AlgoTrace: Algorithmic Primitives and Compositional Geometry of Reasoning in Language Models

Samuel Lippl [*1] **Thomas McGee** [*2] **Kimberly Lopez** [3] **Ziwen Pan** [3] **Pierce Zhang** [3] **Salma Ziadi** [3] **Oliver Eberle** [*4 5] **Ida Momennejad** [*6]

Abstract

How do inference time and latent computations enable large language models (LLMs) to solve multi-step reasoning problems? We introduce AlgoTrace, a framework for tracing and steering algorithmic operations in the model latent space for multi-step reasoning. We operationalize primitives by clustering latent activations of the model when solving four benchmarks: Traveling Salesperson Problem (TSP), 3SAT, AIME, and Graph Navigation. We annotate the clusters using their corresponding tokens in the reasoning trace. We then apply function vector methods to extract primitive vectors as reusable compositional building blocks of reasoning. We find that a) injecting a primitive vector into models (Phi, Qwen, Llama) elicits the associated algorithmic operation in the reasoning trace, b) injecting primitives can steer behavior across tasks, c) primitive vectors can be composed through algebraic operations, revealing a geometric logic in activation space, and d) a fine-tuned model exhibits improved composition of primitives (Phi-4-Reasoning vs. Phi-4). These findings demonstrate that LLM reasoning can be understood as a walk through algorithmic primitives in the latent space governed by compositional geometry. These primitives transfer across tasks, and reasoning finetuning strengthens algorithmic generalization and composition across domains.

[*]Equal contribution [1]Center for Theoretical Neuroscience, Columbia University, New York, USA [2]Department of Psychology, University of California Los Angeles, Los Angeles, USA [3]Research in Industrial Projects for Students (RIPS) program, University of California Los Angeles, Los Angeles, USA [4]Technische Universität Berlin, Berlin, Germany [5]BIFOLD-Berlin Institute for the Foundations of Learning and Data, Berlin, Germany [6]Microsoft Research NYC, New York, USA. Correspondence to: Ida Momennejad <idamo@microsoft.com>, Samuel Lippl <samuel.lippl@columbia.edu>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Inference time compute has remarkably improved reasoning in large language models (LLMs), and reasoning-finetuned models like Phi-4-Reasoning substantially outperform their base counterparts on reasoning tasks they were not directly trained on (Abdin et al., 2025). However, the extent to which LLMs, especially reasoning-finetuned models, can learn generalized algorithmic capacities remains poorly understood (Eberle et al., 2025). While recent advances in mechanistic interpretability (Todd et al., 2024) point in this direction, it's unclear whether LLMs acquire generalized representations of algorithmic primitives (Huh et al., 2024), whether these primitives are geometrically organized in representation space, similar to brains (Fascianelli et al., 2024), and whether LLMs solve reasoning tasks by composition of generalized algorithmic primitives, and through component reuse across tasks and models (Merullo et al., 2024). This presents a unique opportunity to understand the algorithmic basis of LLM reasoning, improvements in compositional generalization after finetuning, and the impact of finetuning on chain-of-thought reasoning (Lobo et al., 2025).

This work aims to understand how algorithmic primitives are generalized and composed in the model's latent space to enable complex reasoning. The work addresses three fundamental questions: (1) What are the basic algorithmic primitives that language models use in specific tasks, and across reasoning domains? (2) Do these primitives compose geometrically in neural activation space? (3) How do reasoning-enhanced models differ from base models in their primitive usage and composition? We introduce AlgoTrace, a framework for multi-level algorithmic understanding and steering of model reasoning. We first extract algorithmic primitives by clustering internal representations and introducing a novel LLM-based automated pipeline for annotating reasoning traces. We then apply function vector methods to extract primitive vectors from the models' internal representations. We induce and steer algorithmic primitives by injecting primitive vectors across the layers. To identify both task-specific and generalized algorithmic primitives, we apply the approach across domains: Travel-

Table 1. Summary of Key Terms and Definitions

Term	Definition	Example
Reasoning	A walk through latent space composing operations to move from prompt to response and solving the task	Composing the steps to solve Traveling Salesperson Problem (TSP)
Algorithmic Primitive	A minimal computational operation observed in a reasoning process	Retrieving the nearest neighbor in TSP
Algorithmic Tracing	The process of identifying relevant primitives for implementing a particular reasoning process	TSP requires distance calculation, nearest-neighbor selection, path comparison, etc
Primitive Vector	A direction in activation space that can be injected into the residual stream to reliably induce a primitive	A vector that causes the model to use more nearest-neighbor heuristics when injected
Primitive Induction	Modifying behavior by injecting a primitive vector	Injecting a nearest-neighbor primitive vector

ing Salesperson Problem, 3-SAT, AIME, and Graph Navigation. Finally, to investigate compositional generalization as a result of finetuning, we compare the algorithmic performance of a base model and its reasoning-finetuned counterpart, Phi-4 and Phi-4-Reasoning (Abdin et al., 2025). This multi-level framework allows us to understand the geometry and compositionality of algorithmic primitives in LLM reasoning, and use them to steer model behavior.

Our contributions include: (1) the systematic identification of cross-domain algorithmic primitives in LLMs, (2) a geometric framework for understanding primitive composition through vector arithmetic, (3) novel methodology linking explicit reasoning behaviors to internal mechanisms, (4) mechanistic evidence for compositional generalization following finetuning LLMs for reasoning, and (5) evaluation of cross-task primitive transfer and generalization.

Financial Conflict of Interest Disclosure. IM is employed by Microsoft Research (MSR), which developed the Phi models studied in this paper. SL and TM interned at MSR.

2. Related Work

Interpretability for Transformers. With increasing scale and complexity of LLMs, the challenge of understanding their predictions has become an important research direction in explainable AI and interpretability research. This has specifically targeted the analysis of internal model structure and feature relationships (Geiger et al., 2022; Eberle et al., 2022; Schnake et al., 2022), as well as representations and manifolds (Kornblith et al., 2019) including concepts (Chormai et al., 2024). Mechanistic interpretability (Sharkey et al., 2025) has focused on the extraction of circuits (Olah et al., 2020; Wang et al., 2023), feature descriptions (Hernandez et al., 2022), and causally effective representations such as function vectors (Todd et al., 2024). Combined analysis of individual neurons (Gurnee & Tegmark, 2024; Templeton et al., 2024), attention scores (Voita et al., 2019; Clark et al., 2019), and task-specific attention heads (Vig & Belinkov, 2019; McDougall et al., 2024) have further deepened our understanding of internal model processing. In parallel, free-text and chain-of-thought explanations (Turpin et al., 2023; Camburu et al., 2018; Huang et al., 2023; Madsen et al., 2024) consider the model's own natural language explanation of what it is doing and why. Gradient-based feature attributions have further enabled scaling the localization and analysis of relevant features in LLMs (Ali et al., 2022; Jafari et al., 2024).

Function Vectors and Behavioral Interventions. LLMs' contextualized representations such as function vectors (Todd et al., 2024) and in-context task vectors (Hendel et al., 2023; Yang et al., 2025) are shown to trigger execution of associated tasks. A broader application of such steering vectors include persona vectors (Chen et al., 2025) and representations of truthfulness (Marks & Tegmark, 2024; Bürger et al., 2024). Probing has identified associations between internal representations and features of interest (Conneau et al., 2018; Hewitt & Manning, 2019). This has also been extended to the analysis of reasoning strategies, e.g., by identifying probes involved in solving grade-school math problems (Ye et al., 2025). These analyses provide some first methods to understand how complex representational geometries and manifolds drive predictions in modern models. Related work from neuroscience shows that task-related variables are encoded in neural geometries in a format that supports generalization to novel situations (Bernardi et al., 2020). Monkey neuroscience research has linked compositional generalization in neural representational geometry with behavioral performance (Fascianelli et al., 2024). These findings suggest that function vector methods can be used to study how compositional generalization may guide algorithmic steering of LLMs, providing a conceptual starting point for our work.

Algorithmic Evaluation and Steering of Language Models. Multi-step planning and graph navigation are standard benchmarks for structured reasoning (Fatemi et al., 2024), yet most LLMs perform poorly and fail to generalize with growing graph complexity (Momennejad et al., 2023). Although efficient algorithms exist, recent evalua-

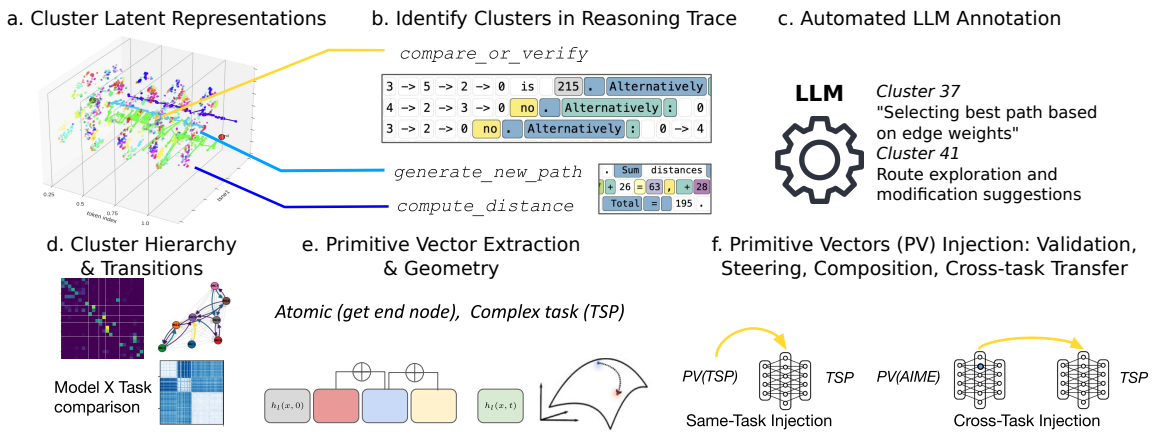

Figure 1. AlgoTrace for Algorithmic Tracing & Steering: Primitive Extraction & Evaluation. We trace algorithmic primitives by **a** clustering latent representations, **b** identifying corresponding tokens in the reasoning traces, **c** annotating them using an automated LLM pipeline, **d** meta-clustering primitives, identifying sequential transition trends among clusters, and comparing cluster similarity across models and tasks. Once we identify primitives, **e** we extract associated primitive vectors from top heads, and **f** use causal patching to validate, explore the compositional geometry and cross-task transfer of primitives.

tions suggest that LLMs rely on policy-dependent heuristics rather than explicit search strategies (Eberle et al., 2025). Complementary works have aimed to map a high-level causal model to an internal realization by an LLM (Geiger et al., 2024), and prompt models to generate sets of abstract hypotheses about the tasks to improve inductive reasoning performance (Wang et al., 2024). Recent analysis of chain-of-thought outputs has uncovered sentence-level relationships, offering the localization of salient reasoning steps (Bogdan et al., 2025). Existing work either targets high-level behaviors such as politeness, creativity, or honesty using relatively coarse vector representations to steer behavior (Chen et al., 2025), or focuses on very simple input-output mapping (Todd et al., 2024). Our work focuses on algorithmic steering, targeting computational primitives and their compositions, aiming at a deep understanding of the model's internal algorithmic phenotypes and applications in future algorithmic finetuning.

3. Experimental Setup: Models and Tasks

Models. We evaluated decoder-only Transformers on multi-step reasoning tasks. Our analyses focused on Phi-4-Base (Abdin et al., 2024) and the reasoning-specialized variant Phi-4-Reasoning (Abdin et al., 2025), with additional validation from Llama-3-8B (Dubey et al., 2024) and different variants of Qwen3 (Qwen Team, 2026).

Tasks and Data Collection. Four multi-step reasoning setups were tested. In selecting the tasks, we were guided by three criteria: tasks need to be 1) multistep, 2) related to real world reasoning like navigation and planning, and 3) formal enough to allow comparison of our insights with known solutions. We primarily focused on the Traveling Salesperson Problem (TSP), an NP-hard benchmark. Notably, Phi-4-Reasoning exhibited improved performance on TSP despite not being finetuned for this task (Abdin et al., 2025). We also investigated the 3-SAT and AIME Mathematical Olympiad benchmarks for the same reasons. Finally, we investigated Graph Navigation tasks used in previous LLM reasoning research (Momennejad et al., 2023; Eberle et al., 2025) (see App. B for task prompts). Our setups demand planning, search, and verification. TSP and Graph Navigation require graph optimization via planning, path generation, search, comparison, and verification. 3-SAT is another NP-hard problem that tests flexible algorithmic problem solving. Finally, AIME requires complex mathematical reasoning and multi-step computation. Together, these setups enable us to examine algorithmic primitives that are task-specific and ones that generalize across tasks. We collected reasoning traces and associated residual stream vectors across 100+ examples per task, with systematic variation in problem complexity.

4. Methodology

4.1. Definitions and Notation

Definition: Algorithmic Primitive. We define *algorithmic primitive* as a minimal computational operation observed in a reasoning process (Eberle et al., 2025) (e.g. TSP), such as retrieving the nearest neighbor or computing a distance. Primitives can be identified both in explicit reasoning traces (e.g. reasoning steps the model produces in the output) and in internal activations (e.g. clusters of token representations, or attention patterns).

Definition: Algorithmic Tracing. We define *algorithmic tracing* as the process of identifying all relevant primitives

for implementing a particular reasoning process. By targeting the computational building blocks rather than just behavioral outcomes, algorithmic tracing helps us better understand how and why a model solves a particular task and what other tasks it might be able to solve.

Definition: Primitive Vector. We define primitive vectors (following the function and steering vector literature (Todd et al., 2024)) as vectors that can be injected into the residual stream to reliably induce a particular primitive. For a given model input x, which is given by the concatenation of both the input prompt and the model response, and transformer layer ℓ, let the residual stream activation at token t be $h_\ell(x,t) \in \mathbb{R}^{d_\ell}$. For a given primitive p, the *primitive vector* $v_\ell^{(p)} \in \mathbb{R}^{d_\ell}$ is a direction in activation space that increases the expression of the primitive p. Here we extract primitive vectors using the function vector approach (Todd et al., 2024) (see Section 4.2).

Definition: Primitive Induction for Algorithmic Steering and Causal Evaluation. Given a primitive vector injection strength $\alpha \in \mathbb{R}$, we can intervene on a representation by adding (or patching) a primitive vector: $\tilde{h}_\ell(x,t) = h_\ell(x,t) + \alpha\, v_\ell^{(p)}$, which increases the probability of expressing p when $\alpha > 0$, and decreases it when $\alpha < 0$. We refer to this procedure as *algorithmic steering* or *induction*.

Algebraic Operations on Primitive Vectors. Assuming compositional geometry, primitive vectors can be combined through simple algebraic operations in activation space. For primitives p and q at layer ℓ, additive and subtractive composition within a layer are defined as $v_\ell^{(p\oplus q)} \approx w_p\, v_\ell^{(p)} + w_q\, v_\ell^{(q)}$, $v_\ell^{(p\ominus q)} \approx w_p\, v_\ell^{(p)} - w_q\, v_\ell^{(q)}$. Scalar modulation (with varying strength) is defined as $v_\ell^{(\alpha p)} = \alpha\, v_\ell^{(p)}$.

Primitive Transfer: Cross-Task Generalization. We test primitive generalizability by examining whether primitives identified in one domain transfer to others; specifically by examining whether injecting primitives extracted from task 1 has a predictable effect on model performance in task 2. We formalize cross-task transfer as follows. Let $p \in \mathcal{P}_{T_1}$ denote a primitive extracted from source task T_1, with vector $v_{\ell,T_1}^{(p)} \in \mathbb{R}^{d_\ell}$. When evaluating target task T_2, we inject this vector into the residual stream: $\tilde{h}_\ell(x_{T_2},t) = h_\ell(x_{T_2},t) + \alpha\, v_{\ell,T_1}^{(p)}$. If this intervention increases the activation $a_{\ell,T_2}^{(p)}(x_{T_2},t) = \langle v_{\ell,T_1}^{(p)}, h_\ell(x_{T_2},t)\rangle$ and increases the associated behavioral hallmark in task T_2, we denote successful transfer as $p \in \mathcal{P}_{T_1} \rightsquigarrow p \in \mathcal{P}_{T_2}$.

4.2. Algorithmic Primitive Extraction & Evaluation

Step 1. Primitive Identification: Geometric Clustering of Latent Representations. We hypothesize that distinct algorithmic primitives are reflected in higher dissimilarities in the model's internal representations, and therefore apply k-means clustering (Lloyd, 1982) to these representations while processing complex self-generated reasoning traces. We chose the k-means algorithm as it provided suitable results on discovering concepts in latent LLM representations in recent investigations (Hawasly et al., 2024; Petukhova et al., 2025; Kopf et al., 2025). For all clustering analyses, we extracted representations from layer 17 of Phi-4-Base and used $k = 50$ clusters (we justify these choices in App. G). We fit separate models to TSP, 3-SAT, AIME, and GraphNav and also fit a joint model to TSP+AIME.

Step 2. Primitive Identification: Mapping Clusters to Associated Reasoning Traces. We analyze the primitives associated with the different clusters revealed in step 1 by analyzing the tokens associated with a particular cluster and the context surrounding them. This lets us categorize algorithmic strategies, verification behaviors, and metacognitive patterns and provides direct insight into the models' reasoning processes. To make this process more automatable, we additionally develop an **LLM-driven pipeline for automated labeling** of the reasoning traces associated with different clusters (App. C). This pipeline samples 50 tokens for each cluster and presents each token together with the 15 preceding and following tokens to a language model. The language model (specifically GPT-4o) was then prompted to generate a label describing the algorithmic role of the token. The labels map algorithmic primitives in the reasoning trace to latent clusters. We further define a "cluster expressivity" metric (defined as the number of unique tokens expressed by a cluster, divided by the radius of that cluster in latent space, App. A.2), reasoning that clusters with higher expressivity will be more likely to be accurate primitives with clear algorithmic meaning. The LLM-driven pipeline and the cluster expressivity together support directly applying our method to novel models and tasks.

Step 3. Primitive Composition: Hierarchical Meta-Clustering and Temporal Clustering. To identify prevalent temporal compositions of these clusters, we then apply spectral clustering (Von Luxburg, 2007) to the matrix of transition probabilities between clusters. This gives rise to a smaller set of meta-cluster which highlight hierarchically structured reasoning steps. To infer the number of meta-clusters, we identify the largest spectral gap, using a minimum of four meta-clusters.

Step 4. Primitive Validation: Primitive Vector Extraction. Finally, we extract primitive vectors associated with particular clusters by adapting the methodology put forward in Todd et al. (2024): We extract attention head activations projected back into the residual stream for each response j, $Z^{(j)} \in \mathbb{R}^{T_j \times H \times L \times D}$ (where T_j is the number of tokens of response j, H is the number of attention heads per layer, L is the number of layers, and D is the residual

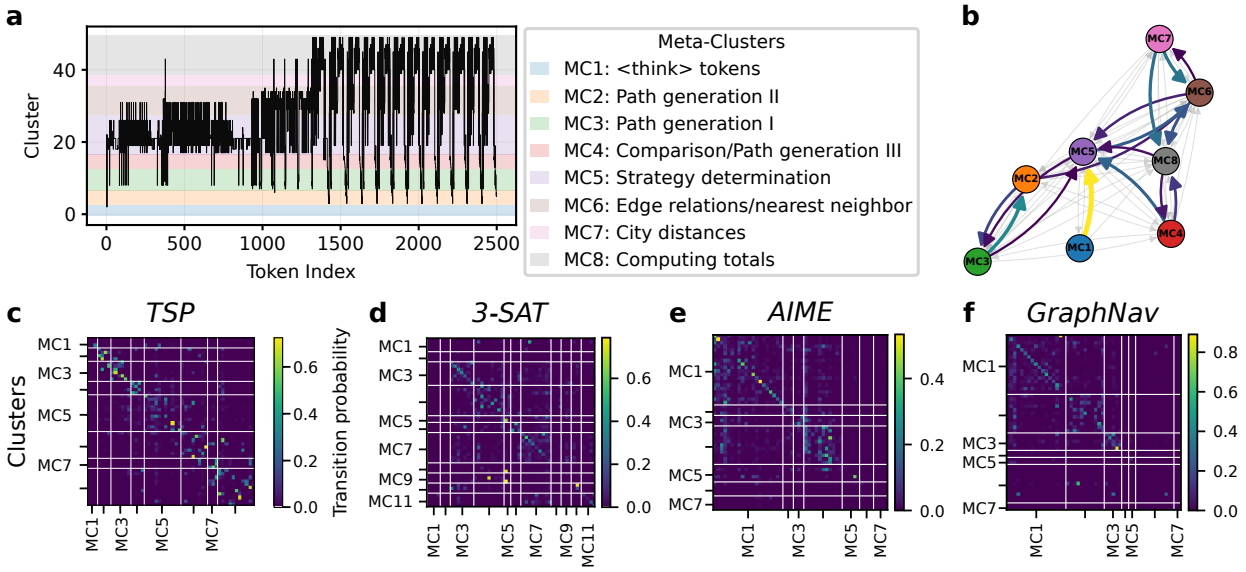

Figure 2. The reasoning trace expressed as a transition among hierarchically organized primitives that are traversed to generate the response. **a** The reasoning trace, defined as a walk through latent space primitives, reflects a transition graph across clusters that generates the output. Different meta-clusters are highlighted by colors. **b** The algorithmic transition graph of the reasoning trace reveals regular sequential patterns and recurring transition motifs. **c-f** Cluster-to-cluster transition matrices structured in terms of the inferred meta-clusters (indicated by the white lines) on tasks and benchmarks **c** TSP (Traveling Salesperson Problem), **d** 3-SAT (Boolean Satisfiability Problem), **e** AIME (Math Olympiad), and **f** GraphNav (Graph Navigation). Most transitions occur within meta-clusters.

stream's dimensionality). Denoting the set of tokens that are in a particular cluster c by $T_j[c] \subseteq \{1, \ldots, T_j\}$, we then compute the average attention head activations

$$\overline{Z}[c] := \frac{1}{\sum_{j=1}^{n} |T_j[c]|} \sum_{j=1}^{n} \sum_{t \in T_j[c]} Z_t^{(j)} \in \mathbb{R}^{L \times H \times D}. \quad (1)$$

We average $\overline{Z}[c]$ over the top $k = 35$ attention heads that reliably carry out in-context learning functions (roughly 2% of total attention heads, ranked by their average indirect effect across tasks, following Todd et al. (2024), Appendix B.6). We average over $n = 100$ responses, using the clusters extracted in step 1.

This yields a candidate primitive vector $v[c] \in \mathbb{R}^D$ for each cluster c. We validate candidate primitives by defining behavioral hallmarks associated with their proposed computational role and testing whether injecting the extracted primitive vectors results in an increase in those behavioral hallmarks. Beyond validation, we also examine their compositional generalization by testing their arithmetic composition and cross-task transfer, injecting primitive vectors extracted from AIME into the model while it performs TSP.

Algorithmic Fingerprinting. For a set of k clusters and a response j, we extract the relative frequency of tokens assigned to each cluster, $f^{(j)} \in \mathbb{R}^k$. $f^{(j)}$ provides an "algorithmic fingerprint", highlighting differences between the primitives involved in different responses. We measure the algorithmic dissimilarity between responses by the symmetric χ-squared distance between their frequencies,

$$\chi(f, g) := \sum_{i=1}^{k} \frac{(f_i - g_i)^2}{f_i + g_i}. \quad (2)$$

For sets of responses $(f^{(j)})_{j=1}^{n}$, $(g^{(j)})_{j=1}^{m}$, we measure differences in their algorithmic primitives by computing the average frequencies $f = \frac{1}{n}\sum_j f^{(j)}$, $g = \frac{1}{m}\sum_j g^{(j)}$ and then computing the signed χ-squared distance

$$\chi_s(f_c, g_c) := \mathrm{sgn}(f_c - g_c)\frac{(f_c - g_c)^2}{f_c + g_c}, \quad (3)$$

for each cluster c. A large positive difference indicates that the corresponding primitive is more prominent in the responses $(f^{(j)})_{j=1}^{m}$, a large negative difference indicates that it is more prominent in the responses $(g^{(j)})_{j=1}^{m}$. In particular, we compare Phi-4 responses to Phi-4-Reasoning responses and TSP responses to AIME responses.

5. Results

Primitive Tracing. We first traced primitives by clustering latent representations over entire reasoning traces, interpreting these clusters by identifying the corresponding tokens in the model output. Perhaps surprisingly, we found that despite the simplicity of our approach, many of these clusters were highly interpretable; for example, we discovered a cluster specifically associated with the implementation of a nearest-neighbor heuristic, and another cluster associated with validations, checks, and corrections. Notably, we found that the labels provided by the LLM-driven

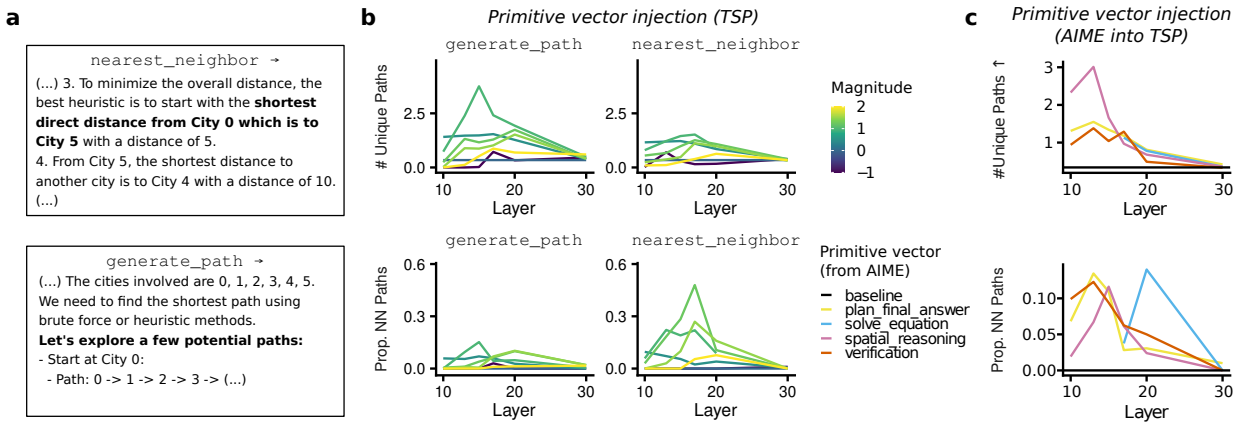

Figure 3. Primitive vector injection induces associated algorithmic behavior. **a** Injecting the `nearest_neighbor` or `generate_path` primitive vectors directly after the prompt increases expression in output. **b** Varying the injection layer and the magnitude modifies the number of unique paths generated (top row) and the proportion of nearest-neighbor paths (bottom row). **c** Primitive vectors from AIME were injected to different layers while solving Traveling Salesperson (TSP), showing cross-task primitive transfer and algorithmic induction.

Table 2. Effects of primitive vector injection on behavioral hallmarks (% above baseline). For each cell, we identify the maximal effect across all positive magnitudes and intervention layers (10, 13, 15, 17, 20, 30). Bold = strongest effect, underline = second strongest per column. *(%NN paths: Proportion of nearest neighbor paths generated. #Paths: Number of unique valid TSP paths generated. Dist. comp.: Earliest token at which a distance computation (sum of several numbers) occurs. Final ans.: Token at which* `<final_answer>` *token is generated. # Verif.: How many verification-related words are generated? # Comp.: How many comparison-related words are generated? See App. F for detailed definitions.)*

	%NN paths ↑	#Paths ↑	Dist. comp. ↓	Final ans. ↓	#Verif. ↑	#Comp. ↑
`nearest_neighbor`	**+56.1%**	+72.0%	-73.6%	-4.5%	+104.3%	+125.6%
`generate_path`	+4.8%	**+143.9%**	-76.0%	+18.4%	+25.0%	+12.2%
`compute_distance`	+22.2%	+36.5%	**-86.5%**	-47.4%	+198.9%	+79.3%
`final_answer`	+23.2%	+34.8%	-29.8%	**-81.5%**	+82.6%	-37.8%
`compare_verify`	+37.9%	+43.6%	-62.8%	-7.0%	+655.4%	**+1103.7%**
`compare_verify2`	+29.5%	+48.4%	-64.9%	-13.7%	**+706.5%**	+526.8%

annotation pipeline were highly consistent with our manual annotations. This validates the accuracy of the LLM-driven pipeline (Table 8). Furthermore, we identified similar primitives on more complex versions of Traveling Salesperson Problems (involving 9 or 13 cities), confirming that primitives remain coherent as complexity increases (App. A.8).

We then determined the average transition probability between the different clusters. Spectral clustering of this transition matrix revealed that, on TSP, clusters are sequentially composed in a hierarchical and stereotyped manner, as most transitions occurred within the same meta-cluster (Fig. 2c). Notably, different meta-clusters were also responsible for highly interpretable parts of the reasoning, e.g. path generation or computing distances (Fig. 2a).

Primitive Steering. Next, we validated the role of these primitives by extracting the corresponding primitive vectors and injecting them into Phi-4 during a response generation to TSP. First, we injected two vectors (`nearest_neighbor` and `generate_path`) into the model throughout the response generation, exploring different layers and magnitudes. We found that `nearest_neighbor` increased the proportion of nearest-neighbor paths generated by the model, whereas `generate_path` increased the number of total generated paths (Fig. 3a,b). This demonstrates that the primitive vectors selectively induce a particular behavior and validates the candidate primitives generated by our clustering.

To expand this investigation, we injected a broader range of candidate primitive vectors into the model after providing example paths and distance computations. For example, injecting the `compute_distance` primitive causes the model to more quickly compute the distance of a candidate path. Interestingly, the reverse is also true: subtracting the primitive makes the model *less* likely to implement a distance computation (Fig. 14). This suggests that subtracting primitive vectors can prevent associated algorithmic primitives. More broadly, we evaluate six behavioral hallmarks across six possible primitive vectors and find that all behavioral hallmarks are most strongly induced by a candidate primitive vector with a relevant computational role (Table 2; see further control experiments in App. F.1,F.3,F.4).

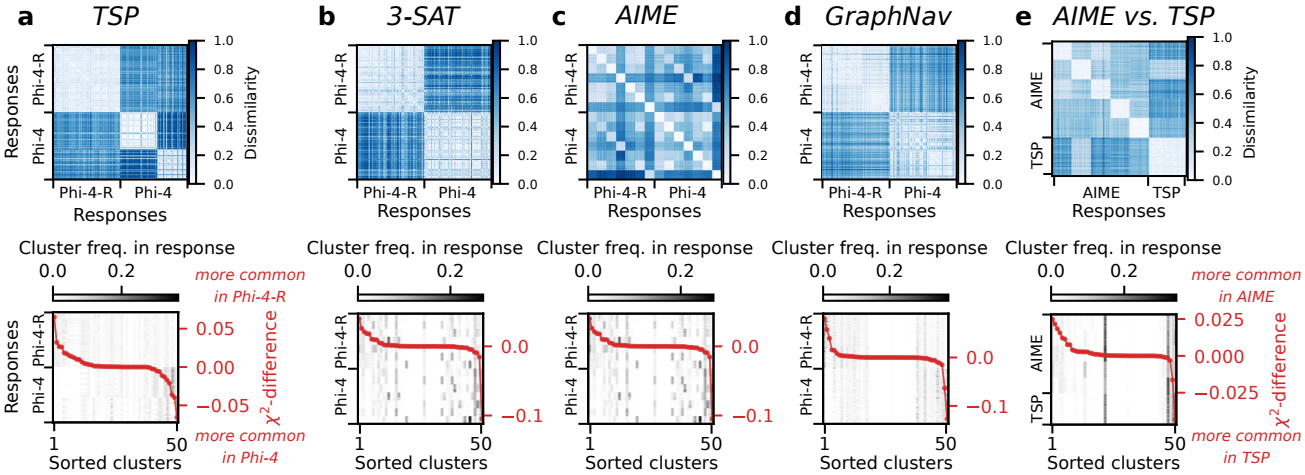

Figure 4. Primitive Cluster Patterns in Phi-4 and Phi-4-Reasoning. *top* Normalized dissimilarity of primitive cluster frequencies between **a-d** Phi-4 and Phi-4-Reasoning for a) TSP, b) 3-SAT, c) AIME, and d) GraphNav; and **e** between AIME and TSP on Phi-4-Reasoning. *bottom* **a-d** Clusters sorted by whether they appear more frequently in responses by Phi-4-Reasoning (positive difference) or Phi-4 (negative difference). The lineplot specifies the differences whereas the rasterplot specifies the relative frequencies of the different clusters per response. **e** Clusters sorted by whether they appear more frequently in AIME (positive difference) or TSP (negative difference).

Notably, the role of a particular cluster cannot always be inferred from the tokens on which it is active alone; the surrounding context often also plays a role. For example, the cluster `compare_and_verify` is largely active on the token corresponding to the final distance of a candidate path. However, we noticed that tokens on which this cluster was active were often followed by subsequent checks and comparisons. We therefore hypothesized that this cluster may not just represent the final distance, but also induce a verification primitive. Table 2 confirms this hypothesis. Interestingly, we can observe this dual nature in representational space as well. In a t-SNE plot of the average representational trajectory of these example primitives, we observe that in earlier layers, `compare_or_verify` evolves along other path-related primitives like `generate_new_path`, whereas it moves closer to `compare_or_verify2` in later layers (Fig. 6).

Comparing Algorithmic Primitives between Models. To compare primitives between Phi-4 and Phi-4-Reasoning, we computed the dissimilarity between the primitive frequencies involved in their responses. We found that responses by Phi-4-Reasoning are highly stereotyped (Fig. 4a). In contrast, Phi-4 has two subgroups of distinct responses. Notably, these groups map onto two distinct approaches towards solving TSP: a brute-force search and a random guess without further refinement. This highlights that algorithmic fingerprinting reveals both differences between models, and differences between responses from the same model. Beyond TSP, the responses from Phi-4 and Phi-4-Reasoning on 3-SAT and GraphNav are highly stereotyped (Fig. 4b,d). On AIME, algorithmic primitives involved in responses to the same question were very similar to each other, whereas primitives involved in responses to different questions were substantially more different, reflecting the higher diversity of questions in AIME (Fig. 4c).

Next, we analyzed which primitives are more common in Phi-4 or Phi-4-Reasoning by computing the signed χ-squared difference, and analyzing the clusters with the largest positive and negative values. This allows us to identify important differences in response strategies across the two models. For example, on TSP and 3-SAT, distinct heuristic approaches arise more commonly in Phi-4-Reasoning than Phi-4 (`nearest_neighbor` and `if_clause_true`, App. A). Beyond task-specific strategies, our comparative analysis also highlights broader differences between the two models: in particular, clusters that are more common in Phi-4-Reasoning relate to general-purpose reasoning steps such as verification, recalling instructions, or planning the final answer.

To assess whether our framework provides relevant insights beyond the Phi model class, we applied it to TSP responses generated by Qwen3-8B. As with Phi models, Qwen's most expressive clusters implemented algorithmic steps that are crucial for solving TSP, such as distance retrieval and summation (App. A.9).

Compositional Primitive Induction. So far, we have considered sequential compositions of algorithmic primitives. To investigate arithmetic compositions of primitives, we consider a set of algorithmic in-context learning tasks that require identifying 1) the last node of a path ("Terminal node recognition," TNR), 2) the node with the higher reward ("Reward comparison," RC; reward was defined by a number between 1 and 100); and 3) the most highly re-

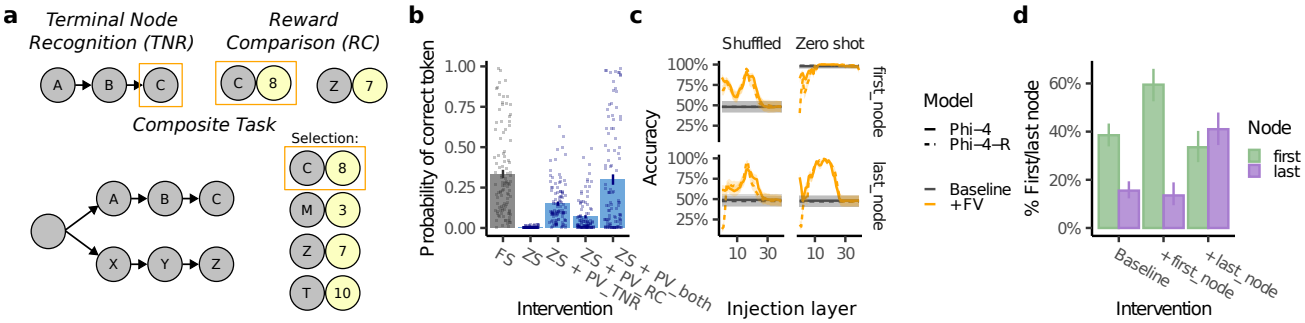

Figure 5. Compositional induction and cross-task transfer of algorithmic primitives. **a Task setting.** We consider three tasks: 1) terminal node recognition (TNR) requires models to return the end node in a graph, 2) reward comparison (RC) requires models to return the node with the highest associated reward, and 3) a composite task where the prompt presents a few states and their reward, and the model needs to return the state with the highest reward that is also a terminal node of the presented graph (ignoring states that are not in the graph). **b Cross-setting injection across layers.** Llama-3-8B can solve the composite task (panel a) in a few-shot setting but not zero-shot. Single primitive injection improves its zero-shot performance, and injecting the sum of a TNR primitive vector (PV_TNR) and a RC primitive vector (PV_RC) in a zero-shot setting (ZS) nearly matches its few-shot performance (FS). See Figs. 12,13 for additional models. **c Few-shot to zero-shot injection.** In Phi-4, we extracted primitive vectors (PVs) for returning the first or last node of a graph in the few-shot setting. Injecting these PVs into different layers of the model reliably improves performance on the zero-shot and shuffled few-shot settings. **d Cross-task injection.** We injected primitive vectors that were extracted in panel c into layer 17 of Phi-4 when solving TSP. This injection increased mention of the corresponding node in Phi-4 reasoning traces.

warded node between two paths, a composition of TNR and RC (Fig. 5a; see example prompt in App. E.1). Remarkably, we find that adding together the primitive vectors for TNR and RC induces this composite behavior in Llama-3-8B, causing it to match few-shot performance in a zero-shot setting (Fig. 5b). We then evaluated compositional induction across a broader range of models (Phi-4, Phi-4-Reasoning, Qwen3.5-4B/8B/14B). Compositional induction was successful for some models but not others (Figs. 12,13). This finding suggests different models may implement composition differently.

Cross-task and Compositional Generalization. Generalized algorithmic primitives which can be applied across different tasks, can help us understand why reasoning-finetuned models like Phi-4-Reasoning show improvements on tasks they were not finetuned on. To better understand the degree to which algorithmic primitives in Phi-4-Reasoning are shared between tasks, we apply our algorithmic tracing framework to a set of responses from both TSP and AIME (Fig. 4e). This analysis highlights that while there are different algorithmic primitives at play in either task, the tasks also have many primitives in common.

Next, we tested whether cross-task primitive injection would induce behavioral changes. We extracted primitive vectors from an algorithmic in-context learning task that required extracting a path's first or last node (Fig. 5c) and injected them into complex reasoning traces solving TSP immediately after Phi-4 had mentioned a particular path. Despite the extremely different domain, injecting these vectors indeed had the expected effect and selectively induced the behavior associated with each primitive. More specifically, the Phi-4 reasoning trace mentioned the first node of the path significantly more often when `get_first_node` was injected (one-sided Wilcoxon rank sum test; $p < 10^{-6}$) but not when `get_last_node` was injected ($p = 0.74$). Similarly, Phi-4 mentioned the last node of the path significantly more often when `get_last_node` was injected ($p < 10^{-6}$) but not when `get_first_node` was injected ($p = 0.88$) (Fig. 5d). Therefore, primitive vectors can steer model behavior even when injected in the middle of a complex reasoning trace.

Finally, we considered relevant primitive vectors from AIME (e.g. `spatial_reasoning`, `plan_final_answer`) and injected them into Phi-4 generating a response to a TSP prompt. We found that these primitive vectors caused the model to generate more paths and a higher proportion of nearest-neighbor paths — a hallmark of Phi-4-Reasoning responses (Fig. 3c). These effects were statistically significant (Table 14).

6. Discussion

AlgoTrace: A Framework for Algorithmic Tracing and Steering. We introduce a framework for tracing and steering algorithmic primitives as building blocks of LLM reasoning with geometric compositionality. We adapted function vector techniques (Todd et al., 2024) to extract minimal, 1-step primitive vectors from carefully designed simple tasks as well as benchmarks with more complex reasoning contexts. When "injected" to LLM layers after the prompt or in the middle of reasoning traces, these primitive vectors increase the behavioral expression of corresponding reasoning steps (Figs. 3 and 5). This extends function vector research beyond simple input-output relations. A

central contribution of our work is the compositional exploration of primitive vector arithmetic. Our framework bridges a principled path from laboratory-identified primitives to real-world reasoning behaviors.

Toward a Geometry of Compositional Abstraction in LLMs. We find that algorithmic primitives exhibit geometric regularities through compositional operations like addition, scalar multiplication, and concatenation. The layer and magnitude of injection shape the expression of the primitive (Fig. 3). We show cross-task transfer and generalizability of primitive vectors: spatial reasoning and verification primitives extracted from AIME successfully transfer to TSP. This transferability hints at potentially universal algorithmic building blocks underlying diverse reasoning capabilities. If this is the case, an algorithm may be expressed by an LLM in terms of an algebraic geometry of primitive vectors and their composition. We here provide a framework for evaluating the composition of primitives across tasks, across the reasoning trace, and their within-layer algebraic composition.

Reasoning vs. Memorization. Our work contributes to recent discussions about the extent to which LLMs exhibit abstract reasoning rather than relying on memorized procedures, failing to generalize in counterfactual scenarios (Wu et al., 2024; Power et al., 2022; Zhang et al., 2024; Wang et al., 2025). While models were not directly trained on our tasks (Abdin et al., 2025), it is possible that they may be using amortized reasoning based on similar examples in the training. LLMs exhibit better problem-solving in high probability than low probability settings (e.g. McCoy et al., 2024a), which rely on statistical regularities rather than abstract reasoning. Moreover, the absence of metacognitive abilities in LLMs results in brittle problem-solving (Johnson et al., 2025; Lewis & Mitchell, 2025). However, models finetuned for reasoning exhibit less sensitivity to task probability than base models (McCoy et al., 2024b) and improved metacognitive-like uncertainty management strategies (e.g., verifying whether a candidate solution is correct) (Guo et al., 2025; Gandhi et al., 2025) that in turn causally improve reasoning (Bogdan et al., 2025). In line with these findings, our analysis identified algorithmic primitives for managing uncertainty (e.g., verification) and in-context recall. Such primitives and their associated behaviors may underlie a shift from memorization toward compositional problem-solving in reasoning models.

Granularity of Algorithmic Primitives. In considering algorithmic primitives, we focus on fine-grained computational steps, in contrast to more coarse-grained interventions such as persona vectors (Chen et al., 2025; Hu et al., 2026). How fine-grained should these primitives be? We suggest a causal definition: algorithmic primitives should be at a level that can be causally induced. For example, while we can induce a nearest-neighbor primitive, it is unlikely that we can induce a computation that only selects the nearest neighbor of the second city. Methodological choices (e.g. the number of latent space clusters) will impact how fine-grained the putative primitives are. It is therefore possible that some of the algorithmic primitives we identify could further decompose into sub-primitives. Likewise, our meta-clustering approach reveals that the identified primitives themselves assemble into meaningful hierarchies and sequential motifs.

Limitations. One limitation is that the primitives, corresponding clusters, and meta-clusters we identify do not always map to a known algorithm. Future work can identify **primitive ontology** and algorithmic logic of different models, identifying commonalities and differences across them. Here, we focus on reasoning as multi-step problem-solving in limited tasks, which may not capture the complexity and multi-modal nature of natural reasoning, studied in cognitive sciences for decades (Tversky, 2005; Shepard & Metzler, 1971; MacGregor & Ormerod, 1996). Notably, we have focused on linear compositions of primitive vectors, leaving more complex interactions, e.g. non-linear combinations of several vectors on manifolds, for future work. Finally, future work can build on our framework to expand on particular choices of methods for identifying candidate primitives, extracting primitive vectors, injecting these vectors, and identifying behavioral hallmarks for evaluating them. For instance, while our k-means and spectral clustering of the latent space yielded promising structures, future work could relax the Euclidean and isotropy assumptions by exploring hyperbolic or other Riemannian clustering geometries (Nickel & Kiela, 2017; Gu et al., 2019).

Future Directions. The algorithmic tracing and steering framework can be applied to any architecture (e.g., vision, diffusion, and multi-modal models) and domain beyond reasoning. An immediate extension is a more detailed manifold analysis in order to capture the compositional geometry of primitives beyond linear composition, and establish a task-specific and universal **primitive ontology**. Moreover, our framework (and in particular compositional primitive induction) should be evaluated across a broader range of models. Another key direction is the **algorithmic training and finetuning of LLMs**, with algorithmic objectives. For instance, instead of next-token-prediction, future architectures can use our framework to define compositional algorithmic objectives, i.e., a next-primitive prediction (NPP).

Conclusion. The identification of primitives and their compositional geometry opens new avenues for interpretability research and suggests principled approaches for computational models of human reasoning, model-human alignment, and enhancing LLM reasoning capabilities.

Acknowledgements

SL and TM primarily conducted their work during an internship at Microsoft Research. We thank Taylor Webb for helpful discussions. We also thank the Institute for Pure and Applied Mathematics (IPAM) at UCLA and its Research in Industrial Projects for Students (RIPS) program. Part of this research originated when IM, OE, and TM began collaborating at the Institute for Pure and Applied Mathematics (IPAM) Long Program on the Mathematics of Intelligences, supported by the National Science Foundation (Grant No. DMS-1925919). The authors acknowledge the participants of this Program. KL, ZP, PZ, and SZ were supported through IPAM's Research in Industrial Projects for Students (RIPS) program. We especially thank Jiayi Li for supporting the students during their internship. OE is supported through institutional funding from the German Federal Ministry of Research, Technology and Space (BMFTR) via BIFOLD (grant ref. 01IS18037A).

Impact Statement

Our work offers a rigorous and multi-level framework for the evaluation and understanding of reasoning behavior in large language models (LLMs). Current LLMs are increasingly capable, but can fail in unpredictable ways at reasoning and can display harmful biases (Gallegos et al., 2024). Our approach offers an algorithmic understanding of LLM reasoning at both the behavioral level of reasoning traces and the internal level of latent processes and layer by layer representations. Such an algorithmic understanding can help us understand the contexts in which LLMs' algorithmic primitives can implement correct reasoning behavior and the contexts in which we would expect them to make mistakes. This improved understanding, in turn, would allow us to avoid deploying these models in contexts where we would expect them to make mistakes. In addition, our approach could flag the specific algorithmic primitives that give rise to particular biases, and offer a path to target finetuning to those specific algorithmic failures. Finally, our approach highlights the fundamental steps required to solve a reasoning problem, offering empirical ways to measure and compare the sequence of primitives generated by the LLM. In the future, this research can inspire improved and more sample-efficient training procedures as well as novel architectures. The sample efficiency of architectures trained with an algorithmic objective of compositional generalization can advance artificial intelligence in two ways, both improving the quality of performance and reducing the ecological impact of training. Such advances have many potential societal consequences, consistent with foundational research on machine learning models more generally.

References

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Abdin, M., Agarwal, S., Awadallah, A., Balachandran, V., Behl, H., Chen, L., de Rosa, G., Gunasekar, S., Javaheripi, M., Joshi, N., Kauffmann, P., Lara, Y., Mendes, C. C. T., Mitra, A., Nushi, B., Papailiopoulos, D., Saarikivi, O., Shah, S., Shrivastava, V., Vineet, V., Wu, Y., Yousefi, S., and Zheng, G. Phi-4-reasoning technical report, 2025. URL https://arxiv.org/abs/2504.21318.

Ali, A., Schnake, T., Eberle, O., Montavon, G., Müller, K.-R., and Wolf, L. XAI for transformers: Better explanations through conservative propagation. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 435–451. PMLR, 2022. URL https://proceedings.mlr.press/v162/ali22a.html.

Balachandran, V., Chen, J., Chen, L., Garg, S., Joshi, N., Lara, Y., Langford, J., Nushi, B., Vineet, V., Wu, Y., et al. Inference-time scaling for complex tasks: Where we stand and what lies ahead. *arXiv preprint arXiv:2504.00294*, 2025.

Bernardi, S., Benna, M. K., Rigotti, M., Munuera, J., Fusi, S., and Salzman, C. D. The geometry of abstraction in the hippocampus and prefrontal cortex. *Cell*, 183(4): 954–967, 2020.

Bogdan, P. C., Macar, U., Nanda, N., and Conmy, A. Thought anchors: Which LLM reasoning steps matter? In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=VnSlfeRCaU.

Bürger, L., Hamprecht, F. A., and Nadler, B. Truth is universal: Robust detection of lies in llms. *Advances in Neural Information Processing Systems*, 37:138393–138431, 2024.

Camburu, O.-M., Rocktäschel, T., Lukasiewicz, T., and Blunsom, P. e-snli: Natural language inference with natural language explanations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/4c7a167bb329bd92580a99ce422d6fa6-Paper.pdf.

Chen, R., Arditi, A., Sleight, H., Evans, O., and Lindsey, J. Persona vectors: Monitoring and controlling character traits in language models, 2025. URL https://arxiv.org/abs/2507.21509.

Chormai, P., Herrmann, J., Müller, K.-R., and Montavon, G. Disentangled explanations of neural network predictions by finding relevant subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. What does BERT look at? an analysis of BERT's attention. In Linzen, T., Chrupała, G., Belinkov, Y., and Hupkes, D. (eds.), *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL https://aclanthology.org/W19-4828/.

Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2126–2136, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1198. URL https://aclanthology.org/P18-1198/.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Eberle, O., Büttner, J., Kräutli, F., Müller, K.-R., Valleriani, M., and Montavon, G. Building and interpreting deep similarity models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1149–1161, 2022. doi: 10.1109/TPAMI.2020.3020738.

Eberle, O., McGee, T. A., Giaffar, H., Webb, T. W., and Momennejad, I. Position: We need an algorithmic understanding of generative AI. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025. URL https://openreview.net/forum?id=eax2ixyeQL.

Fascianelli, V., Battista, A., Stefanini, F., Tsujimoto, S., Genovesio, A., and Fusi, S. Neural representational geometries reflect behavioral differences in monkeys and recurrent neural networks. *Nature Communications*, 15(6479), 2024. doi: 10.1038/s41467-024-50503-w. URL https://doi.org/10.1038/s41467-024-50503-w.

Fatemi, B., Halcrow, J., and Perozzi, B. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IuXR1CCrSi.

Gallegos, I. O., Rossi, R. A., Barrow, J., Tanjim, M. M., Kim, S., Dernoncourt, F., Yu, T., Zhang, R., and Ahmed, N. K. Bias and fairness in large language models: A survey. *Computational Linguistics*, 50 (3):1097–1179, September 2024. doi: 10.1162/coli_a_00524. URL https://aclanthology.org/2024.cl-3.8/.

Gandhi, K., Chakravarthy, A. K., Singh, A., Lile, N., and Goodman, N. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STars. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=QGJ9ttXLTy.

Geiger, A., Wu, Z., Lu, H., Rozner, J., Kreiss, E., Icard, T., Goodman, N., and Potts, C. Inducing causal structure for interpretable neural networks. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7324–7338. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/geiger22a.html.

Geiger, A., Wu, Z., Potts, C., Icard, T., and Goodman, N. Finding alignments between interpretable causal variables and distributed neural representations. In *Causal Learning and Reasoning*, pp. 160–187. PMLR, 2024.

Gu, A., Sala, F., Gunel, B., and Ré, C. Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJxeWnCcF7.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Gurnee, W. and Tegmark, M. Language models represent space and time. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jE8xbmvFin.

Han, J., Kamber, M., Pei, J., et al. Data mining: concepts and techniques. Technical report, Morgan Kaufmann Publishers, 2012.

Hawasly, M., Dalvi, F., and Durrani, N. Scaling up discovery of latent concepts in deep nlp models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 793–806, 2024.

Hendel, R., Geva, M., and Globerson, A. In-context learning creates task vectors. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9318–9333, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.624. URL https://aclanthology.org/2023.findings-emnlp.624/.

Hernandez, E., Schwettmann, S., Bau, D., Bagashvili, T., Torralba, A., and Andreas, J. Natural language descriptions of deep features. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=NudBMY-tzDr.

Hewitt, J. and Manning, C. D. A structural probe for finding syntax in word representations. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://aclanthology.org/N19-1419/.

Hu, Z., Rostami, M., and Thomason, J. Expert personas improve llm alignment but damage accuracy: Bootstrapping intent-based persona routing with prism. *arXiv preprint arXiv:2603.18507*, 2026.

Huang, S., Mamidanna, S., Jangam, S., Zhou, Y., and Gilpin, L. H. Can large language models explain themselves? a study of llm-generated self-explanations, 2023. URL https://arxiv.org/abs/2310.11207.

Huh, M., Cheung, B., Wang, T., and Isola, P. Position: The platonic representation hypothesis. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 20617–20642. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/huh24a.html.

Jafari, F. R., Montavon, G., Müller, K.-R., and Eberle, O. Mambalrp: Explaining selective state space sequence models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Johnson, S. G., Karimi, A.-H., Bengio, Y., Chater, N., Gerstenberg, T., Larson, K., Levine, S., Mitchell, M., Rahwan, I., Schölkopf, B., et al. Imagining and building wise machines: The centrality of ai metacognition. *Trends in Cognitive Sciences*, 2025.

Kopf, L., Feldhus, N., Bykov, K., Bommer, P. L., Hedström, A., Höhne, M. M., and Eberle, O. Capturing polysemanticity with PRISM: A multi-concept feature description framework. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=btJUnAPQ7j.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMLR, 2019.

Lewis, M. and Mitchell, M. Evaluating the robustness of analogical reasoning in large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=t5cy5v9wph.

Lloyd, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Lobo, E., Agarwal, C., and Lakkaraju, H. On the impact of fine-tuning on chain-of-thought reasoning. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11679–11698, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.584. URL https://aclanthology.org/2025.naacl-long.584/.

MacGregor, J. N. and Ormerod, T. Human performance on the traveling salesman problem. *Perception & psychophysics*, 58(4):527–539, 1996.

Madsen, A., Chandar, S., and Reddy, S. Are self-explanations from large language models faithful? In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 295–337, 2024.

Marks, S. and Tegmark, M. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=aajyHYjjsk.

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., and Griffiths, T. L. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121, 2024a.

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., and Griffiths, T. L. When a language model is optimized for reasoning, does it still show embers of autoregression? an analysis of openai o1. *arXiv preprint arXiv:2410.01792*, 2024b.

McDougall, C. S., Conmy, A., Rushing, C., McGrath, T., and Nanda, N. Copy suppression: Comprehensively understanding a motif in language model attention heads. In Belinkov, Y., Kim, N., Jumelet, J., Mohebbi, H., Mueller, A., and Chen, H. (eds.), *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 337–363, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.blackboxnlp-1.22. URL https://aclanthology.org/2024.blackboxnlp-1.22/.

Merullo, J., Eickhoff, C., and Pavlick, E. Circuit component reuse across tasks in transformer language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fpoAYV6Wsk.

Momennejad, I., Hasanbeig, H., Vieira Frujeri, F., Sharma, H., Jojic, N., Palangi, H., Ness, R., and Larson, J. Evaluating cognitive maps and planning in large language models with cogeval. *Advances in Neural Information Processing Systems*, 36:69736–69751, 2023.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30, 2017.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Petukhova, A., Matos-Carvalho, J. P., and Fachada, N. Text clustering with large language model embeddings. *International Journal of Cognitive Computing in Engineering*, 6:100–108, 2025.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., and Montavon, G. Higher-order explanations of graph neural networks via relevant walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7581–7596, 2022. doi: 10.1109/TPAMI.2021.3115452.

Sharkey, L., Chughtai, B., Batson, J., Lindsey, J., Wu, J., Bushnaq, L., Goldowsky-Dill, N., Heimersheim, S., Ortega, A., Bloom, J. I., Biderman, S., Garriga-Alonso, A., Conmy, A., Nanda, N., Rumbelow, J. M., Wattenberg, M., Schoots, N., Miller, J., Saunders, W., Michaud, E. J., Casper, S., Tegmark, M., Bau, D., Todd, E., Geiger, A., Geva, M., Hoogland, J., Murfet, D., and McGrath, T. Open problems in mechanistic interpretability. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=91H76m9Z94. Survey Certification.

Shepard, R. N. and Metzler, J. Mental rotation of three-dimensional objects. *Science*, 171(3972):701–703, 1971.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Todd, E., Li, M., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AwyxtyMwaG.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=bzs4uPLXvi.

Tversky, B. Visuospatial reasoning. In *The Cambridge Handbook of Thinking and Reasoning*, pp. 209–240. 2005.

Vig, J. and Belinkov, Y. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL workshop BlackboxNLP: analyzing and interpreting neural networks for NLP*, pp. 63–76, 2019.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580/.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Wang, R., Zelikman, E., Poesia, G., Pu, Y., Haber, N., and Goodman, N. Hypothesis search: Inductive reasoning with language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=G7UtIGQmjm.

Wang, X., Antoniades, A., Elazar, Y., Amayuelas, A., Albalak, A., Zhang, K., and Wang, W. Y. Generalization v.s. memorization: Tracing language models' capabilities back to pretraining data. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=IQxBDLmVpT.

Wu, Z., Qiu, L., Ross, A., Akyürek, E., Chen, B., Wang, B., Kim, N., Andreas, J., and Kim, Y. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. Association for Computational Linguistics, 2024.

Yang, L., Lin, Z., Lee, K., Papailiopoulos, D., and Nowak, R. D. Task vectors in in-context learning: Emergence, formation, and benefits. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=lODGn1Rp5t.

Ye, T., Xu, Z., Li, Y., and Allen-Zhu, Z. Physics of Language Models: Part 2.1, Grade-School Math and the Hidden Reasoning Process. In *Proceedings of the 13th International Conference on Learning Representations*, ICLR '25, April 2025. Full version available at http://arxiv.org/abs/2407.20311.

Zhang, Y., Wang, H., Feng, S., Tan, Z., Han, X., He, T., and Tsvetkov, Y. Can LLM graph reasoning generalize beyond pattern memorization? In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2289–2305, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.127. URL https://aclanthology.org/2024.findings-emnlp.127/.

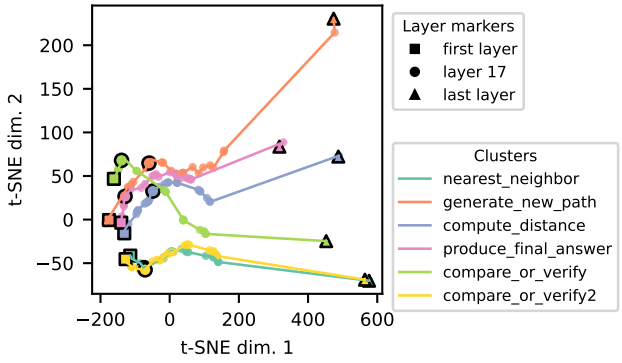

Figure 6. Average t-SNE trajectories across different highlighted primitives.

A. Identification of algorithmic primitives

A.1. Detailed methods

We implemented and analyzed five clustering models, using the following datasets:

1. five responses on TSP from Phi-4-Reasoning
2. five TSP responses from Phi-4 and five TSP responses from Phi-4-reasoning
3. five AIME responses from Phi-4 and Phi-4-Reasoning
4. five 3-SAT responses from Phi-4 and Phi-4-Reasoning
5. five GraphNav responses from Phi-4 and Phi-4-Reasoning

In all cases, we extracted the token-by-token representation from layer 17 of Phi-4-base. We used k-means clustering with $k = 50$ clusters and designed an HTML interface to visually inspect the different clusters.

Our analysis involved three key "choice points": First, we selected a specific layer and number of clusters. In follow-up analyses, we found that our insights were robust to different choices of both layers (App. G.1) and number of clusters (App. G.2). Second, we prioritized methodological simplicity by using a well-established approach for constructing function vectors (Todd et al., 2024), leaving a systematic exploration and evaluation of more complex approaches for future work. Third, we selected specific behavioral hallmarks to evaluate different primitive vectors. We emphasize that existing work on interpreting LLMs largely relies on pre-specified motifs (e.g. verification), thus presenting a similarly simple "choice point" (e.g. Gandhi et al., 2025). By comparison, our choices of behavioral hallmarks are grounded in the labeling of associated primitives. Future work should explore generating these behavioral hallmarks in an automated manner, e.g. by extending our proposed LLM-based pipeline.

Below we highlight selected clusters from the different clustering models.

A.2. Cluster expressivity

To ease the automated evaluation of identified clusters, we introduce a novel metric to quantify the expressivity of each cluster. "Cluster expressivity" measures the number of unique tokens expressed by a cluster, divided by the radius of that cluster in latent space. More formally, consider a response with tokens $T \in \mathcal{T}^n$ where $\mathcal{T}$ is the set of tokens. Let $x \in \mathbb{R}^{n \times d}$ be its latent representation and $C \in \mathcal{C}^n$ be its associated cluster identities. For each $c \in \mathcal{C}$, let $\text{cent}[c] \in \mathbb{R}^d$ be the associated centroid. We then define the radius

$$\text{radius}[c] := \max_{n:C_n=c} \|x_n - \text{cent}[c]\|_2 \tag{4}$$

as the maximal ℓ_2-distance of a data point within this cluster. We define the number of unique tokens expressed by the cluster as

$$m[c] := |\{t \in \mathcal{T} : \exists_n C_n = c, T_n = t\}|. \tag{5}$$

We then compute "cluster expressivity" as

$$\frac{m[c]}{\text{radius}[c]}. \tag{6}$$

To summarize this metric, we compute its median value across all responses. We reason that high expressivity indicates better-defined operations with rich vocabulary, i.e. multiple valid ways to express the same operation. We therefore take them as more likely to be accurate primitives with clear algorithmic meaning. Table 8 shows metrics of expressivity for all clusters extracted from the responses by Phi-4-Reasoning on TSP. Notably, our selected primitives consistantly have high cluster expressivity.

A.3. Phi-4-Reasoning responses on TSP

By manually inspecting the responses, we find that many clusters correspond to specific reasoning motifs and candidate algorithmic primitives.

Cluster 37: `nearest_neighbor`. This cluster is active specifically during the tokens where the model identifies the nearest-neighbor path or the closest city within a particular candidate path. As Phi-4-Reasoning's responses, in contrast to those by Phi-4, often started out by an initial guess using the nearest-neighbor heuristic (see Appendix D), we hypothesized that this would be a particularly relevant primitive.

Cluster 26: `generate_new_path`. This cluster usually preceded a new candidate path. We therefore hypothesized that representations in these tokens may encode relevant primitives for generating new paths.

Cluster 15: `compute_distance`. This cluster seems to precede the computation of the total distance of a candidate path.

Cluster 41: `compare_or_verify_2`. This cluster corresponds to a range of statements involved in comparisons (in particular comparing the distance of different paths), verification (e.g. verifying whether a generated path is valid), or recall (e.g. recalling the best path so far, a closely related step to comparisons).

Cluster 35: `compare_or_verify`. Interestingly, this cluster is usually active on the final presentation of the total distance. However, we noticed that not every token corresponding to a total distance belonged to this cluster. Rather, this cluster reliably predicted whether the next sentence involved a comparison or verification step — in particular, cluster 35 often preceded cluster 41. We therefore hypothesized that this cluster could be involved in promoting these comparisons and verifications.

Cluster 30: `produce_final_answer`. This cluster was active during the production of the final answer and we hypothesized that it would be relevant for that step.

A.4. Responses by Phi-4 and Phi-4-Reasoning on TSP

By selectively analyzing the clusters associated with the largest-magnitude differences between their involvement in Phi-4 and Phi-4-Reasoning, we identify a set of clusters that are more common in Phi-4 or Phi-4-Reasoning:

A.4.1. Clusters more common in Phi-4-Reasoning

Cluster 19: `compare_or_verify`. This cluster again implements comparisons and verifications. It arises almost exclusively in Phi-4-Reasoning responses, highlighting the reasoning-finetuned model's tendency to implement more comparisons and verifications.

Clusters 13 and 10: `path_generation` Cluster 13 represents early cities during the path generation while cluster 10 represents the connections between cities. Notably, it only represents paths that are not generated as a part of the brute-force strategy, indicating that the computations involved in the brute-force strategy are different. For responses by Phi-4 that do not implement a brute-force strategy, these clusters are also more frequently active.

Cluster 22: `generate_path_guided`. This cluster precedes a path generation, but only in Phi-4-Reasoning. We therefore hypothesize that clusters 6 and 22 might induce different structures in their generated paths.

A.4.2. Clusters more common in Phi-4.

Cluster 11: `brute_force`. While this cluster sometimes arises in Phi-4-Reasoning, it is more strongly associated with Phi-4 and arises, in particular, when Phi-4 states its approach to implement a brute-force search.

Cluster 41: `generate_path_brute_force` This cluster is specifically active during the generation of paths involved in brute-force searches.

Cluster 0: `compound_distance_lookup` This cluster is active specifically before a distance lookup requires a multi-step computation. For example, consider the following segment: `**Permutation: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0** - Distance = 44 (0 to 1) + 36 (1 to 2) + 32 (2 to 3) + 46 (3 to 4) + 26 (4 to 5) + 37 (5 to 0) = 221`. Here, predicting the distance requires first identifying the relevant edge in the path before looking up the corresponding entry in the weight matrix. In contrast, consider the following segment: `Alternatively: 0->5->3->2->1->4->0. Then: 0->5=37, 5->3=31, 3->2=32, 2->1=36, 1->4=28, 4->0=42. Total: 37+31=68, +32=100, +36=136, +28=164, +42=206.` Here, adding the new distances only requires moving forward in the previously generated sequence and therefore does not require a multi-step computation. Importantly, cluster 0 is not active in this sentence.

A.4.3. Clusters common to both models.

Cluster 27: `edge_retrieval`. This cluster indicates the retrieval of the distance between two edges. It is a shared primitive between Phi-4 and Phi-4-Reasoning.

Cluster 6: `generate_path_1`. This cluster precedes a path generation and arises in both Phi-4 and Phi-4-Reasoning.

A.5. Responses by Phi-4 and Phi-4-Reasoning on 3-SAT

Clusters more common in Phi-4-Reasoning. **Cluster 30:** `reasoning_scaffold`. This cluster appears to involve a lot of strategizing and reasoning.

Cluster 37: `if_clause_true`. This cluster corresponds to a particular strategy implemented by Phi-4-Reasoning in solving 3-SAT problems: investigating the consequences of one particular clause being true or false. Notably, this is a cluster rarely occurring in Phi-4, which may therefore illustrate an algorithmic primitive largely occurring in Phi-4-Reasoning. Indeed, we discovered these differences in strategy from our clustering analysis, illustrating how our approach could potentially support better understand differences in algorithmic strategies.

A.5.1. Clusters more common in Phi-4.

Cluster 34: `logical_expressions_latex`. This cluster is largely active on Latex based logical expressions. This is consistent with more general observations that Latex formats are much more common in Phi-4 than Phi-4-Reasoning.

A.5.2. Clusters common to both models.

Cluster 11: `if_variable_true`. In contrast, this cluster mostly arises in Phi-4. Interestingly, besides being involved in general reasoning, it appears to promote an alternative strategy to the strategy above: setting particular variables to true or false. This results in a more brute-force oriented approach, mirroring the differences between Phi-4 and Phi-4-Reasoning on TSP.

Cluster 38: `recall_clause`. This cluster precedes the recall of particular clauses from the problem.

A.6. Responses by Phi-4 and Phi-4-Reasoning on AIME

A.6.1. Clusters more common in Phi-4-Reasoning

Cluster 3: `solve_equation`. This cluster is mostly active on two tasks which require solving an equation/inequation. While it is also partially active in Phi-4 for one of those tasks, it is much more common in Phi-4-Reasoning.

Cluster 31: `verification_alternate_approach`. This cluster corresponds to verifying specific statements or registering potential concerns with an approach and considering an alternative approach. It arises almost exclusively in Phi-4-Reasoning.

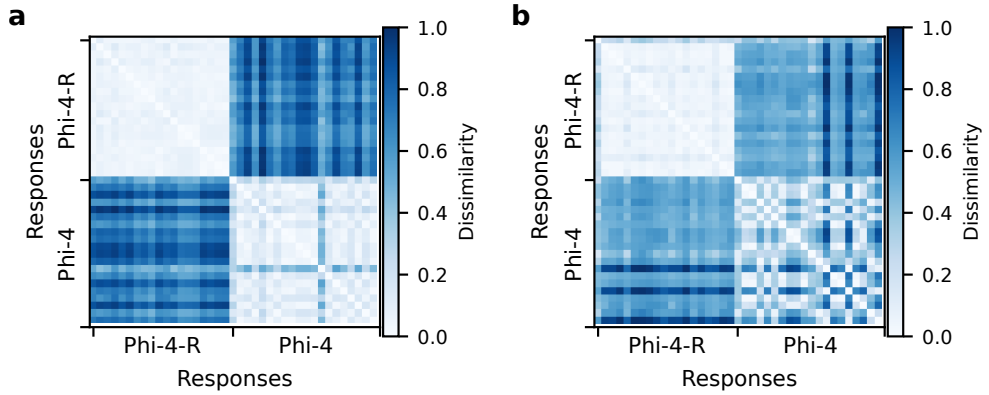

Figure 7. Dissimilarity in algorithmic primitive frequency for a clustering model fitted on TSP with **a** 9 and **b** 13 cities. In both cases, responses are clearly clustered according to model. Furthermore, especially on 13 cities, responses by Phi-4 appear to be split into two distinct groups while responses by Phi-4-R are much more stereotyped.

A.6.2. Clusters more common in Phi-4

Cluster 49: `solve_equation`. This cluster is mostly active when Phi-4 solves equations, whereas it is much less active on Phi-4-Reasoning.

A.7. Responses by Phi-4 and Phi-4-Reasoning on GraphNav

A.7.1. Clusters more common in Phi-4-Reasoning.

Cluster 2: `reasoning_scaffold`. This cluster is involved in structuring the overall reasoning in Phi-4-Reasoning.

Cluster 22: `plan_final_answer`. Phi-4-Reasoning commonly plans its final answer during the thinking section. This cluster is active in that section.

Cluster 9: `recall_instructions`. This cluster is involved in Phi-4-Reasoning recalling its instructions.

A.7.2. Clusters more common in Phi-4.

Cluster 23: `breadth_first_search`. This cluster is specifically active when the model plans and implements a breadth-first search algorithm. This indicates that Phi-4 is more likely to do this.

Cluster 48: `reasoning_scaffold`. This cluster is involved in structuring the overall reasoning in Phi-4, but not Phi-4-Reasoning.

A.8. Responses by Phi-4 and Phi-4-Reasoning on more complex versions of TSP

To investigate whether our method could also be applied to more complex versions of TSP, we additionally applied our Primitive Tracing pipeline to a version of TSP with 9 or 13 cities. This allowed us to both contrast the most common primitives in Phi-4 vs. Phi-4-Reasoning (Table 3 for 9 cities and Table 5 for 13 cities) and to investigate the most expressive clusters across both models (Table 4 for 9 cities and Table 6 for 13 cities). Overall, we found that as task complexity increases, the primitives remain both interpretable and semantically coherent enough to separate Phi-4 from Phi-4-Reasoning (Fig. 7). Moreover, across both task complexities, Phi-4 used more brute-force primitives, while Phi-4-Reasoning used more iterative comparisons. Notably, as task complexity increases, both models increasingly use a nearest neighbor motif (which is less common for Phi-4 in less complex versions of TSP).

A.9. Responses by Qwen3-8B on TSP

To compare the results of our pipeline for other models, we additionally generated responses for TSP from Qwen3-8B and applied our pipeline to those responses. We considered TSP with 6 cities and used the recommended parameters for text generation with Qwen (temperature $= 0.6, \text{top_p} = 0.95, \text{top_k} = 20$). We capped the response at 8096 tokens. Table 7 depicts the most expressive cluster within each of the five meta-clusters. In particular, the cluster in meta-cluster

Table 3. Clusters with the largest differences between Phi-4 and Phi-4-Reasoning on 9 cities, together with their χ^2-difference. Large positive values reflect clusters that arise more commonly in Phi-4-Reasoning; large negative values reflect clusters that arise more commonly in Phi-4. The most common clusters in Phi-4-Reasoning include clusters focused on exploring alternative routes and comparing those routes. The most common clusters in Phi-4 include clusters focused on brute-force solutions.

Context-centric label	Token-centric label	χ^2-Difference
Distance-based decision-making for route optimization.	Logical reasoning and decision-making in route selection.	0.0430
Evaluating and comparing route distances.	Comparison and evaluation of route distances.	0.0407
Route optimization through reordering and swapping city sequences.	Exploring alternative routes and configurations.	0.0366
Incremental distance summation for route optimization.	Intermediate sum calculations	0.0268
Incremental cost calculations and path sequences.	Delimiters for cost calculations and route segments	0.0265
Distance calculations between nodes in a graph.	Distance values in TSP calculations	-0.0311
Distance calculations and permutations for route optimization.	Placeholders or missing data.	-0.0343
Finding shortest route visiting all cities once, returning to start.	Traveling Salesman Problem (TSP) related terms	-0.0634
Generating permutations for brute-force TSP solutions.	Brute-force and heuristic approaches for solving TSP.	-0.1361
Distance calculations between cities.	Punctuation and separators in data structures.	-0.1541

Table 4. Clusters with the largest minimal frequency across both Phi-4 and Phi-4-Reasoning on 9 cities. These clusters include general nearest-neighbor motifs and distance calculations.

Context-centric	Token-centric	Minimal frequency
Selecting nearest unvisited city based on distance.	City and distance selection	0.0364
Generating permutations for brute-force TSP solutions.	Brute-force and heuristic approaches for solving TSP.	0.0254
Calculating and summing distances between cities.	Distance values in equations or comparisons	0.0241
Distance calculations and permutations for route optimization.	Placeholders or missing data.	0.0185
Distance calculation and route optimization.	Route separators and list delimiters	0.0178

0 is responsible for determining general strategies, the cluster in meta-cluster 2 is responsible for representing distances between city pairs, the cluster in meta-cluster 3 is responsible for retrieving those distances, and finally the cluster in meta-cluster 4 is responsible for summing them up. This echos motifs identified on Phi-4-Reasoning. (Note that the cluster in meta-cluster 1 appears to consist of the "Thinking process markers" which also mirrors a similar cluster in Phi-4-Reasoning.)

B. Experimental Setup Details

B.1. Traveling Salesperson Problem

Prompt: The traveling salesman problem (TSP) is a classic optimization problem that aims to find the shortest possible route that visits a set of cities, with each city being visited exactly once and the route returning to the original city.

Table 5. Clusters with the largest differences between Phi-4 and Phi-4-Reasoning on 13 cities, together with their χ^2-difference. Large positive values reflect clusters that arise more commonly in Phi-4-Reasoning; large negative values reflect clusters that arise more commonly in Phi-4. Most common clusters on Phi-4-Reasoning reflect iterative route optimization and comparisons. Most common clusters on Phi-4 reflect brute-force solutions and, interestingly, a cluster reflecting nearest-neighbor heuristics as well.

Context-centric label	Token-centric label	χ^2-Difference
Distance comparisons for route optimization.	Algorithmic decision-making and reasoning tokens	0.0347
Route optimization and evaluation through iterative adjustments.	Exploring alternative routes and adjustments	0.0276
Exploring alternative routes and segment reordering for optimization.	Exploration and optimization suggestions	0.0230
Path sequences in traveling salesperson solutions.	Path separators and list delimiters	0.0205
Incremental distance calculations and cumulative sum updates.	Cumulative totals in route calculations	0.0203
Selecting nearest unvisited city based on distance.	Decision-making and route selection in TSP solutions.	-0.0155
”Nearest neighbor heuristic for path optimization.”	City and distance selection	-0.0240
Distances between cities in traveling salesperson problems.	Delimiters and separators in data or calculations	-0.0386
Heuristic and brute-force TSP solutions.	Heuristic and algorithmic strategies for solving TSP.	-0.0691
Nearest neighbor heuristic for solving the Traveling Salesman Problem.	Traveling Salesman Problem terminology and operations	-0.0731

Table 6. Clusters with the largest minimal frequency across both Phi-4 and Phi-4-Reasoning on 13 cities. These clusters include a nearest-neighbor heuristic and clusters involved in distance calculations.

Context-centric	Token-centric	Minimal frequency
Nearest-neighbor heuristic for route optimization.	Punctuation and separators in routes and distances.	0.0519
Distance calculations between cities in a route.	City indices in routes and distances	0.0283
Distances between cities in traveling salesperson problems.	Delimiters and separators in data or calculations	0.0275
City-to-city distance calculations and comparisons.	Distance assignment or comparison operators	0.0271
Selecting nearest unvisited city based on distance.	Decision-making and route selection in TSP solutions.	0.0266

You must find the shortest path that visits all cities. The distances between each pair of cities are provided. Please list each city in the order they are visited. Provide the total distance of the trip. The final output of the result path and total distance wrapped by the final answer tag, like `{<final_answer>{'Path': '0->1->2->...->N->0', 'TotalDistance': 'INT_TOTAL_DISTANCE'}</final_answer>}`

The distances between cities are below: The path between City 0 and City 1 is with distance 44. The path between City 0 and City 2 is with distance 45. The path between City 0 and City 3 is with distance 45. The path between City 0 and City 4 is with distance 42. The path between City 0 and City 5 is with distance 37. The path between City 1 and City 2 is with distance 36. The path between City 1 and City 3 is with distance 27. The path between City 1 and City 4 is with distance 28. The path between City 1 and City 5 is with distance 29. The path between City 2 and City 3 is with distance 32. The path between City 2 and City 4 is with distance 38. The path between City 2 and City 5 is with distance 42. The path between City 3 and City 4 is with distance 46. The path between City 3 and City 5 is with distance 31. The path between

Table 7. Labels of the most expressive cluster in each meta-cluster for responses generated by Qwen3-8B. These clusters reflect general strategy determination, distance representation, retrieving those distances in calculations, and computing intermediate sums in calculations. Given the fact that the latter three clusters all correspond to simple integers (just in different contexts), this replicates the fact that our clustering algorithm sheds light on model reasoning beyond simple token-level analysis.

Meta-cluster	Token-centric	Context-centric	Expressivity
0	Approaches and strategies for solving TSP.	"Exploring permutations and heuristics for small TSP instances."	2.19
1	Thinking process markers	Problem-solving initiation for Traveling Salesman Problem (TSP).	0.11
2	Placeholder for missing data or values	Distances between city pairs in the Traveling Salesman Problem.	0.26
3	Single-digit numbers in calculations or sequences	Calculating and comparing path distances in TSP solutions.	0.38
4	Intermediate sum values in calculations	Incremental cost calculations for path evaluation.	0.32

City 4 and City 5 is with distance 26.

B.2. American Invitational Mathematics Examination (AIME)

LLMs were presented with problems from the AIME benchmark, which tests various domains of mathematical reasoning such as algebra, geometry, number theory, and combinatorics. The correct answer for all AIME questions is an integer from 0-999.

Example prompts (from 2025 AIME I):

- **Problem 1:** Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.
- **Problem 2:** On $\triangle ABC$ points A, D, E, and B lie in that order on side $\overline{AB}$ with $AD = 4$, $DE = 16$, and $EB = 8$. Points A, F, G, and C lie in that order on side $\overline{AC}$ with $AF = 13$, $FG = 52$, and $GC = 26$. Let M be the reflection of D through F, and let N be the reflection of G through E. Quadrilateral $DEGF$ has area 288. Find the area of heptagon $AFNBCEM$.

B.3. 3-literal Satisfiability Problem (3-SAT)

Models were tasked with determining the satisfiability of various 3-SAT problems, a class of NP-hard problems requiring combinatorial reasoning. If a model responded that a problem was satisfiable, they were required to provide the specific literals that would achieve satisfiability.

B.4. Graph Navigation (GraphNav)

Models were tasked with identifying the shortest path between two nodes in binary trees of varying depth (2-6, corresponding to 7-127 nodes). Each node was a randomly generated integer from 1-200, and edge lists were presented in randomized order. We included a 'forward' condition, where the initial node was the root and the goal was a randomly selected leaf, and a 'reverse' condition where the initial node was a randomly selected leaf and the goal was the root.

Forward Direction Prompt Example: Given the following list of connected rooms, someone wants to get to 91 from 114. The initial room and other rooms are denoted by numbers. `114->45, 114->90, 45->167, 45->91, 90->49, 90->9`. Starting at 114, what is the shortest path of rooms to visit if someone wants to arrive at 91? Include the final response in parentheses as the list of rooms separated by commas.

Reverse Direction Prompt Example: Given the following list of connected rooms, someone wants to get to 63 from 119. All of the rooms are denoted by numbers. `164->63, 119->147, 52->147, 54->164, 147->63, 62->164`. Starting at 119, what is the shortest path of rooms to visit if someone wants to arrive at 63? Include the final response in parentheses as the list of rooms separated by commas.

B.5. Generated model responses

For TSP, AIME, and 3-SAT we used previously generated model responses (Balachandran et al., 2025); for GraphNav, we generated responses from Phi-4 and Phi-4-Reasoning ourselves, using default generation parameters (temperature: 0.8, top_k: 50, top_p: 0.95).

B.6. Primitive vector extraction

To extract primitive vectors we first compute the average indirect effect (AIE), as described in Section 2.3 of Todd et al. (2024), averaging over the same set of six in-context learning tasks (antonym, capitalize, country-capital, english-french, present-past, singular-plural). We then pick the 35 attention heads with the largest AIE, denoting them by $\mathcal{A} \subseteq \{(l,h) | l = 1,\ldots,40, h = 1,\ldots,40\}$, where l denotes the layer of the corresponding attention head and h denotes its particular index. We consider a set of $n = 200$ responses in total (100 responses from Phi-4 and 100 responses from Phi-4-Reasoning). For each response $j = 1,\ldots,n$, we extract the activities in every attention head on this response, $Z^{(j)} \in \mathbb{R}^{T_j \times L \times H \times D}$, where $L = 40$ is the number of layers, $H = 40$ is the number of attention heads, T_j is the number of tokens in response j, and D is the residual dimension ($D = 5420$). Importantly, to compute $Z^{(j)}$, we project the attention head activations back into the residual dimension using the projection weights following that attention heads (this is the same method implemented by Todd et al. (2024)). Like Todd et al. (2024), we do not apply any further scaling or normalization. For a given cluster c, we denote all tokens that are a part of this cluster by $T_j[c] \subseteq \{1,\ldots,T_j\}$. We then compute the average activity in each attention head over all those tokens:

$$\overline{Z}[c] := \frac{1}{\sum_{j=1}^{n} |T_j[c]|} \sum_{j \in T_j[c]} Z^{(j)} \in \mathbb{R}^{L \times H \times D}. \tag{7}$$

Finally, we compute the primitive vector $v^{(p)}[c]$ corresponding to this cluster as the average across the 35 attention heads with the largest AIE:

$$v^{(p)}[c] = \frac{1}{|\mathcal{A}|} \sum_{l,h \in \mathcal{A}} \overline{Z}_{lh} \in \mathbb{R}^{D}. \tag{8}$$

B.7. Time complexity

Our time complexity is determined by two main processes:

1. **CoT Generation + Activation Collection:** $O(L^2)$, where L is the sequence length (number of tokens in the reasoning trace). Complexity is quadratic because Transformer self-attention computes all token-to-token interactions. This happens once per response to generate the trace and extract internal activations. Typical responses contain between 500 and 6000 tokens; at that scale, the activation collection is the most expensive step for our method, taking between 5–10 minutes on 4 A40 GPUs (we note that multiple GPUs are required to analyze long responses).

2. **Clustering:** $O(K \times N \times I)$, with K the number of clusters ($K = 50$ usually), N the number of datapoints (activations collected), and I the number of iterations until convergence. Scaling is linear with each factor. This is much faster than activation collection: we measured 61 seconds for 10 responses with 50 clusters (linear, $\tilde{1}$ min with a CPU).

C. Automated LLM Annotation

We developed an automated pipeline for labeling our clusters, which corroborates our manual annotations. Specifically, for each cluster we randomly sampled 50 tokens, included the 15 preceding and following tokens as context, and formatted these into 50 examples for GPT-4o. Within each example, we highlighted the critical token using the notation `**[critical token]**`. Examples were ordered by response and then by token index within each response.

We include four examples below from Phi4-Reasoning's Cluster 41, which illustrate a heterogeneous set of tokens that are all used during the evaluation and comparison of reasoning steps.

Example 1: `0->5->4->1->3->2->0 is**[ best]**.`

```
Perhaps:  0->5->1->3->4->2
```

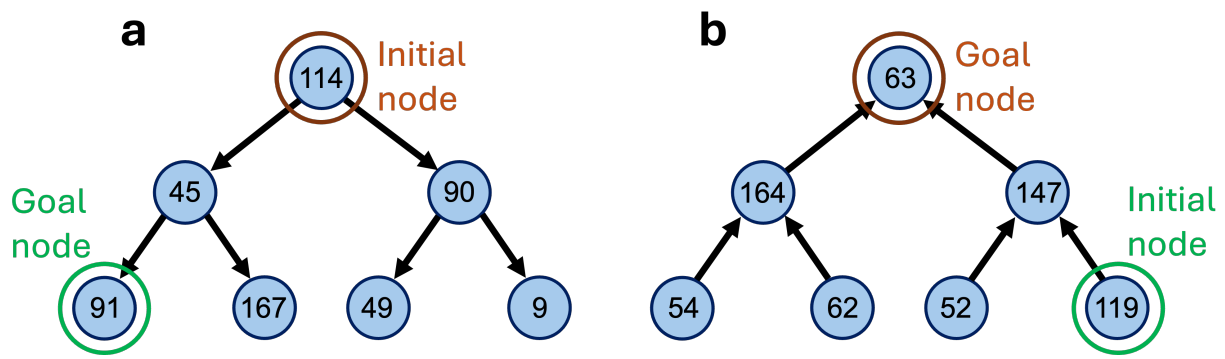

Figure 8. Examples of binary trees used in the two conditions. **a** Forward condition: models were tasked with finding the shortest path from the root node to a randomly selected leaf node, as in the forward prompt above. **b** Reverse condition: models were tasked with finding the shortest path from a randomly selected leaf node back to the root node.

Example 2: `=121, 121+30=151, so that's 151**[,]** not better.`

`What if 0->2->5->1->4`

Example 3: `90. Remaining: F, then F->A: 50. But**[ wait]**, we still have F: from D, F: D->F =`

Example 4: `+12+11+45+17+21=120, we already**[ computed]** that.`

`Wait, check: 0->4->2->5->`

We employed three prompting methods to extract labels for each cluster: Prompt 1 emphasized the contextual usage of critical tokens, while Prompt 2 included each 31-token excerpt without highlighting the critical tokens. Prompt 3 served as an alternative to Prompt 1, placing greater emphasis on the critical tokens themselves, rather than their surrounding context. We found that Prompts 1 and 2 yielded the most accurate labels, aligning closely with our manual annotations. The full prompts are provided below:

Prompt 1 - Token-Centric: This text includes excerpts from a language model solving various traveling salesperson problems. It includes tokens highlighted with double asterisks and brackets (e.g., **[token]**), with the surrounding tokens provided as context. Consider the context in which each highlighted token appears, and provide a single phrase (10 words or fewer) that describes their algorithmic role or roles across the whole set. The descriptive label should be specific, because it will later be contrasted with labels for tokens from other excerpts. The label may relate to a specific type of operation, reasoning strategy, heuristic, organizational principle, class of words, numbers relevant to the problem, tokens that appear in a specific context, etc. Remember that the label should be based on the context of the highlighted tokens.

Prompt 2- Context-Centric: This text includes excerpts from a language model solving various traveling salesperson problems. Consider all of the provided examples, and provide a single phrase (10 words or fewer) that describes the algorithmic role or roles common across the examples. The descriptive label should be specific, because it will later be contrasted with labels for other excerpts. The label may relate to a specific type of operation, reasoning strategy, heuristic, organizational principle, class of words, numbers relevant to the problem, tokens that appear in a specific context, etc.

Prompt 3 - Token-Centric (Alternative to Prompt 1): This text includes excerpts from a language model solving various traveling salesperson problems. It includes various critical tokens highlighted with double asterisks and brackets (e.g., **[critical token]**), with the surrounding tokens provided as context. Consider each highlighted token in the text, and provide a single phrase (10 words or fewer) that describes the entire set of critical tokens or encapsulates their algorithmic role or roles. The descriptive label should be specific, because it will later be contrasted with labels for tokens from other excerpts. The label may relate to a specific type of operation, reasoning strategy, heuristic, organizational principle, class of words, numbers relevant to the problem, tokens that appear in a specific context, etc. Remember that the label should be based on the highlighted tokens.

C.1. Labels of all clusters for Phi-4-Reasoning responses on TSP

We used our automated pipeline to assign labels to all cluster on the clustering model fit to Phi-4-Reasoning responses on TSP (see Table 8), sorting these labels by their associated expressivity. We excluded all clusters that were expressed in fewer than two tokens on average, resulting in a total of 46 clusters. Our results highlight that our pipeline demonstrates a strong agreement with our manual labels and, furthermore, that our selected clusters generally have high associated expressivity. To better understand how many clusters correspond to distinct algorithmic primitives, we used an LLM-based pipeline to further sort them into specific groups. This revealed eight clusters that were better characterized by linguistic patterns than algorithmically distinct roles:

- **Algorithmic Clusters:**
  - High-Level Strategies: 18, 37, 41, 35, 26, 6
  - Arithmetic Operations: 15, 11, 34, 42, 4, 13, 16, 8
  - Distance/Cost Calculation: 9, 0, 17, 19, 22, 23
  - Path Construction/Evaluation: 3, 14, 1, 38, 25
  - Edge/Path Operations: 5, 27, 33, 45, 47
  - Route Optimization: 31, 39, 46, 44, 21, 10
  - Organizational/Workflow: 30, 43
- **Linguistic Clusters:**
  - Notation/Punctuation: 12, 29, 24
  - Identifiers/Labels: 7, 40, 20
  - Structural Markers: 32, 48

Table 8. Annotated labels for the clustering model fit to TSP, showing both manual labels and the two distinct automated labels. Clusters are ranked according their expressivity (number of unique tokens expressing the cluster, divided by the radius of the cluster in the latent space)

Rank	Cluster	Manual Label	Automated label (Token-centric)	Automated label (Context-centric)	Expressivity
1	18		Exploration of alternative routes and possibilities	Evaluating candidate routes and calculating total distances.	2.58
2	30	produce_final_answer	Output formatting with `<final_answer>` tag	Output formatting with `<final_answer>` tag.	1.17
3	41	compare_or_verify_2	Route exploration and modification suggestions	Route evaluation and improvement through swaps and recalculations.	1.11
4	37	nearest_neighbor	"Selecting best path based on edge weights"	"Selecting best path based on edge costs."	0.94
5	19		Path and distance notation	Calculating and comparing path distances.	0.82
6	7		Key terms in TSP problem description	Shortest path computation in Traveling Salesman Problem (TSP).	0.65
7	9		City and path identifiers	Distance calculations between city pairs.	0.56

Continued on next page

Table 8. (continued)

Rank	Cluster	Manual Label	Automated label (Token-centric)	Automated label (Context-centric)	Expressivity
8	35	compare_or_verify	Proposed alternative routes	Route optimization and evaluation in TSP solutions.	0.35
9	15	compute_distance	Summation and accumulation of path costs	Addition and summation of route distances.	0.32
10	40		City and path descriptors	City-to-city distance representation.	0.32
11	11		Intermediate cumulative sum calculations	Incremental distance calculations with comparisons to find optimal path.	0.30
12	25		Delimiters and separators in route descriptions	Evaluating and selecting paths based on distance or cost.	0.30
13	0		Distance values between cities	City-to-city distance calculations	0.29
14	6		Exploring alternative routes and strategies for optimization.	Route exploration and optimization through swaps and recalculations.	0.28
15	29		Punctuation or spacing in route calculations	Calculating and updating total distances in routes.	0.27
16	17		Distances between city pairs	Pairwise city distances in traveling salesperson problems.	0.25
17	26	generate_new_path	Proposed routes for alternative solutions	Route exploration and evaluation for optimization.	0.21
18	42		Intermediate sum in cumulative addition calculations	Incremental sum calculations for path cost evaluation.	0.20
19	12		Path and distance notation in TSP solutions	Path calculation and distance summation.	0.18
20	34		Intermediate sum values in calculations	Incremental addition of distances to calculate total path length.	0.15
21	45		Edge weight in TSP route calculations	Edge traversal and cost calculation in paths.	0.14
22	24		Colon and equals sign for distance assignment	Node-to-node distance calculations	0.13
23	22		City identifiers in path descriptions	Pairwise city distance calculations	0.13
24	33		Edge weights in path calculations	Incremental cost calculations and route suggestions	0.13
25	1		Node in a potential TSP route sequence.	Proposing alternative routes and computing distances.	0.12
26	32		Next node in path sequence	Node-to-node distance calculations	0.12

Continued on next page

Table 8. (continued)

Rank	Cluster	Manual Label	Automated label (Token-centric)	Automated label (Context-centric)	Expressivity
27	43		Instructional and structural language for problem-solving process	Structured reasoning and solution presentation.	0.12
28	20		City identifiers in routes and calculations	Path selection and distance calculation.	0.12
29	38		Node in a potential route path.	Evaluating and comparing potential routes and distances.	0.10
30	47		Separating route segments and calculations	Edge weights and path calculations	0.10
31	3		Route segment in candidate solutions	Evaluating and updating candidate routes based on distance calculations.	0.09
32	5		Edge weights in path calculations	Edge weights and path sequences in TSP solutions	0.09
33	14		Path traversal representation	Path construction and evaluation	0.09
34	23		Route termination or cycle completion indicator	Proposing alternative routes and calculating their distances.	0.08
35	4		Addition operation in arithmetic calculations	Incremental sum calculation with intermediate results	0.07
36	13		Addition operation in distance calculations	Incremental distance calculation for route optimization.	0.07
37	27		Edge weights in path calculations	Sequential path calculations with cumulative distance summation.	0.07
38	16		Addition operation in cost calculations	Partial sum calculations for route cost evaluation.	0.06
39	31		Route transition indicator in path sequences	Route exploration and evaluation in TSP solutions.	0.05
40	48		Indicating direction or transition between nodes.	Node-to-node path with distances	0.05
41	8		Equality check in arithmetic operations	Incremental distance calculations for route optimization.	0.05
42	46		Route transition indicator	Route exploration and evaluation for optimization.	0.03
43	39		Route transition indicator	Route exploration and evaluation for optimality.	0.03

Continued on next page

Cluster	Automated label (Token-centric)	Automated label (Context-centric)	χ^2-difference
89	Punctuation and separators in problem-solving context	Selecting shortest path or edge based on distance.	0.04
67	Exploration of alternative routes and solutions.	Route optimization through permutation and manual inspection.	0.03
54	Route and distance calculation punctuation	Route evaluation and distance calculation.	0.02
22	Comparative or superlative evaluation of routes or distances.	Route exploration and optimization in TSP solutions.	0.02
48	Route evaluation and comparison indicators	Route exploration and evaluation for optimization.	0.02
1	Placeholders for missing or irrelevant information	City-to-city distance calculations	-0.02
58	Route representation in traveling salesperson problem solutions.	Node sequences representing potential solutions	-0.02
64	Punctuation and formatting for path or route descriptions.	Node sequence representation	-0.03
15	Placeholder for missing distance values	Summing distances for paths or routes.	-0.05
59	Brute-force approach for small number of cities (6).	Brute-force permutation evaluation for small-scale TSP solutions.	-0.06

Table 9. Labels for clustering model fit to responses from Phi-4 and Phi-4-Reasoning with $k = 100$. We provide the five clusters with the largest positive χ-squared difference (indicating that they are more often used in Phi-4-Reasoning) and the five clusters with the largest negative χ-squared difference (indicating that they are more often used in Phi-4).

Table 8. (continued)

Rank	Cluster	Manual Label	Automated label (Token-centric)	Automated label (Context-centric)	Expressivity
44	44		Intermediate node in proposed paths	Evaluating and suggesting alternative routes for optimization.	0.03
45	21		Return to starting point in route.	Evaluating and calculating potential routes and their distances.	0.02
46	10		Starting point or reference city in TSP routes.	Evaluating and comparing potential TSP routes and distances.	0.02

C.2. Robustness of our pipeline to different numbers of clusters

To explore the robustness of our pipeline to varying the number of clusters, we identified automated labels for the clusters inferred on responses by Phi-4 and Phi-4-Reasoning at $k = 100$. We analyzed the five responses that were most strongly used in Phi-4-Reasoning and Phi-4 respectively, as indicated by their χ^2-difference. Notably, the top cluster in Phi-4-Reasoning expressed a nearest-neighbor approach, whereas the top cluster in Phi-4 expressed a brute-force approach. This replicates our insights at $k = 50$ and demonstrates the utility of our automated approach.

D. Nearest Neighbor Strategy Analysis

We compared the implementation of the nearest neighbor search heuristic on the Traveling Salesperson Problem (TSP) in Phi-4-Reasoning and Phi-4-Base. First, we computed the minimum edit distance between nearest neighbor solution and the optimal solution and analyzed how this distance relates to accuracy. We computed these minimal NN-optimal edit distances using any city as the starting point (Figure 9 a) and with city 0 as the starting point (Figure 9 b). The latter edit distance was chosen, because in practice, we observed that the models frequently used city 0 as the initial node.

Next, we tested whether one of the first five generated paths is a nearest-neighbor solution; we sampled from the first five paths to account for idiosyncrasies in the model's output (e.g., repeating path segments from the prompt). Additionally, we

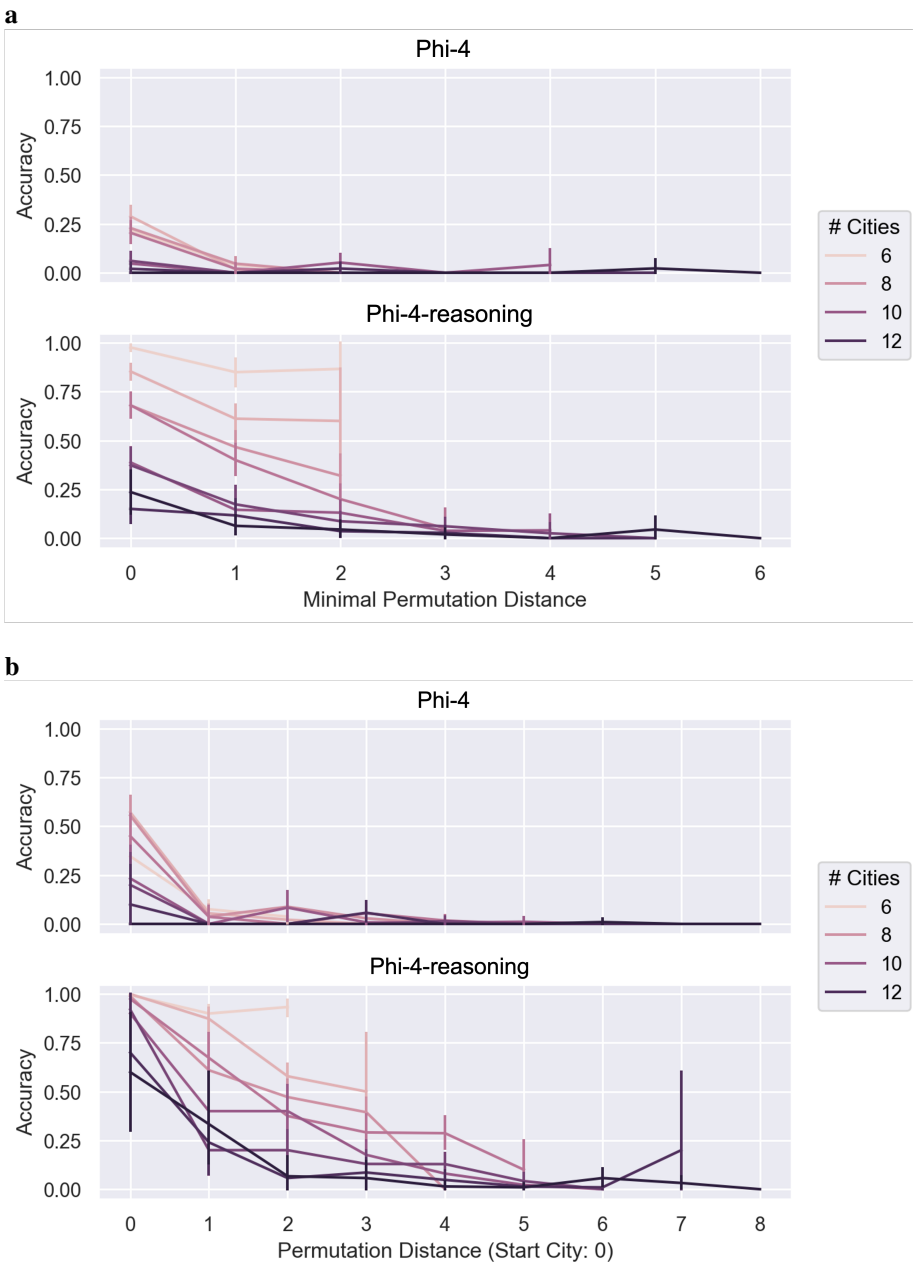

Figure 9. **a** The minimal NN-optimal edit distances with any city as the starting point. **b** The minimal NN-optimal edit distance using city 0 as the starting point. We observed a strong trend between edit distance and accuracy in **b**, even when controlling for the number of cities (and a somewhat weaker trend in **a**). This trend was especially strong for Phi-4-Reasoning, suggesting this model may effectively implement the nearest neighbor heuristic beginning at city 0.

examined the relationship between the number of considered paths and the average distance from the NN solution.

Overall, these findings demonstrate that Phi-4-Reasoning has a strong tendency to implement the nearest neighbor search heuristic. Specifically, the model tends to begin with the NN solution starting at city 0 and iteratively edit the path to minimize its total distance. While Phi-4-Base does not invoke the nearest neighbor heuristic as reliably as Phi-4-Reasoning, it does to some degree as shown in Figures 9 b and 10 a.

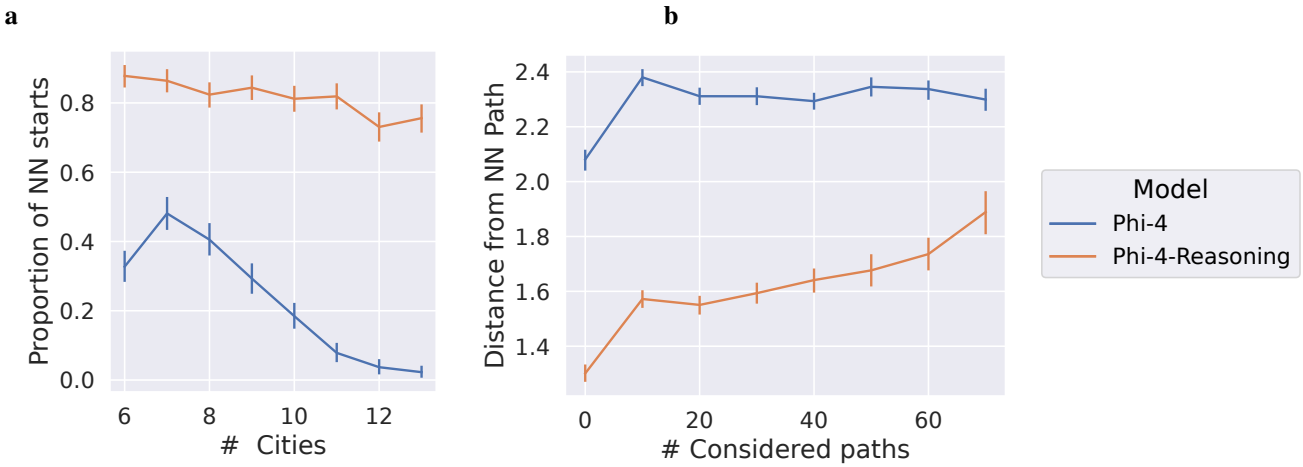

Figure 10. **a** We found that Phi-4 reasoning begins with the NN heuristic much more frequently than Phi-4-Base, especially when >8 cities are present. **b** Paths generated by Phi-4-Reasoning tend to have much lower distance on average from NN paths than those generated by Phi-4. Additionally, in Phi-4-Reasoning edit distance is lowest among the initial paths and increases as more candidate paths are selected.

E. Identifying and Composing Function Vectors

E.1. Tasks

We generated a set of algorithmic in-context learning tasks that require specific operations over graphs.

Terminal node recognition (TNR), identifies the final node in a path (e.g., given the input "path: T-P-Q-V", the model is tasked with outputting the token 'V'). Reward comparison (RC) compares rewards (represented by numbers between 1 and 100) across candidate nodes (e.g., given the input "rewards=[M:100 vs S:44]", the model is tasked with outputting 'M'). The evaluation task in Fig. 5a required combining both primitives: given two paths, the model was prompted to return the node that contains the highest reward.

Example Prompt (Correct Answer: A)

path1: B-O-Q-D-A.
path1-rewards=[A:65 vs Y:75].
path2: C-W-V.
path2-rewards [V:15 vs Y:45]

Relatedly, `get_first_node` and `get_last_node` requires responding with the first or last node of a presented path (each node consists of numbers or capital letters and has between three and six elements). `get_predecessor` and `get_successor` receive a path and a node has input and need to return the predecessor or successor node respectively.

Example (`get_successor`)
Input: Graph: D-C-N-J, Node: C
Output: N

E.2. Detailed methods and results

We extracted function vectors corresponding to each of these tasks using the same approach as (Todd et al., 2024). We identified the 35 attention heads with the highest average indirect effect (see definition in (Todd et al., 2024)) for Phi-4 and Phi-4-Reasoning and 20 attention heads the highest average indirect effect for Llama-3-8B. We injected these function vectors across all layers and analyzed the layer with the largest effect. For the Qwen models, we used 35 attention heads for the 4B and 8B models, and 45 attention heads for the 14B models. Figure 12 depicts the effects on Phi-4, Phi-4-Reasoning, and Llama-3-8B, Figure 13 depicts the effects on the Qwen models.

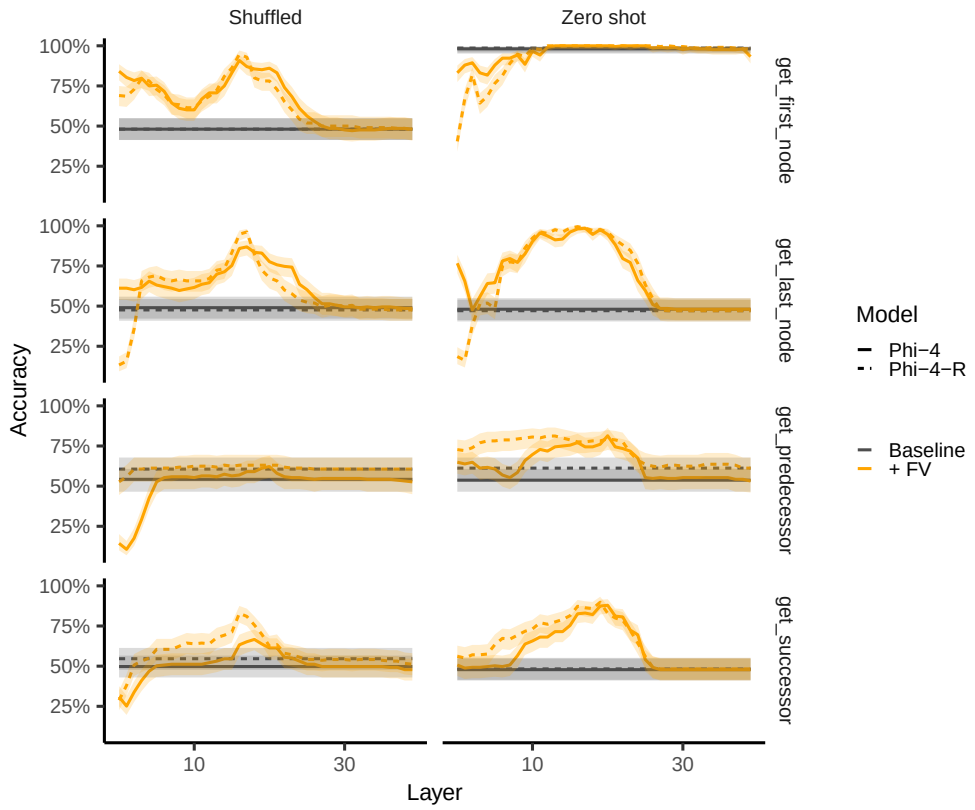

Figure 11. Accuracy on different graph operations after injecting the corresponding function vector into Phi-4 or Phi-4-Reasoning, either after shuffled in-context examples are in a zero-shot setting.

F. Behavioral hallmarks on TSP

We consider three setups for injecting primitive vectors into TSP. In Figs. 3b,d, we only present the model with the TSP prompt and inject the primitive vectors throughout generation. We maximally generate 500 tokens. This is not sufficient to reliably generate a full answer, but is sufficient to test if the behavior is expressed. In Fig. 3c and Table 2, we add a sequence of randomly generated paths to the assistant response before injecting the function vectors. In both of these cases, we define several measures of different behavioral hallmarks. For these measures we automatically extract the set of paths generated in the model response using a regular expression. Below we specify our operational definitions of the different behavioral hallmarks:

1. % NN paths: What proportion of generated paths corresponds to a nearest-neighbor heuristic?
2. # Unique paths: How many valid TSP candidate paths are generated (after removing duplicate paths)?
3. Distance computation: What is the earliest token at which a distance computation occurs (operationalized as a sum of several numbers)? A lower number thus corresponds to a stronger expression of this behavior. If no distance computation occurs in the entire response, we set the value to 512.
4. Final answer: At what token does the model generate the `<final_answer>` token?
5. # Verifications: How many verification-related words are mentioned in the response?
6. # Comparison: How many comparison-related words are mentioned in the response?

Overall, our behavioral hallmarks span increasing levels of complexity:

1. Token matching to detect behavior via explicit tokens or phrases (e.g. for "# Verifications" or "# Comparisons")

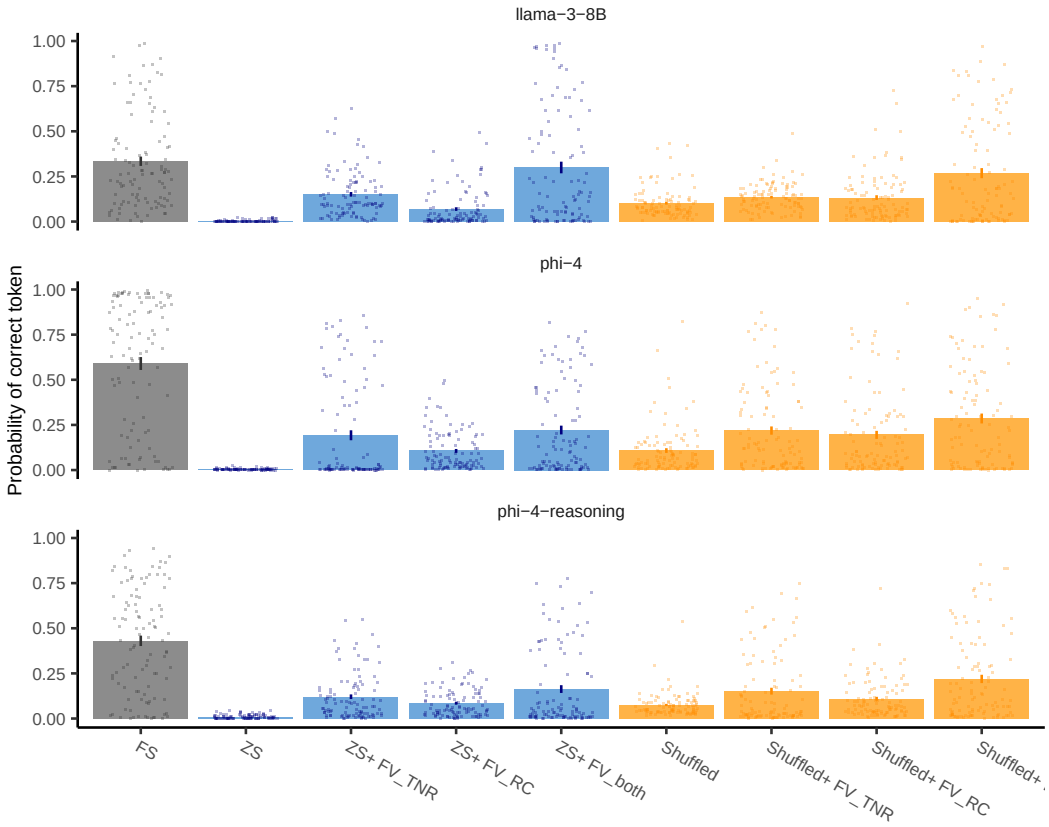

Figure 12. Average probability assigned to the correct tokens when injecting the Terminal_node_recognition and Reward_comparison FVs (FV_TNR and FV_RC) and the sum of both (FV_both). We compare the performance across Llama-3-8B, Phi-4, and Phi-4-Reasoning, and across a corrupted ICL and zero-shot context.

2. Token matching and positional signals to identify when a token is generated (e.g. for "Final answer")
3. Extracting structured output via regex + parsing rules to extract objects (e.g. valid paths, numbers; "% NN paths", "Distance computation")
4. Like 3., but with additional filtering, e.g. unique sets in "# Unique paths"

Finally, we provided an additional control evaluation for the "Final answer" behavioral hallmark, evaluating whether the model produces a parsable JSON object with a valid path. This was intended to ensure that injecting the `produce_final_answer` primitive vector genuinely produced a valid answer rather than simply causing the model to generate the specific `<final_answer>` without any additional effects. This analysis was consistent with the findings in Table 3, confirming that the primitive vector induces genuine algorithmic execution (Table 10).

Table 10. More procedural metric for returning the final answer, testing whether the answer is correctly formatted, parsable as a JSON object, and contains a valid path.

	Valid final ans. ↓
`nearest_neighbor`	-4.7%
`generate_new_path`	+15.1%
`compute_distance`	-18.9%
`produce_final_answer`	**-62.1%**
`compare_or_verify`	-7.3%
`compare_or_verify_2`	-9.9%

Finally, when evaluating the impact of the `get_first_node` and `get_last_node` primitive vectors (Fig. 5c), we automatically extract a generated path from the middle of the reasoning trace (constraining our selection to only consider

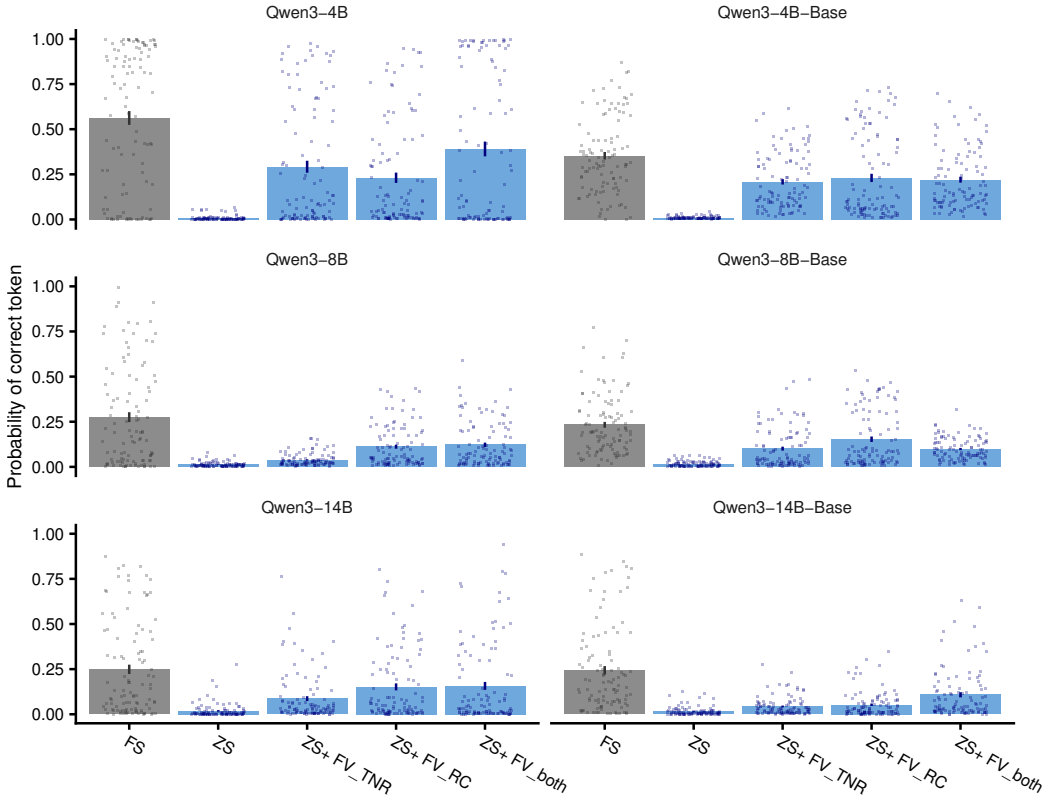

Figure 13. Average probability assigned to the correct tokens when injecting the Terminal_node_recognition and Reward_comparison FVs (FV_TNR and FV_RC) and the sum of both (FV_both). We compare the performance across different versions of Qwen.

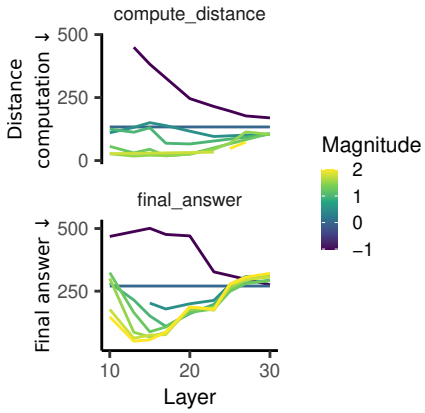

Figure 14. Injecting primitive vectors for `compute_distance` and `final_answer` in the middle of the reasoning trace increases relevant behavioral hallmarks in the output.

paths where the first and last node are not identical). We then generate the model response starting immediately after the generated path and injecting the corresponding primitive vector throughout. We generate 20 tokens and assess whether the first node that is mentioned corresponds to the first node of the path, the last node of the path or some other node.

F.1. Comparison to Bag-of-Tokens explanation

To control for the potential alternative explanation that the injected primitive vectors simply boost a "bag of tokens," increasing the probability of specific tokens, we also injected the primitive vector at the end of the residual stream, directly before the unembedding. We reasoned that if the primitive vector simply boosts a bag of tokens, the algorithmic expression should be evoked with similar strength by this procedure. We computed the maximal effect of this intervention across a

wide range of magnitudes ($\alpha = 0.5, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0, 4.0, 10.0, 100.0, 1000.0$) and subtracted it from the effect obtained by injecting the primitive vector into intermediate layers. We found that each primitive vector most strongly induced its corresponding behavioral hallmark even with this control, indicating that this behavior is caused by a latent procedure (Table 11).

Table 11. Effects of primitive vector injection on behavioral hallmarks (% above baseline), controlling for the effect of boosting a "bag of tokens". For each cell, we identify the maximal effect across all positive magnitudes and all intervention layers (10, 13, 15, 17, 20, 30) and subtract the maximal effect across all positive magnitudes of injecting the primitive vector at the end of the residual stream. Bold = strongest effect, underline = second strongest per column. *(%NN paths: proportion of nearest neighbor paths generated. Dist. comp.: Distance computation. Final ans.: Final answer. #Verif.: number of Verifications in the output. #Comps: number of comparisons in the output. See Appendix F for detailed definitions.)*

	%NN paths ↑	#Paths ↑	Dist. comp. ↓	Final ans. ↓	#Verif. ↑	#Comp. ↑
nearest_neighbor	**+33.2%**	+47.3%	-42.2%	-18.6%	+9.8%	+141.5%
generate_new_path	+0.0%	**+73.9%**	-19.7%	-0.3%	-213.0%	-7.3%
compute_distance	+10.1%	+33.7%	**-44.8%**	-50.5%	+128.3%	+103.7%
produce_final_answer	+12.8%	+34.0%	-21.2%	**-63.5%**	+47.8%	-1.2%
compare_or_verify	+7.9%	+38.5%	-38.0%	-20.2%	**+559.8%**	**+1059.8%**
compare_or_verify_2	+5.9%	+0.8%	-24.3%	-24.7%	+516.3%	+290.2%

F.2. Analysis of interference between primitive vectors

In Table 2 each cell had its own optimized injection magnitude α and injection layer L (maximizing expression for that specific primitive-task combination). To investigate the specificity of algorithmic primitives, we additionally investigated the effect of injecting the algorithmic primitive at the specific α and L that optimizes that primitive's corresponding behavioral hallmark (Table 12).

Primitive vectors were substantially more disentangled in this analysis. The analysis reveals that the entanglement of some primitives seems to follow the sequential order of their usual expression. For instance, injecting nearest_neighbor increases the expression of its algorithmic successor, compute_distance, but injecting the successor doesn't increase the expression of its predecessor (nearest_neighbor). This suggests that the entanglement of two primitives may be, at least partially, due to them being commonly expressed in specific sequential algorithmic motifs, with specific ordering.

Table 12. Effects of primitive vector injection on behavioral hallmarks (% above baseline). For each row, we identify the magnitude and layer maximizing the effect for that row's primitive and then evaluate the effect on all behavioral hallmarks in that row. Bold = strongest effect, underline = effects that increase expression of the behavioral hallmark.

	%NN paths ↑	#Paths ↑	Dist. comp. ↓	Final ans. ↓	#Verif. ↑	#Comp. ↑
nearest_neighbor	**+56.1%**	-4.2%	-34.9%	+6.3%	+40.2%	-22.0%
generate_new_path	-41.4%	**+143.9%**	-76.0%	+85.3%	-93.5%	-82.9%
compute_distance	-100.0%	-100.0%	**-86.5%**	+37.9%	+35.9%	+79.3%
produce_final_answer	-100.0%	-95.2%	+283.8%	**-81.5%**	-48.9%	-84.1%
compare_or_verify	-100.0%	-95.5%	+266.4%	+85.5%	+568.5%	**+1103.7%**
compare_or_verify_2	-100.0%	-97.7%	+280.2%	+76.5%	**+706.5%**	+325.6%

F.3. Analysis at fixed layer

We generally considered the maximal effect across different layers and injection magnitudes. To assess the robustness of our analysis, we also considered the effects of injecting the primitive vectors at a fixed layer and magnitude. We generally found that injecting primitive vectors at a layer slightly before the middle (e.g. layer 17) and at a magnitude slightly above 1 (e.g. 1.25) worked best. Table 13 assesses the effect of consistently injecting primitive vectors with these hyperparameters. Except for the nearest-neighbor motif, all primitive vectors reliably evoke the associated behavioral hallmark even when fixing these hyperparameters. The nearest-neighbor motif is generally more reliably evoked at a slightly earlier layer, highlighting potential differences across primitives in the layer in which they are invoked.

Table 13. Behavioral hallmarks at layer 17 with magnitude 1.25. Most hallmarks are largely produced by their associated primitive vector; with the only exception of the nearest-neighbor motif, which is more reliably produced in earlier layers. This highlights the fact that primitive induction may vary across different layers.

	%NN paths ↑	#Paths ↑	Dist. comp. ↓	Final ans. ↓	#Verif. ↑	#Comp. ↑
`nearest_neighbor`	+0.7%	-14.4%	-68.1%	+12.3%	-44.6%	-29.3%
`generate_new_path`	-16.3%	**+58.4%**	+17.2%	+88.1%	-97.8%	-95.1%
`compute_distance`	-100.0%	-99.2%	**-85.9%**	-14.3%	-63.0%	-63.4%
`produce_final_answer`	-70.5%	-59.5%	+92.4%	**-60.5%**	-90.2%	-82.9%
`compare_or_verify`	**+8.9%**	+15.6%	-62.8%	+56.0%	**-1.1%**	**+30.5%**
`compare_or_verify_2`	+5.8%	-2.5%	-64.9%	+40.0%	-26.1%	-8.5%

F.4. Reconstructing primitive vectors by their token outputs

To investigate whether primitive vectors can be reconstructed by their token decodings, we follow the approach in Todd et al. (2024): for a given primitive vector v, we apply the LLM's unembedding matrix to generate an output distribution, $Q_t := D(v)$, where D is the unembedding operation. $Q_t \in \mathbb{R}^T$ where T is the number of tokens. We then truncate this distribution to the top 100 tokens and apply the softmax to generate a probability distribution P_{t100} over the top-100 tokens. We then perform an optimization where $\hat{v}_{t100}$ is inferred as the vector approximating this probability distribution:

$$\hat{v}_{t100} = \arg\min_{v} \mathrm{CE}(v, P_{t100}), \tag{9}$$

where CE is the crossentropy loss. $\hat{v}_{t100}$ thus reflects the information about the primitive vector that is contained within its decoded vocabulary. We then used those inferred vectors to generate new outputs, computing the associated effect on the different behavioral hallmarks. We found that in almost all cases, their effect was smaller than that of the original primitive vector. This indicates that knowledge of the top tokens associated with a given primitive vector are not sufficient to reconstruct its effect and some additional information is needed. A notable exception is given by the impact of `compare_or_verify` on the number of comparisons, for which the inferred vector has an even stronger effect. This suggests that the effect induced by this primitive vector is captured by its top-decoded tokens. Overall, our analysis suggests that primitive vectors usually contain some information beyond what can be inferred from their top decoded tokens.

Metric	PV	v	$\hat{v}_{t100}$
% NN paths ↑	`nearest_neighbor`	**+56.1%**	+21.1%
# Paths ↑	`generate_new_path`	**+143.9%**	+62.0%
Dist. comp. ↓	`compute_distance`	**-86.5%**	-71.9%
Final ans. ↓	`produce_final_answer`	**-81.5%**	-16.2%
# Verif. ↑	`compare_or_verify_2`	**+706.5%**	+71.7%
# Comp. ↑	`compare_or_verify`	+1103.7%	**+2458.5%**

Table: Performance of the primitive vector (PV) (column v) compared to the reconstruction $\hat{v}_{t100}$ that matches the top 100 tokens (both columns show % above baseline). For each primitive vector, we considered the behavioral hallmark on which it had the strongest effect (see Table 2). In all cases except for the number of comparisons, the original PV induces a stronger effect.

F.5. Injection of AIME primitive vectors into TSP

Tab 14 Table 14 shows the results of injecting the different AIME primitive vectors into Phi-4 while it solves TSP. Notably, they all significantly increase expression of those behavioral hallmarks.

G. Hyperparameter Comparisons

To evaluate robustness to our chosen hyperparameters, we repeated the clustering analysis for different choices of layer, number of clusters, and number of responses.

Table 14. Behavioral hallmarks arising from injecting primitive vectors from AIME into TSP at layer 17 (magnitude 1.0). Asterisks indicate significance levels from a one-sided Mann-Whitney U test: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

Vector	NN Paths Ratio	Unique Paths
solve_equation	3.7% *	1.13 ***
verification	6.2% **	1.29 ***
plan_final_answer	2.8% *	1.18 ***
spatial_reasoning	6.1% **	0.97 ***

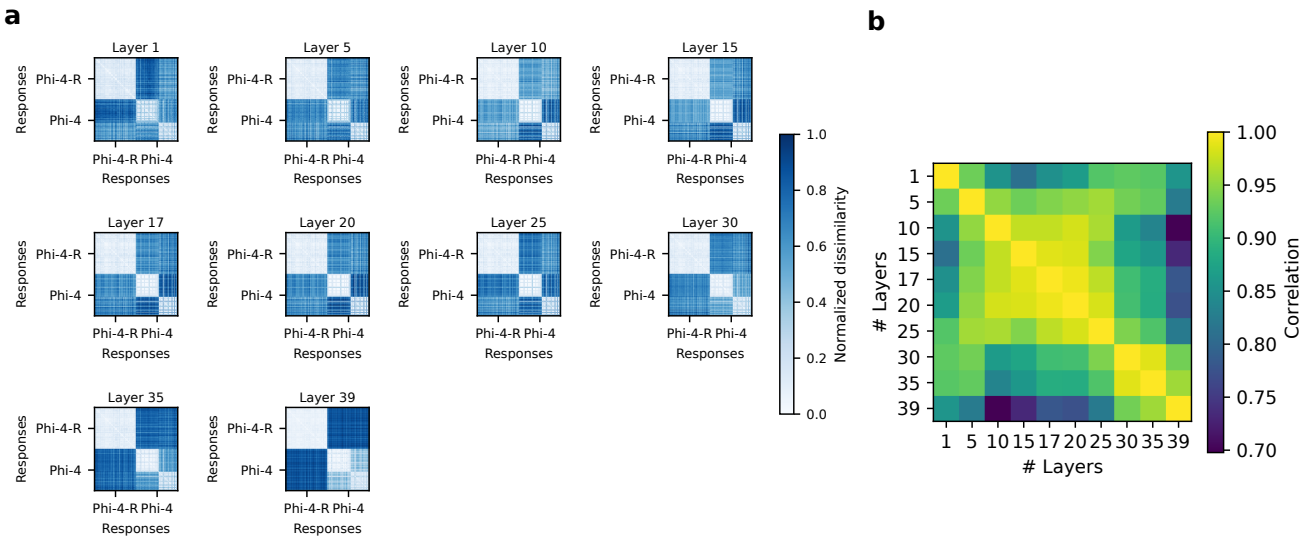

Figure 15. **a** Dissimilarity matrices between the frequency of clusters occurring in different responses, plotted for cluster analyses fitted to different layers. **b** Similarity between these different dissimilarity matrices. **c** Average dissimilarity between responses from different groups: 1) responses by Phi-4-Reasoning, which consistently employ a heuristic nearest-neighbor strategy (see Appendix D), 2) responses by Phi-4 that employ a brute-force strategy, 3) responses by Phi-4 that articulate a random guess. Intermediate layers emphasize similarities between the heuristic strategy employed by Phi-4-Reasoning and both strategies employed by Phi-4, whereas later layers mostly reflect dissimilarities between different models.

G.1. Number of layers

We first repeated the clustering analysis for all layers by increments of five in both Phi-4 and Phi-4-Reasoning when solving Traveling Salesperson (TSP). We then compared the frequency of each layer's clusters showing up both across responses within a model, and across the responses of the two models (the analysis presented in Fig. 4a) (see Fig. 15a). To evaluate uniquely informative patterns across the layers, we computed the correlation across the layer-wise dissimilarity matrices (Fig. 15b). Together, these analyses revealed broad patterns in the effect of layers: after the first layers, there was a large highly correlated cohort in the intermediate layers, especially layers 10-25 (centered on layer 17), and a smaller cohort of highly correlated layers, ranging from 30-40. We observed that intermediate layers captured more aspects of the reasoning hierarchy, as they more strongly reflected the differences between the brute-force and random guess strategies implemented by different responses from Phi-4 (see Appendix D). In contrast, later layers more strongly highlight the differences between Phi-4 and Phi-4-Reasoning. As we are interested in identifying the underlying algorithmic primitives, these findings justify layer 17 as a suitable candidate for clustering.

G.2. Number of clusters

To evaluate robustness to our choice of K, we conducted a similar analysis, this time keeping the layer fixed (17) and varying K by 5, 30, 50, and 100. As described above, we first computed dissimilarity matrices for frequency of cluster occurrences in within-model and cross-model responses per K (Fig. 16a), and then computed the correlation across these matrices (Fig. 16b). Applied to 100 responses to TSP per model (Phi-4 and Phi-4R), this analysis revealed high correlations

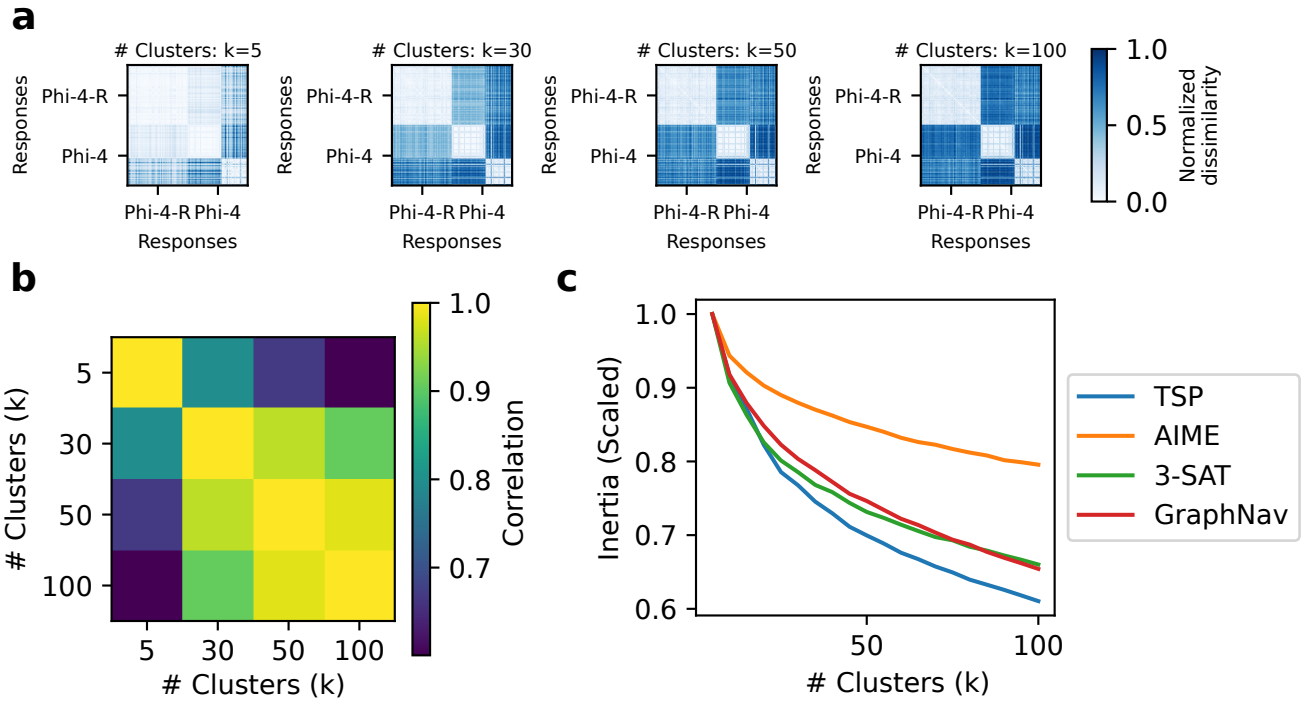

Figure 16. **a** Dissimilarity matrices as in Fig. 4a for different numbers of clusters. **b** Similarity between the dissimilarity matrices for different numbers of clusters, confirming a high degree of similarity for sufficiently many clusters. **c** Inertia (normalized by the inertia for five clusters for each task) plotted against the number of clusters across the five tasks.

among K=30, 50, and 100, with the highest Pearson correlation of >0.98 between K=50 and 100.

Next, we computed normalized inertia, or the sum of squared distance between each data point and its assigned cluster centroid for 4 tasks per $K = 5, 10, ..., 100$, with increments of 5 on the x axis (Fig. 16c). We then used the elbow method (Han et al., 2012) to identify an approximate K at which inertia dropped for each task. Together, these findings revealed $K = 50$ as the parsimonious and functionally appropriate choice, offering a reasonable tradeoff between explaining variance in the latent space and parsimony in the potential number of algorithmic primitives.

G.3. Number of samples

To evaluate robustness to sampling choice, we fixed the layer ($L = 17$) and $K = 50$, and evaluated the robustness of clustering to different numbers of responses, between 1 and 50. First, we found very high correlation between assigned clusters across varying number of responses with a minimum Pearson's correlation of 0.96 for using 1 response, and a correlation of 0.9945 between 5 and 25 or 50 responses (Fig. 17a,b). We then measured the consistency of assigned clusters across a varying number of responses with 2 measures: adjusted mutual information (MI) score and adjusted Rand score (Fig. 17c), revealing high consistency for clustering models fit to at least 5 responses. Taken together these findings show robustness of assigned clusters to varying number of responses included in the analysis.

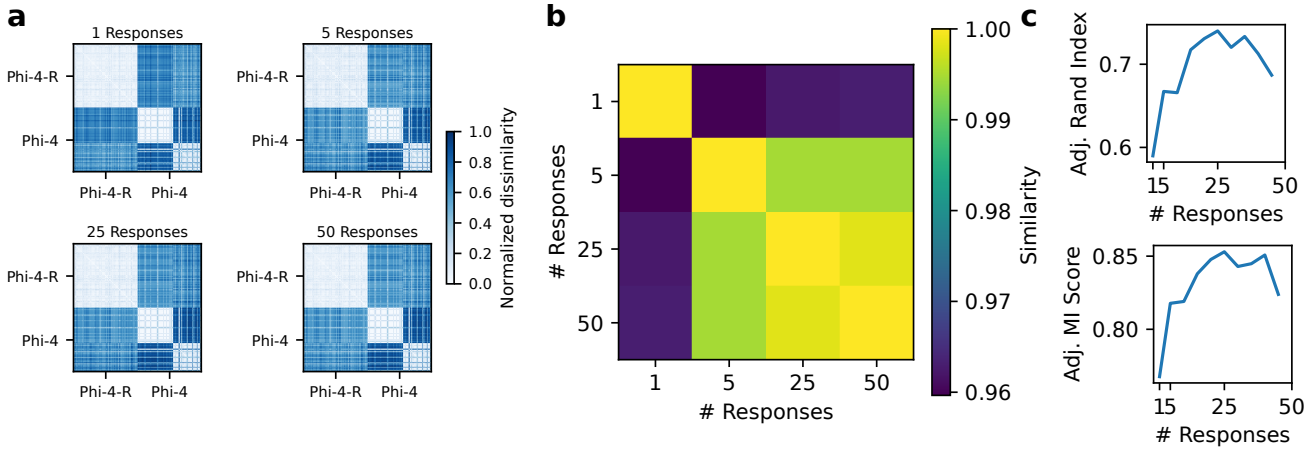

Figure 17. **a** Dissimilarity matrices between the algorithmic fingerprints of different responses, plotted for cluster analyses fitted to different numbers of responses. **b** Similarity between these different dissimilarity matrices. **c** Adjusted Rand Index and Adjusted Mutual Information Score (averaged across 100 responses) for clustering models fit to different numbers of responses.