 candidate convention families, 53 are retained. Six fail the selection rule: `css_shorthand`, `py_ternary`, `early_return`, `py_bool_prefix`, `c_braces`, and `py_abbrev`. We also exclude `js_hungarian`, whose alternative rendering adds type-encoding Hungarian prefixes: no exact rewrite rule can produce it, and generated rewrites failed review. Each retained family has 200 paired documents. The families cover formatting, naming, comments, literals, and programming idioms or dialect choices, so the suite is broader than purely cosmetic formatting. Each pair gives a programming task and two renderings meant to express the same solution in different conventions. A prompt consists of `Task:`, the task description, a language header such as `JavaScript:`, and the code prefix, with no chat template and no instruction about which convention to use. The main-text example is an excerpt from the quote-style family; its cue token is the token containing the final equals sign.

Prompts are cut at the cue token before the $(k+1)$st scored choice, for $k = 0, \ldots, 4$. Here k counts scored convention choices, not every style-bearing token, and incidental evidence can appear earlier in the prefix, so $k = 0$ means before the first scored choice rather than a style-free prompt. We keep a family when both convention-specific $k = 4$ success rates are at least 0.30, focusing on conventions the model can follow in context. As for the simple tasks, the screen keeps the conventions the model can learn in context; it uses the full document pool. Table 4 lists all 60 candidate families with one example of each rendering.

Table 4: All 60 candidate coding-convention families, with one example of each rendering from a judge-approved document. A and B are the paired renderings (the generated document and its rewritten twin); the labels do not imply a preferred style. †: not in the 53-family analysis pool (six fail the $k = 4$ selection rule; `js_hungarian` is excluded by design). Examples for `c_braces`, `css_shorthand`, `early_return` and `py_bool_prefix`, which were excluded before the corpus review, come from unreviewed documents. ↩: line break; ␣: space; →: tab.

Family	Conventions	Example (A / B)
bash_subst	Bash: $(...) substitution (A) vs. backtick substitution (B).	A: `tmp=$(mktemp)` B: `tmp=`mktemp``
bash_test	Bash: [[ ]] tests (A) vs. [ ] tests (B).	A: `[[ -w "$log" ]]` B: `[ -w "$log" ]`
c_braces†	C: K&R braces (A) vs. Allman braces (B).	A: `int main() {` B: `int main()` ↩ `{`
css_shorthand†	CSS: long hex / 0px (A) vs. short hex / 0 (B).	A: `top: 0px;` B: `top: 0;`
c_comment_style	JavaScript: // comments (A) vs. /* */ comments (B).	A: `// Get today's date.` B: `/* Get today's date. */`
hex_constants	JavaScript: decimal constants (A) vs. hexadecimal constants (B).	A: `Read: 16,` B: `Read: 0x10,`
js_arrow	JavaScript: arrow functions (A) vs. function expressions (B).	A: `tremors.forEach(t => {` B: `tremors.forEach(function(t) {`
js_camel_snake	JavaScript: camelCase identifiers (A) vs. snake_case identifiers (B).	A: `let bestIndex = 0;` B: `let best_index = 0;`
js_func_pascal	JavaScript: camelCase function names (A) vs. PascalCase function names (B).	A: `red: toCount(red),` B: `red: ToCount(red),`
js_hungarian†	JavaScript: plain identifiers (A) vs. Hungarian notation (B).	A: `let isEven;` B: `let bIsEven;`

Continued on next page

Table 4 (continued)

Family	Conventions	Example (A / B)
js_quotes	JavaScript: double-quoted strings (A) vs. single-quoted strings (B).	A: `do: "c",` B: `do: 'c',`
js_semicolons	JavaScript: semicolons (A) vs. no semicolons (B).	A: `let i = 0;` B: `let i = 0`
js_strict_eq	JavaScript: === / !== (A) vs. == / != (B).	A: `if (w === h) {` B: `if (w == h) {`
js_template	JavaScript: template literals (A) vs. string concatenation (B).	A: `return ` ${value}`;` B: `return "" + value;`
js_var	JavaScript: const/let (A) vs. var (B).	A: `let i = 0;` B: `var i = 0;`
php_array	PHP: [] arrays (A) vs. array() arrays (B).	A: `$clean = [];` B: `$clean = array();`
blank_lines	Python: two blank lines between defs (A) vs. one blank line (B).	A: `)` ↩ ↩ ↩ `def validate(vessel):` B: `)` ↩ ↩ `def validate(vessel):`
comma_space	Python: space after comma (A) vs. no space after comma (B).	A: `(-2, 1),` B: `(-2,1),`
comment_case	Python: capitalized comments (A) vs. lowercase comments (B).	A: `# Extract the last digit.` B: `# extract the last digit.`
comment_language	Python: English comments (A) vs. Spanish comments (B).	A: `# If it is, return its index` B: `# Si lo es, devolver su índice`
docstring_quotes	Python: triple double quotes (A) vs. triple single quotes (B).	A: `"""Render a dictionary as text.` B: `'''Render a dictionary as text.`
docstring_style	Python: Google docstrings (A) vs. NumPy docstrings (B).	A: `Returns:` B: `Returns` ↩ `-----` ↩ `list`
early_return†	Python: guard clauses (A) vs. nested if/else (B).	A: `if n < 2: return -1` B: `if n >= 2:`
float_literals	Python: full float literals (A) vs. abbreviated float literals (B).	A: `'q': 0.1,` B: `'q': .1,`
line_wrap	Python: wrapped long calls (A) vs. single long lines (B).	A: `def validate_` `ski(` ↩ `index,` ↩ `ski,` ↩ `):` B: `def validate_ski(index, ski):`
num_separators	Python: plain numeric literals (A) vs. underscore digit separators (B).	A: `h = 65536` B: `h = 65_536`
operator_spaces	Python: spaces around operators (A) vs. no spaces around operators (B).	A: `G0 = 9.8` B: `G0=9.8`
py2_except	Python: except E as e (A) vs. except E, e (B).	A: `except OSError as err:` B: `except OSError, err:`
py2_iter	Python: range/items/keys (A) vs. xrange/iteritems/iterkeys (B).	A: `for i in range(n):` B: `for i in xrange(n):`
py2_print	Python: print() function (A) vs. print statement (B).	A: `print("")` B: `print ""`
py_abbrev†	Python: abbreviated identifiers (A) vs. spelled-out identifiers (B).	A: `err = []` B: `error = []`
py_bool_prefix†	Python: is_/has_ boolean names (A) vs. bare boolean names (B).	A: `is_two = n == 2` B: `two = n == 2`

Continued on next page

Table 4 (continued)

Family	Conventions	Example (A / B)
py_builtin_generics	Python: list[int] builtins (A) vs. typing.List (B).	A: `list[str],` B: `List[str],`
py_class_naming	Python: PascalCase class names (A) vs. snake_case class names (B).	A: `class BoxCar:` B: `class box_car:`
py_comprehension	Python: list comprehensions (A) vs. explicit loops (B).	A: `coords = [float(c) for c in p]` B: `coords = []↩for c in p:↩coords.append(float(c))`
py_const_naming	Python: UPPER_SNAKE constants (A) vs. camelCase constants (B).	A: `LATTE = 2` B: `latte = 2`
py_enumerate	Python: enumerate (A) vs. range(len()) (B).	A: `for i, ch in enumerate(text):` B: `for i in range(len(text)):`
py_fstring	Python: f-strings (A) vs. str.format (B).	A: `return f"{name}."` B: `return "{}.".format(name)`
py_indent	Python: 4-space indentation (A) vs. 2-space indentation (B).	A: `␣␣␣␣while b:↩␣␣␣␣␣␣␣␣temp = b` B: `␣␣while b:↩␣␣␣␣temp = b`
py_is_none	Python: is None (A) vs. == None (B).	A: `if n is None:` B: `if n == None:`
py_join_concat	Python: str.join (A) vs. + concatenation (B).	A: `size = 'x'.join([w, h])` B: `size = w + 'x' + h`
py_literal_ctor	Python: {} / [] literals (A) vs. dict() / list() constructors (B).	A: `run = []` B: `run = list()`
py_loop_vars	Python: single-letter loop variables (A) vs. descriptive loop variables (B).	A: `for k in set1:` B: `for key in set1:`
py_not_in	Python: x not in y (A) vs. not x in y (B).	A: `if ch not in odd:` B: `if not ch in odd:`
py_optional	Python: X | None (A) vs. Optional[X] (B).	A: `int | None` B: `Optional[int]`
py_paren_if	Python: if x: (A) vs. if (x): (B).	A: `if rush:` B: `if (rush):`
py_private	Python: underscore-prefixed private members (A) vs. no underscore prefix on private members (B).	A: `_MIN = 0` B: `MIN = 0`
py_quotes	Python: single-quoted strings (A) vs. double-quoted strings (B).	A: `DOT = '.'` B: `DOT = "."`
py_self_name	Python: self (A) vs. this (B).	A: `def die(self):` B: `def die(this):`
py_snake_camel	Python: snake_case identifiers (A) vs. camelCase identifiers (B).	A: `gc_count = 0` B: `gcCount = 0`
py_tabs	Python: spaces (A) vs. tabs (B).	A: `def is_prime(n):↩␣␣␣␣if n < 2:` B: `def is_prime(n):↩→if n < 2:`
py_ternary†	Python: conditional expressions (A) vs. if/else blocks (B).	A: `size = n if n >= 0 else 0` B: `if n >= 0:↩size = n↩else:↩size = 0`

Continued on next page

Table 4 (continued)

Family	Conventions	Example (A / B)
py_type_hints	Python: type hints (A) vs. no type hints (B).	A: `raw: str,` B: `raw,`
py_with_open	Python: with open(...) (A) vs. open() / close() (B).	A: `with open("flight_path.bin", "wb") as path_file:` B: `path_file = open("flight_path.bin", "wb")`
trailing_commas	Python: trailing commas (A) vs. no trailing commas (B).	A: `"pluto",` B: `"pluto"`
r_assignment	R: <- assignment (A) vs. = assignment (B).	A: `run <- 1` B: `run = 1`
rust_question	Rust: ? operator (A) vs. explicit match (B).	A: `let last = *hits.last()?;` B: `let last = *match hits.last() { Some(v) => v, None => return None };`
sql_bool_case	SQL: lowercase true/false (A) vs. uppercase TRUE/FALSE (B).	A: `RETURN true;` B: `RETURN TRUE;`
sql_join_style	SQL: explicit JOIN ... ON (A) vs. implicit comma join (B).	A: `FROM Posts AS P ↩ JOIN Users AS U` B: `FROM Posts AS P, Users AS U`
sql_keyword_case	SQL: uppercase keywords (A) vs. lowercase keywords (B).	A: `LEFT JOIN` B: `left join`

Generation and scoring. For each prompt, the model samples one continuation at temperature 1 (no top-p or top-k truncation, up to 48 new tokens), trimmed at a code boundary. Family-specific deterministic detectors classify which convention the continuation uses, with contextual rules for naming and comments; a continuation showing neither convention fails the style criterion. Separately, Gemini 2.5 Flash judges whether the continuation is syntactically plausible and sensibly continues the programming task, ignoring the convention under evaluation. The judge allows reasonable alternatives to the reference and incomplete statements at the sampling boundary, so acceptance measures local plausibility, not whether the complete program passes tests or solves the task. Success requires both the target convention and judge acceptance.

In-context learning curve. Figure 18 pools the 53 retained families and both conventions (200 prompts per family, convention and k, equally weighted). Success rises from 29.6% at $k = 0$ to 68.0% at $k = 1$ and 76.3% at $k = 4$. Many $k = 0$ failures are continuations that make no style choice at all (28.5% of prompts, against 35.6% that use the wrong convention and 6.3% that use the target convention but are rejected by the judge); this share falls to 10.1% by $k = 4$.

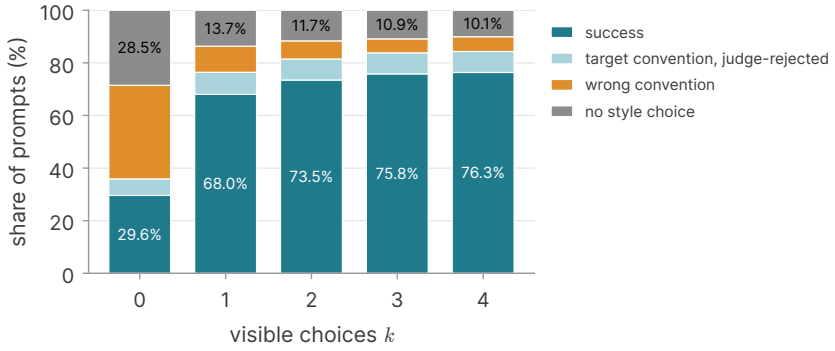

Figure 18: **Coding-style in-context learning.** Outcome shares by number of visible choices k, pooled over the 53 retained families and both conventions.

Document separation and vector estimation. A document goes to the steering test set when the CRC32 hash of its identifier is zero modulo four, which reserves about a quarter of the documents; both renderings of a document share the assignment. Vectors are estimated from the remaining training documents, keeping $k = 3$ and $k = 4$ prompts only when the sampled continuation follows the demonstrated convention and passes the judge. The two convention means use independently selected prompts, so a document need not succeed in both renderings to contribute, and the mean difference vector may partly reflect differences between the two pools of successful prompts. Identification capture averages evidence-token activations within each prompt before averaging across prompts; execution capture averages final cue-token activations. For steering, mean difference vectors keep their original magnitude rather than being unit-normalized. Each family has its own mean difference vector, and hence its own feature, at each site; no single feature is shared by all families. This mean-difference construction with signed addition is related to in-context vectors (Liu et al., 2024), which use the leading principal direction of paired activation differences concatenated across layers, steer at all layers and tokens, and restore activation norms. We instead take a difference of means at each site, intervene at specified layers and token positions, and do not restore the norm.

Evidence-token definition. Evidence tokens are the tokens expressing earlier scored convention choices, any other tokens that differ when the paired renderings are aligned, and, in comment and docstring families, the relevant comment or docstring spans. Changes in leading spaces are ignored unless whitespace is itself the convention. The task header, demonstration cues, and final cue token are never evidence. One choice can yield several evidence tokens, and identification steering adds the same mean difference vector at each of them; prefixes with no evidence positions are excluded from this intervention.

Worked example. Figure 19 shows one quote-style document in both conventions at $k = 3$, including the task header omitted from the coding-style setup. Each prompt contains the task description, the language header, and the code up to the fourth scored choice; the two prompts differ only at the six evidence tokens that carry the convention. Identification vectors average activations over these evidence tokens, where identification steering and ablation also act; execution vectors are read at the cue token, the final =. The scored continuation begins with `"error` for convention A or `'error` for convention B, and success requires the demonstrated convention and judge acceptance. Qwen's tokenizer fuses each closing quote with the following semicolon and newline, so the whole fused token counts as evidence.

Intervention settings and evaluation. Identification steering uses prompts showing three choices in the opposite convention and adds the signed mean difference vector at every evidence position. Execution steering uses prefixes before the first scored choice and adds its mean difference vector only at the final cue token. Both modify layer outputs during prefill and leave decoding unchanged. The code calls these settings layer 8 and layer 24, hidden-state indices that count the embedding output as 0; they correspond to the outputs of zero-indexed layers 7 and 23 in this paper.

The identification setting, $\alpha = 4$ at layer 7, comes from a 10-family sweep over layers and strengths. The execution setting, $\alpha = 2$ at layer 23, comes from the corresponding execution-site layer sweep. The preliminary sweeps predate the refreshed corpus and their families overlap the reported pool. On the refreshed corpus, the setting remains the best of those re-tested: execution steering with $\alpha = 4$ at layer 23 and with $\alpha \in \{2, 4\}$ at layer 25 gives pooled success of 54.2%, 54.8%, and 48.2%, against 56.0% at the canonical setting.

Steering is evaluated on 40 held-out documents per family and target convention, and the unsteered condition uses the same test context. The in-context references are ordinary $k = 3$ or $k = 4$ success rates over all 200 documents per convention, so they are descriptive comparisons rather than matched ceilings on the test documents. Pooled means weight each family and both target conventions equally. Figure intervals use 10,000 family resamples, keeping both directions together, and condition on the learned vectors and fixed intervention settings.

Identification steering raises target-convention detection from 5.1% to 70.9%, while judge acceptance falls from 88.5% to 82.2%; for execution steering, judge acceptance of the steered continuation is 78.2%. Because success requires both, higher style adherence alone does not count when the judge rejects the continuation. The ablation and cue-shift experiments below test causal roles at the two

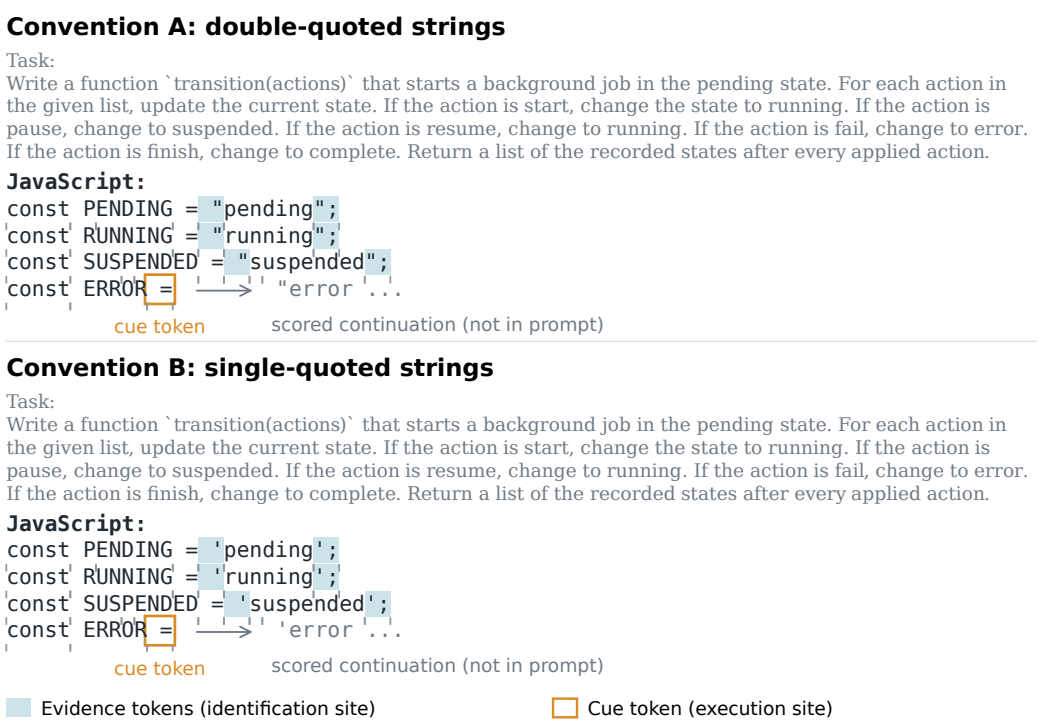

Figure 19: Worked coding-style prompt (`js_quotes`, $k = 3$) in both conventions. Teal: evidence tokens; orange box: cue token; gray: scored continuation, not part of the prompt.

Site	Target convention	Unsteered	Steered	Context reference
Identification	A	6.8%	66.2%	79.3% ($k = 3$)
Identification	B	2.0%	53.7%	72.3% ($k = 3$)
Execution	A	47.3%	61.1%	79.1% ($k = 4$)
Execution	B	12.9%	50.9%	73.6% ($k = 4$)

Table 5: **Coding-style steering in both directions.** A and B designate the paired convention labels within each family (the generated document and its rewritten twin); these labels do not imply a universally preferred or correct style.

sites and the relationship between them, and the mapping analysis tests whether the paired contrasts are predictively related.

Ablation at both sites. We use $k = 4$ prompts from the first 40 held-out documents of each of the 53 families. The headline prompts demonstrate each family's convention B (Table 4), so they test whether the intervention removes the demonstrations' effect; success requires keeping that convention and passing the judge. At each depth, we ablate the unit mean difference vector estimated from training documents. Identification ablation acts at every evidence token of all four demonstrations, at the embedding output and the outputs of all layers 0–27; execution ablation acts only at the cue token, at the outputs of all layers 0–27. All edits happen during prompt processing, and decoding is unchanged. Zero ablation removes the projection. Mean ablation replaces it with the corpus-wide mean projection, weighting both conventions' training-activation means by prompt count. The control uses one seeded other family's features and means. Averaged over families, the model adopts convention A far more often than B without demonstrations (47.3% versus 12.9% target-style success at $k = 0$; Table 5), so in this sense only convention B is learned in context. This is why the ablations use convention-B demonstrations.

The unablated baseline is 73.0% (95% interval $[69.1, 76.6]$). Table 6 reports both operators, with intervals from 10,000 paired resamples of the 53 families (seed 20260923). With convention-A

demonstrations, execution success is 80.6% without ablation, 64.7% after own-family zero ablation, and 60.6% after mean ablation; other-family controls keep 80.0% and 80.3%.

Intervention	Identification	Execution
No ablation	73.0 [69.1, 76.6]	73.0 [69.1, 76.6]
Own family, zero	31.0 [24.2, 37.9]	40.5 [34.5, 46.3]
Own family, mean	33.9 [27.2, 41.1]	40.8 [34.1, 47.7]
Other family, zero	73.8 [69.8, 77.5]	71.8 [67.8, 75.5]
Other family, mean	73.6 [69.6, 77.3]	74.1 [69.8, 78.0]

Table 6: **Coding-style ablation.** Success percentages with 95% family-bootstrap intervals, under convention-B demonstrations.

Figure 20 compares own-family and other-family zero-ablation drops for each family. Counting exact trials, 32 of 53 families at each site show an own drop of at least 0.20 with a control drop below 0.10. The identification families with an own drop below 0.10 are `comment_language`, `c_comment_style`, `py_private`, `py_literal_ctor`, `py_optional`, `py_tabs`, and `py_not_in`; at the execution site they are `sql_keyword_case`, `bash_test`, `comment_language`, `py_join_concat`, `py_fstring`, `js_template`, `py_enumerate`, `operator_spaces`, and `sql_join_style`. How much a family depends on the tested directions therefore varies.

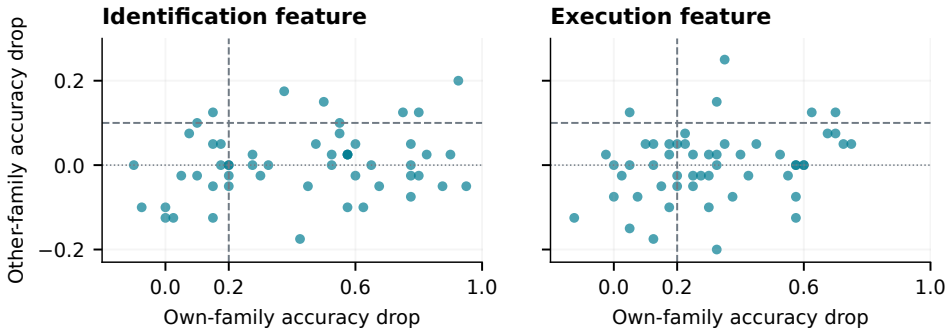

Figure 20: **Family-specific ablation effects.** Each point is a family. Dashed guides mark own drop 0.20 and other-family drop 0.10; specificity lies below and to the right.

Identification-to-execution cue shifts. This is the controlled 53-family experiment: $k = 3$ prompts demonstrating convention A, from 40 held-out documents per family. During prompt processing, we add $-\alpha d_f^{\mathrm{id}}$, which steers toward convention B, the training evidence-token mean difference vector, at every evidence token at the layer 7 output. At the layer 23 output, let $\Delta z_\alpha = z_\alpha - z_0$ be the resulting change at the cue token, and $\Delta z_{\mathrm{alt}} = z_{\mathrm{alt}} - z_0$ the change when the real demonstrations use convention B instead. Controls replace the identification vector with another family's (seeded, same language when possible) or with a random direction of equal norm. We measure

$$\cos(\Delta z_\alpha, -d_f^{\mathrm{exec}}), \qquad \frac{\langle \Delta z_\alpha, \hat{d}_f^{\mathrm{exec}} \rangle}{\langle \Delta z_{\mathrm{alt}}, \hat{d}_f^{\mathrm{exec}} \rangle}.$$

Cosines are averaged over prompts within each family. Projection ratios take the median over prompts within each family and then the mean across families; they are not fractions of behavioral mediation. Figure 21 shows both statistics across strengths, with 95% percentile intervals from 10,000 family resamples (seed 20260923). The first-token result (67.4% of prompts) is a pairwise preference: the first token of convention B has a higher log-probability than that of convention A. It is not vocabulary-wide top-1 accuracy or generated-code success. At $\alpha = 2$, own-family cosine exceeds both controls in 52/53 families and is at least 0.5 in 42/53; the real demonstration-induced shift reaches 0.721 (at least 0.5 in 51/53).

Mapping between paired, normalized contrasts. The mapping analysis uses the 53-family pool and five family partitions, nominally 66/34 splits carried over from an earlier pool; Table 7 gives

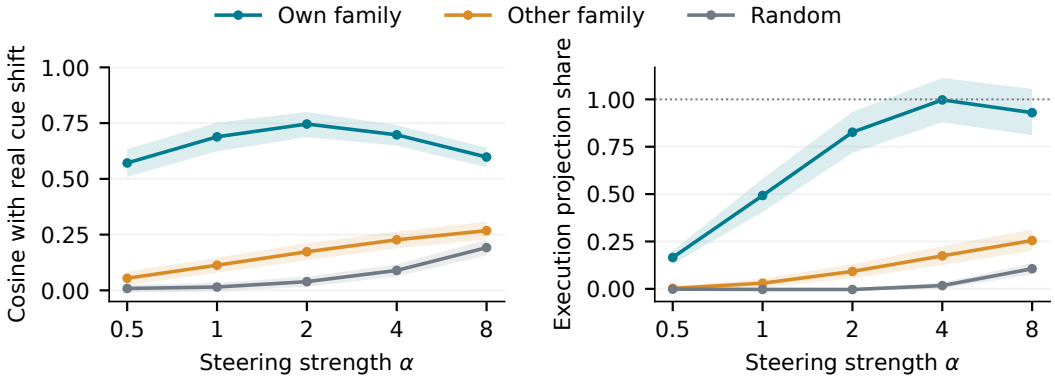

Figure 21: **Identification steering changes execution-site activations.** Alignment with the execution feature $-d_f^{\text{exec}}$ (left) and reproduced execution-direction projection (right). Bands: 95% family-bootstrap intervals.

their current training and held-out counts. A held-out family's prompts are never used for fitting or regularization selection. These splits are by family, unlike the within-family document split used for steering.

For family f, document–choice pair j, and site $q \in \{\text{id}, \text{exec}\}$, let $d_{fj}^q = z_{fj,\text{A}}^q - z_{fj,\text{B}}^q$ and $\hat{d}_{fj}^q = d_{fj}^q / \|d_{fj}^q\|$. Identification activations are averaged over evidence tokens at layer 7, and execution activations are taken at the final cue token at layer 23. The two renderings are matched by document identifier and k, and all available pairs at $k = 3, 4$ are included whether or not the sampled continuations succeeded. The five fits contain 13,600–14,800 training pairs. Within each split, we fit a shared affine ridge predictor on training pairs $(x, y) = (\hat{d}_{fj}^{\text{id}}, \hat{d}_{fj}^{\text{exec}})$, centered on prompt-pair training means:

$$\widehat{y}(x) = \bar{d}_{\text{train}}^{\text{exec}} + W_{\text{code}}\big(x - \bar{d}_{\text{train}}^{\text{id}}\big),$$

where $\bar{d}_{\text{train}}^q$ is the mean of $\hat{d}_{fj}^q$ over training pairs. The ridge grid is $10^{-4}, 10^{-3}, \ldots, 10^3$. Leave-one-family-out validation maximizes pooled R^2 on the omitted prompt-pair targets, with means recomputed in each fold; every split selects $\lambda = 10$.

Split	Training families	Held-out families	$R^2_{\text{train-mean}}$
1	37	16	0.223
2	36	17	0.248
3	35	18	0.267
4	34	19	0.191
5	35	18	0.262
Mean ± sample standard deviation			0.238 ± 0.031

Table 7: **Coding-style mapping across five nominal 66/34 splits.** Scores use the training-family mean execution centroid as reference. The splits overlap, so the displayed standard deviation describes split variability and is not a confidence interval from independent replications. Means and standard deviations summarize the saved per-split scores.

Evaluation aggregates at the family level instead. For each family, we form the centroid $\hat{d}_f^q$, the mean of d_{fj}^q over J_f, the set of document–choice pairs of family f, normalized once to unit length, and set $x_f = \hat{d}_f^{\text{id}}$ and $y_f = \hat{d}_f^{\text{exec}}$; these normalized family centroids are the vectors on which the map is evaluated. The prediction $\widehat{y}_f = \widehat{y}(x_f)$ is not renormalized before squared-error scoring. The main-text score is

$$R^2_{\text{train-mean}} = 1 - \frac{\sum_{f \in \mathcal{F}_{\text{test}}} \|y_f - \widehat{y}_f\|^2}{\sum_{f \in \mathcal{F}_{\text{test}}} \|y_f - \bar{y}_{\text{train}}\|^2}, \qquad \bar{y}_{\text{train}} = \frac{1}{|\mathcal{F}_{\text{train}}|} \sum_{f \in \mathcal{F}_{\text{train}}} y_f,$$

where $\mathcal{F}_{\text{train}}$ and $\mathcal{F}_{\text{test}}$ are the split's training and held-out families. Its reference $\bar{y}_{\text{train}}$ is the mean of the normalized training-family execution centroids, not the prompt-pair mean used as the fitted intercept. A constant predictor of $\bar{y}_{\text{train}}$ therefore scores zero by construction. Because only 53 families are available, we use the training-mean predictor as the reference rather than the held-out mean, which would be an unfair comparison with so few families.