A Neuroscience-Inspired Dual-Process Model of Compositional Generalization

Alex Noviello[1†] **Claas Beger**[1†] **Jacob Groner**[1] **Kevin Ellis**[1‡] **Weinan Sun**[2‡]
[1]Department of Computer Science, Cornell University
[2]Department of Neurobiology and Behavior, Cornell University
{abn52,cbb89,jrg349,kellis,ws467}@cornell.edu

Abstract

Deep learning models struggle with systematic compositional generalization, a hallmark of human cognition. We propose MIRAGE, a neuro-inspired dual-process model that offers a processing account for this ability. It combines a fast, intuitive "System 1" (a meta-trained Transformer) with a deliberate, rule-based "System 2" (a Schema Engine), mirroring the brain's neocortical and hippocampal–prefrontal circuits. Trained to perform general, single-step decomposition on a stream of random grammars, MIRAGE achieves >99% accuracy on all splits of the SCAN benchmark in a task-agnostic setting. Ablations confirm that the model's systematic behavior emerges from the architectural interplay of its two systems, particularly its use of explicit, prioritized schemas and iterative refinement. In line with recent progress on *recursive/recurrent* Transformer approaches, MIRAGE preserves an iterative neural update while *externalizing* declarative control into an interpretable schema module. Our work provides a concrete computational model for interpreting how compositional reasoning can arise from a modular cognitive architecture.

1 Introduction

Humans effortlessly generalize by composing known concepts in new ways, a capacity termed systematic compositionality [1, 2]. Yet, even large-scale neural networks often fail at such generalization, tending to learn shallow heuristics instead of abstract rules [3–5]. This paper provides a processing account for compositional reasoning, inspired by the dual-process architecture of the human brain.

Cognitive neuroscience suggests that intelligent behavior arises from the interplay of complementary learning systems (CLS) [6, 7]. This theory posits that a fast, intuitive "System 1," supported by the neocortex, handles pattern recognition, while a slower, deliberate "System 2," involving the hippocampus (HPC) and prefrontal cortex (PFC), supports structured, episodic, and goal-directed reasoning [8, 9]. The HPC rapidly encodes specific experiences (episodes), which the PFC abstracts over time into reusable schemas, generalizable knowledge structures, that enable flexible planning and inference [10, 11]. This division of labor allows for both rapid adaptation and systematic application of knowledge.

We instantiate this theory in MIRAGE (Meta-Inference with Rules and Abstractions from Generalized Experience), a dual-process model with two interacting components:

(1) **System 1 (Neural Decomposer)**: A compact Transformer trained via meta-learning to perform a single step of compositional decomposition. This parallels the fast, pattern-matching function of the neocortex.

[†]Equal contribution.
[‡]Equal advising.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

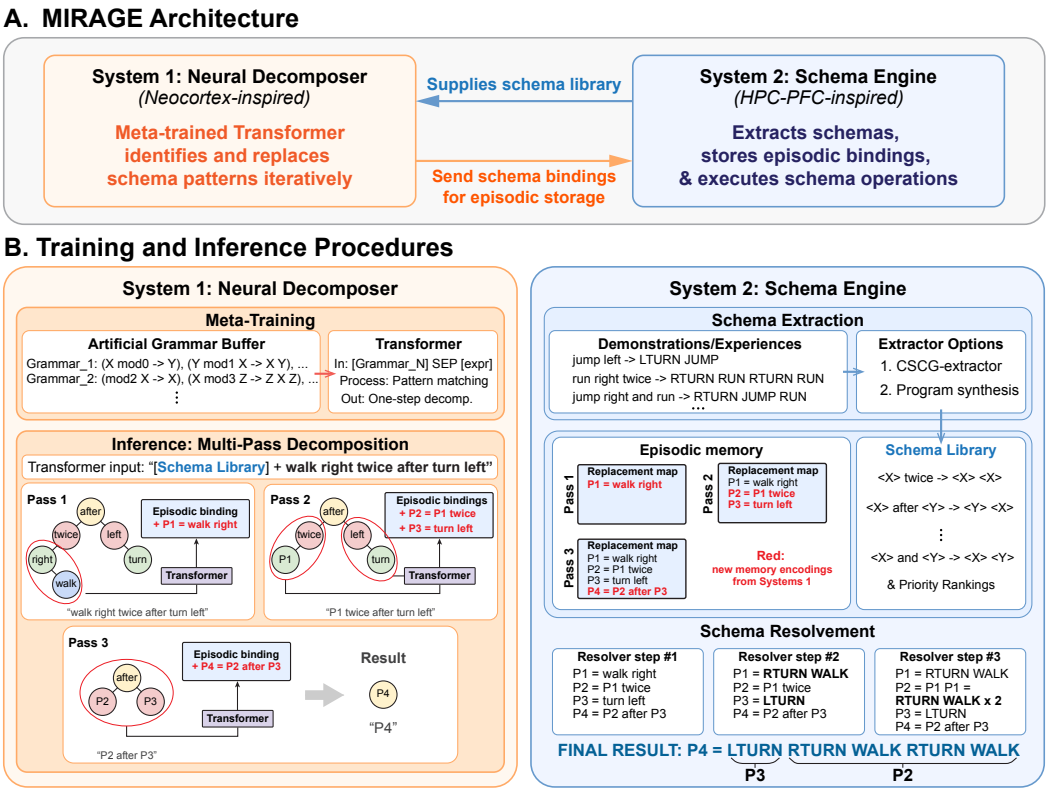

Figure 1: MIRAGE architecture and inference loop. (A) System 1 (Neural Decomposer) is a meta-trained Transformer for pattern matching. System 2 (Schema Engine) extracts and manages a library of prioritized schemas, modeling HPC–PFC function. (B) During inference, the systems iterate. For a command like "walk right twice after turn left," System 2 provides the relevant grammar. System 1 applies one decomposition step per pass, with outputs stored in an episodic memory. This process repeats, reducing the composition tree layer by layer until a primitive action sequence is produced.

(2) **System 2 (Schema Engine)**: An explicit symbolic module that extracts, stores, and applies prioritized schemas, managing variable bindings in an episodic memory. This mirrors the deliberate, schema-based reasoning of the HPC–PFC loop.

At inference, these systems engage in an iterative loop: System 2 extracts a compositional grammar from examples and uses it to guide the repeated application of System 1. Each pass of the System 1 Transformer refines the problem representation by one level, allowing the model to recursively decompose complex structures. This architecture provides an interpretable mechanism for generalization. On the SCAN benchmark [4], MIRAGE achieves state-of-the-art performance across all splits, despite its Transformer component never being trained on SCAN-specific data. Our work demonstrates that a modular, neuro-inspired architecture can provide a powerful and interpretable processing account for systematic generalization.

2 A Dual-Process Framework

MIRAGE solves compositional tasks through an iterative refinement process managed by its two systems. The core idea is to offload declarative knowledge (the "what") to an explicit schema library (System 2) and procedural knowledge (the "how") to a general-purpose neural reasoner (System 1).

System 2: Schema Engine. A compositional task is defined by a grammar $\mathcal{G} = (\Sigma, \pi, \psi)$, where Σ is a set of schemas (rewrite rules), π is a priority ordering, and ψ is an evaluator for primitive actions. System 2's role is to induce this grammar from a small set of examples. This extraction is modular; we show it can be done with either a symbolic rule-miner or a neuro-inspired model

based on Clone-Structured Causal Graphs (CSCGs) [12, 13]. For details on these methods, see Appendix B.2 and Appendix E.

System 1: Meta-Learned Transformer Decomposer. System 1 is a decoder-only Transformer $\mathcal{T}_\theta$ trained not to solve any single task, but to perform a generic, single step of compositional decomposition. We meta-learn $\mathcal{T}_\theta$ on an endless stream of randomly generated grammars. Each training sample consists of a grammar definition, an input sequence, and the target output after one decomposition step at the highest-priority location. This forces the model to learn the abstract *process* of rule application rather than memorizing solutions for a specific task. See Appendix A.3 and Appendix A.4 for a full description of the training procedure and model vocabulary.

Inference as a Processing Account. At inference time, System 2 first extracts the task grammar $\mathcal{G}$. Then, for a given input command, the model enters a loop (Figure 1B) that serves as a processing account of step-by-step deliberation. In each step, the current sequence and the grammar are fed to the Transformer $\mathcal{T}_\theta$. For example, given 'walk right twice after turn left' and a grammar where 'twice' has higher priority than 'after', the Transformer identifies 'walk right twice' and rewrites it to a placeholder token, say `P1`. The sequence becomes '`P1` after turn left'. In the next iteration, it decomposes this simpler structure. This process repeats until only a sequence of primitive actions remains. This iterative refinement allows a small model with a fixed context window to solve problems of arbitrary compositional depth. The full inference algorithm is detailed in Algorithm 1.

3 Experiments and Behavioral Analysis

We evaluate MIRAGE on the canonical **SCAN** benchmark [4], which tests systematic generalization to novel compositions and longer sequences. The System 1 Transformer (1.19M parameters) is trained task-agnostically on random grammars; for evaluation, the System 2 Schema Engine extracts the SCAN grammar from its training split, which is then provided to the frozen Transformer.

3.1 Compositional Generalization Performance

As shown in Table 1, **MIRAGE achieves near-perfect accuracy on all SCAN splits**, including those designed to foil standard sequence-to-sequence models. A vanilla Transformer trained directly on SCAN's simple split masters it but completely fails on the generalization splits, indicating it learns superficial correlations rather than abstract rules. Providing the schema library as a simple prompt ('Transformer+SC_Library') does not help, confirming that a specialized architecture and training objective are required to properly utilize schematic knowledge.

Table 1: SCAN accuracy (mean $\pm$ SEM over 4 runs). Baselines are re-trained on each split's training set. MIRAGE uses a single, task-agnostically trained Transformer for all splits.

		SCAN splits			
Model	**Full Task**	**Simple**	**Length**	**Add prim. 'jump'**	**Template**
Transformer	N/A	99.85 ± 0.00	13.58 ± 0.01	0.40 ± 0.13	3.09 ± 3.01
Transformer$_{\text{+SC_Library}}$	N/A	99.91 ± 0.11	15.86 ± 1.36	0.03 ± 0.04	0.00 ± 0.00
MIRAGE (ours)	99.59 ± 0.24	99.50 ± 0.30	99.35 ± 0.41	99.65 ± 0.20	99.55 ± 0.23

3.2 Processing Account: Ablation Studies

To understand which components drive this behavior, we conducted several ablations. These studies confirm that MIRAGE's performance is an emergent property of its cognitive architecture, not just one component.

1. **Schema priorities are crucial for cognitive control.** Removing the explicit priority ordering from the schema library causes accuracy to plummet from 99.6% to 71.9%. The model becomes unable to resolve ambiguous commands (e.g., 'turn left twice and walk'), analogous to a failure of executive function in selecting the correct order of operations.
2. **Iterative refinement enables deliberation.** Replacing the iterative loop with a single-pass, end-to-end mapping fails to generalize. The step-by-step process is essential, allowing the model to focus on one sub-problem at a time, a form of deliberate, sequential reasoning that monolithic models lack.
3. **Accurate schemas are necessary.** Introducing minor errors into the extracted grammar (e.g., misidentifying a schema's arguments) causes performance to collapse to nearly 0%. This highlights the model's reliance on accurate declarative knowledge. Its reasoning algorithm is sound, but it cannot function with a corrupted world model, a behavior consistent with human reasoning [14].

4 Related Work

Research on compositionality spans auxiliary objectives, neuro-symbolic models, meta-learning, and analyses of Transformer mechanisms. Auxiliary supervision can encourage structural understanding in Transformers [15], while memory-augmented models like LANE [16] explore variable-slot reasoning. Neuro-symbolic systems such as the Compositional Program Generator [17] and the Neural-Symbolic Recursive Machine [18] highlight the benefits of grammar-based modularity, paralleling MIRAGE's schema-based design.

Meta-learning approaches have also demonstrated strong potential for systematic generalization. In particular, Lake and Baroni [19] show that a meta-learned neural network can achieve human-like compositional generalization, supporting the view that inductive biases learned from experience can yield flexible reasoning skills.

Transformer-focused studies reveal both strengths and limits: while large models can sometimes achieve compositional generalization [20], their reasoning often emerges only through grokking [21] or limited mechanistic patterns [22], underscoring the need for specialized architectures.

Finally, neuroscience perspectives propose that hippocampal replay supports compositional computation [23], and that HPC–PFC circuits represent an evolutionary solution to flexible reasoning [24]. These ideas motivate MIRAGE's dual-process design, bridging cognitive neuroscience and AI.

5 Discussion and Conclusion

MIRAGE offers a hybrid approach to compositional generalization, using a neural network for a general procedural task (decomposition) while relying on an explicit, interpretable symbolic layer for declarative knowledge and control. Unlike monolithic models that must implicitly learn reasoning algorithms through "grokking," MIRAGE explicitly encodes a process of deliberate, iterative refinement. This dual-process framework offers a concrete processing account for systematic generalization. By instantiating cognitive theories of complementary learning systems, our model achieves state-of-the-art zero-shot performance on SCAN. We release the full implementation of MIRAGE on GITHUB. Our main contributions as an interpretive work are:

- **Processing Account:** We show that systematicity can emerge from an architecture that separates procedural skill from declarative knowledge and applies them iteratively.
- **Behavioral Account:** The model's success where standard Transformers fail demonstrates the behavioral advantage of a neuro-inspired, modular design.
- **Developmental Account:** Meta-learning provides a hypothesis for how agents can acquire robust, generalizable cognitive algorithms from varied experience.
- **Interpretability Account:** Because MIRAGE explicitly marks and resolves schemas step by step, it produces a transparent trace of intermediate problem states. This contrasts with black-box models that leap directly to an answer, making our framework inherently more interpretable and diagnostically useful.

Connection to Cognitive Learning Systems Beyond enabling stepwise compositional inference, schema formation as utilized by MIRAGE provides a mechanism for rapid assimilation of new rules into structured knowledge. In neuroscience, once an associative schema is established, new information consistent with that schema is integrated and becomes neocortically supported unusually quickly [10, 11, 25]. Recent work further links medial prefrontal schema representations with reinforcement-learning style updating and deployment for flexible behavior [9]. MIRAGE's prioritized schema selection echoes this PFC-mediated control over hippocampal content during retrieval and planning [11].

In addition, when viewed through a continual-learning lens, schemas could act as structured inductive priors: previously consolidated structure may be used to constrain and accelerate learning on new tasks. Bayesian treatments formalize this as using posteriors from past tasks to define priors for future ones, while also highlighting practical limits when approximations are poor [26, 27]. These observations motivate future work in the direction of probabilistic extension: inferring over candidate schema targets (and, when needed, over the library itself) to maintain sample-efficient generalization when direct schema identification is challenging.

Relation to recurrent/recursive Transformers A growing line of work explores *iterative* neural reasoning with weight-tying over steps, truncated credit assignment, and, in some cases, adaptive halting, yielding recurrent or recursive Transformer loops [28–31]. MIRAGE lies squarely within this trend by retaining an iterative per-step neural update, yet it differs in where the *control signal* lives: rather than learning control implicitly, we *externalize* it as explicit schemas with priorities. This keeps the empirical benefits of looped refinement while producing stepwise, editable traces and an algorithmic-level account of the reasoning process, which are central to Cognitive Interpretability. In this sense, MIRAGE complements recurrent and recursive Transformer practice with a cognitively motivated control layer, clarifying how and why each step is chosen.

Limitations and Future Work While successful on SCAN, the current framework relies on reliable extraction of a consistent grammar. Future work should explore robustness to noisy or incomplete schemas. Such issues could potentially be addressed by extending the extractor with probabilistic rule learning and/or the Neural Decomposer with active retrieval mechanisms. We also plan to apply this architecture to more complex reasoning tasks and use its inspectable states to test specific neuroscientific hypotheses about HPC–PFC interactions during planning. By building explicit, interpretable cognitive architectures, we can move beyond black-box evaluations and toward a mechanistic understanding of reasoning in neural systems.

References

[1] Jerry A Fodor. *The language of thought*, volume 5. Harvard University Press, 1975.

[2] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

[3] Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[4] Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR, 2018.

[5] Zhuoyan Xu, Zhenmei Shi, and Yingyu Liang. Do large language models have compositional ability? an investigation into limitations and scalability. *arXiv preprint arXiv:2407.15720*, 2024.

[6] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[7] Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[8] Morris Moscovitch, Roberto Cabeza, Gordon Winocur, and Lynn Nadel. Episodic memory and beyond: the hippocampus and neocortex in transformation. *Annual review of psychology*, 67(1):105–134, 2016.

[9] Oded Bein and Yael Niv. Schemas, reinforcement learning and the medial prefrontal cortex. *Nature Reviews Neuroscience*, pages 1–17, 2025.

[10] Dorothy Tse, Rosamund F Langston, Masaki Kakeyama, Ingrid Bethus, Patrick A Spooner, Emma R Wood, Menno P Witter, and Richard GM Morris. Schemas and memory consolidation. *Science*, 316(5821): 76–82, 2007.

[11] Alison R. Preston and Howard Eichenbaum. Interplay of hippocampus and prefrontal cortex in memory. *Current Biology*, 23(17):R764–R773, 2013. doi: 10.1016/j.cub.2013.05.041.

[12] Dileep George, Rajeev V Rikhye, Nishad Gothoskar, J Swaroop Guntupalli, Antoine Dedieu, and Miguel Lázaro-Gredilla. Clone-structured graph representations enable flexible learning and vicarious evaluation of cognitive maps. *Nat. Commun.*, 12(1):2392, April 2021.

[13] Weinan Sun, Johan Winnubst, Maanasa Natrajan, Chongxi Lai, Koichiro Kajikawa, Arco Bast, Michalis Michaelos, Rachel Gattoni, Carsen Stringer, Daniel Flickinger, et al. Learning produces an orthogonalized state machine in the hippocampus. *Nature*, pages 1–11, 2025.

[14] Michael H. Connors and Peter W. Halligan. A cognitive account of belief: a tentative road map. *Frontiers in Psychology*, Volume 5 - 2014, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2014.01588. URL `https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2014.01588`.

[15] Yichen Jiang and Mohit Bansal. Inducing transformer's compositional generalization ability via auxiliary sequence prediction tasks, 2021. URL `https://arxiv.org/abs/2109.15256`.

[16] Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. Compositional generalization by learning analytical expressions, 2020. URL `https://arxiv.org/abs/2006.10627`.

[17] Tim Klinger, Luke Liu, Soham Dan, Maxwell Crouse, Parikshit Ram, and Alexander Gray. Compositional program generation for few-shot systematic generalization, 2024. URL `https://arxiv.org/abs/2309.16467`.

[18] Qing Li, Yixin Zhu, Yitao Liang, Ying Nian Wu, Song-Chun Zhu, and Siyuan Huang. Neural-symbolic recursive machine for systematic generalization, 2024. URL `https://arxiv.org/abs/2210.01603`.

[19] Brenden M Lake and Marco Baroni. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623(7985):115–121, 2023.

[20] Jacob Russin, Sam Whitman McGrath, Danielle J. Williams, and Lotem Elber-Dorozko. From frege to chatgpt: Compositionality in language, cognition, and deep neural networks, 2024. URL `https://arxiv.org/abs/2405.15164`.

[21] Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024.

[22] Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task, 2024. URL https://arxiv.org/abs/2402.11917.

[23] Zeb Kurth-Nelson, Timothy Behrens, Greg Wayne, Kevin Miller, Lennart Luettgau, Ray Dolan, Yunzhe Liu, and Philipp Schwartenbeck. Replay and compositional computation, 2022. URL https://arxiv.org/abs/2209.07453.

[24] Elisabeth A Murray, Steven P Wise, and Kim S Graham. *the Evolution of Memory Systems: Ancestors, anatomy, and adaptations*. Oxford university press, 2017.

[25] Dharshan Kumaran, Demis Hassabis, and James L. McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(9): 512–534, 2016. doi: 10.1016/j.tics.2016.05.004. URL https://web.stanford.edu/~jlmcc/papers/KumaranHassabisMcClelland16FinalMS.pdf.

[26] Sebastian Farquhar and Yarin Gal. A unifying bayesian view of continual learning. *arXiv preprint arXiv:1902.06494*, 2019. URL https://arxiv.org/abs/1902.06494.

[27] Samuel Kessler, Adam Cobb, Tim G. J. Rudner, Stefan Zohren, and Stephen J. Roberts. On sequential bayesian inference for continual learning. *arXiv preprint arXiv:2301.01828*, 2023. URL https://arxiv.org/abs/2301.01828.

[28] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[29] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[30] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

[31] Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871. arXiv:2510.04871.

A Methodology details

A.1 Formal definition of Transformer Inference

Algorithm 1 Zero-Shot Inference on Single Input Sequence Example

Require: Trained transformer $\mathcal{T}_\theta$, grammar $\mathcal{G}$, input sequence $x^{(k)}$ with composition depth k
Ensure: Final flat output sequence $\mathcal{S}_{\text{final}}$ and replacement map λ

$\lambda \leftarrow \{\}$ ▷ empty map from schema-instances to primitives
$x^{(k)} \leftarrow$ input sequence
for $d = k, k-1, \ldots, 1$ **do**
  $y \leftarrow \mathcal{T}_\theta\big(x^{(d)}\big)$ ▷ predict next-level decomposition
  while there is a schema token s in y **do**
    let $(\sigma_s,\, a_1, \ldots, a_m)$ be the schema name and its m argument tokens in y
    $p \leftarrow \texttt{ApplySchema}(\sigma_s,\, a_1, \ldots, a_m)$ ▷ collapse into one primitive
    $\lambda\big[\sigma_s(a_1, \ldots, a_m)\big] \leftarrow p$
    replace the subsequence $\langle \sigma_s, a_1, \ldots, a_m \rangle$ in y with p
  $x^{(d-1)} \leftarrow y$ ▷ one less level of composition
return $\mathcal{S}_{\text{final}} \leftarrow x^{(0)},\ \lambda$

A.2 Formal definition of CSCG-based System 2 Algorithm

Algorithm 2 Extraction of compositional schemas

Require: Demonstrations $\mathcal{S}$, minimum support k

Train CHMM on $\mathcal{S}$; decode each $s \in \mathcal{S}$ to obtain episodes $\mathcal{E}$
$\mathcal{C} \leftarrow \varnothing$ ▷ candidate set
for all $(e_i, e_j) \in \binom{\mathcal{E}}{2}$ **do**
  $\tau \leftarrow \textsc{Align}(e_i, e_j)$ ▷ injective LCS with variables
  if $\tau \neq \perp$ **then**
    $\text{supp} \leftarrow \{e \in \mathcal{E} \mid \textsc{Validate}(\tau, e)\}$
    if $|\text{supp}| \geq k$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{\langle \tau, \text{supp} \rangle\}$
$\mathcal{S}_{\text{raw}} \leftarrow \textsc{Deduplicate}(\mathcal{C})$
$\mathcal{S}_{\text{final}} \leftarrow \textsc{PruneByComposition}(\mathcal{S}_{\text{raw}})$
return $\mathcal{S}_{\text{final}}$

A.3 Transformer Model Vocabulary Specifications for Meta-Learning

The vocabulary of our transformer $\mathcal{T}_\theta$ is defined with the following components. To use the model with any arbitrary grammar with its own specific vocabulary, like SCAN, tokens must be anonymized to match this format. For example, in SCAN, 'primitives' are tokens like 'walk', 'jump', 'look', etc., while modifiers are tokens 'and', 'after', 'opposite left', or 'twice'. Anonymizing a grammar in this way is a simple, deterministic process.

1. **Primitives**: `PRIM_0`, ..., `PRIM_P`. Primitives are the basic atomic elements in a vocabulary. Integer parameter P sets the number of allowed unique primitives.
2. **Modifiers**: `MOD_0`, ..., `MOD_M`. Schemas bind to and define the action of modifier tokens as functions. A given grammar $\mathcal{G}$ defines the action of a modifier in a sequence generated under it. Integer parameter M sets the number of allowed unique modifiers.
3. **Argument Tokens**: `ARG_0`, ..., `ARG_2*A`. Argument tokens are used in the in-context token sequences used to represent grammars. Specifically, argument tokens define the placeholders that a schema could bind to. For example, a schema may be represented presented via a token sequence as `ARG_0, ..., ARG_A MOD_0 ARG_A+1, ..., ARG_2A`, where the argument tokens are placeholders for real arguments to the schema and the modifier token represents the action the schema is binding to. Integer parameter A sets the maximum number of arguments a schema can have that occur before and after its principle modifier.

4. **Schema Name Tokens**: `SC_0, ..., SC_S`. These tokens define the names of specific schemas in the grammar or $|\Sigma|$. These names occur in-context to serve as markers for bindings in output sequences after a model application. Integer parameter S controls the number of schemas in a grammar, or $|\Sigma|$. We must have $S \geq m$ for a grammar G, as each modifier must occur in at least one schema to be well-defined.
5. **Priority Tokens**: `PRIORITY_0, ..., PRIORITY_S`. These tokens define the priority with which schemas are to be bound to. These tokens occur next to schema name tokens in-context to define their priority.
6. **Administrative Tokens**: `EOS`, `SEP`, `SC_DEF`, `SC_PRI`, `SC_SEP`, `LP_SEP` , `PAD`. These tokens are for formatting grammars in-context with input sequences generated by them.

A.4 Training Algorithms

During the transformer training process, new random grammars are generated at regular intervals. These grammars are added to a 'Grammar Buffer' maintaining the full set of previously generated grammars during the training process. At every step, the transformer samples grammars from this buffer and delegates the construction of random 2-deep composition input sequences and their corresponding output sequences. The concatenation of these components is then fed into the model. Multiple grammars are generally represented in each batch.

In the Methods section, we referenced a variety of small sub-algorithms that the transformer must delegate during the meta-learning training process. These include actually generating random grammars to add to the buffer at regular intervals (defined by a hyperparameter), generating input/output sequence pairs given a grammar, and actually sampling from the grammar buffer for each batch. Each of these algorithms are detailed here.

- **Random Grammar Generation:** For some subset of modifier tokens $M_G \subseteq M$, we generate a schema, σ. For each,
  - Choose $A_{before}, A_{after} \leq A_{max}$ for number of arguments before and after the modifier token in the schema output format.
  - Define $\psi(\sigma, (t_1, \ldots, t_{A_{before}+A_{after}}))$.
  - Define $\pi(\sigma)$.
- **Input Sequence Generation:** To generate an input sequence from a grammar, choose a schema, σ. For each argument in σ, select an atomic primitive or another schema σ_{sub} with $\pi(\sigma) < \pi(\sigma_{sub})$. For σ_{sub}'s, select all primitives as arguments. This generates 2-deep schema composition sequences.
- **Output Sequence Generation:** For the deepest layer of composition, simply replace the modifier token M of applied schema σ_M with a 'schema name token' like `SC_M` to represent σ_M. We train our model to output this format so that we can easily detect where the model has performed decomposition. This facilitates the zero-shot inference procedure, given a new grammar specification, outlined below.
- **Grammar Buffer Sampling:** A Grammar Buffer consists of a set of N previously used grammars. The buffer maintains the content of these grammars, as well as a corresponding C_i representing the number of times a grammar G_i has been used. When sampling from the buffer, we sample from the inverse C_i's for all buffers. Let C_i' be $\frac{1}{C_i+s}$, for smoothing factor $s, 0 < s \leq 1$. Then, the probability of selecting G_i is $\frac{C_i'}{\sum_{j=1}^{N} C_j'}$.

B Additional details about schema extractors

B.1 Priority Scoring Algorithm for CSCG-extractor

Algorithm 3 Learning operator precedence from one-step demonstrations

Require: fixed schema set $\mathcal{S}$, demonstration episodes $\mathcal{E}$ of the form $\langle \text{cmd} \rangle < \text{SEP} > \langle \text{1-step} \rangle$

$C \leftarrow \mathbf{0}$ ▷ pair-wise win counts
for all episode $e \in \mathcal{E}$ **do**
  $(\mathbf{x}, \mathbf{y}) \leftarrow$ split e at <SEP>
  $P \leftarrow \{\, s \in \mathcal{S} \mid \text{pattern}(s) \subset \mathbf{x} \,\}$ ▷ schemas present in the input command
  $F \leftarrow \{\, s \in P \mid \text{apply}(s, \mathbf{x}) = \mathbf{y} \,\}$ ▷ schemas that fired in the first step
  for all $s_w \in F$ **do**
    for all $s_\ell \in P \setminus F$ **do**
      $C[s_w, s_\ell] \leftarrow C[s_w, s_\ell] + 1$
for all $s_i \in \mathcal{S}$ **do**
  $\text{score}(s_i) \leftarrow \sum_{j \neq i} \big([\![C_{i,j} > 0]\!] - [\![C_{j,i} > 0]\!] \big)$ ▷ Copeland score
return $\{\text{score}(s_i)\}_{s_i \in \mathcal{S}}$

B.2 Enumerative Rule-Miner Details

This appendix expands on the *enumerative rule miner* introduced in Section 2.

B.2.1 Goal

From a support set $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$ of input–output strings, the miner induces a rewrite grammar $G = (\mathcal{P}, \mathcal{M}, \Sigma, \pi)$: primitives $\mathcal{P}$, modifiers $\mathcal{M}$, schemas $\Sigma = \{\sigma_1, \ldots, \sigma_S\}$, and a precedence map π. Each schema is a variable-binding template whose left-hand side (LHS) may contain string literals, span variables x_k (arbitrary substrings), or token variables u_k (single words).

B.2.2 Algorithm

(1) **Template generation** Enumerate candidate LHS–RHS templates in order of description length via three language-agnostic edit primitives: (i) span $\rightarrow$ token replacements, (ii) span wrappers inserting a control token, and (iii) span splicing / re-ordering.

(2) **Repair test** Insert a candidate template τ into the current grammar and re-evaluate all demonstrations. τ is accepted *iff* it rewrites every match consistently and increases corpus-level exact accuracy; accepted templates are appended to Σ.

(3) **Precedence induction** During replay, whenever two schemas match the same string and σ_i fires before σ_j, record $\sigma_i \succ \sigma_j$. A topological sort of this graph yields π.

(4) **Termination** Iterate steps (1–3) until the candidate queue empties, accuracy plateaus, or a fixed budget is reached.

Any alternative proposal mechanism (e.g., beam search, neural scorers, constraint solvers) can replace the enumerator in step (1) without changing the downstream interface: the Schema Engine consumes only $(\mathcal{P}, \mathcal{M}, \Sigma, \pi)$.

C CSCG application on SCAN

In line with showcasing the inability of pure Transformers to solve different SCAN splits on its own in Section 3, we explored directly applying CSCG on concatenated SCAN sequences. However, due to innate traits of the model architecture, SCAN, or other compositional tasks, quickly exhaust the structural clone bottlenecks. Specifically, the model is unable to distinguish between more than n sequences due to the Separator token, which is contained between every input and output. At the point of entering this token, the model must commit to one specific clone, leading to the loss of prior

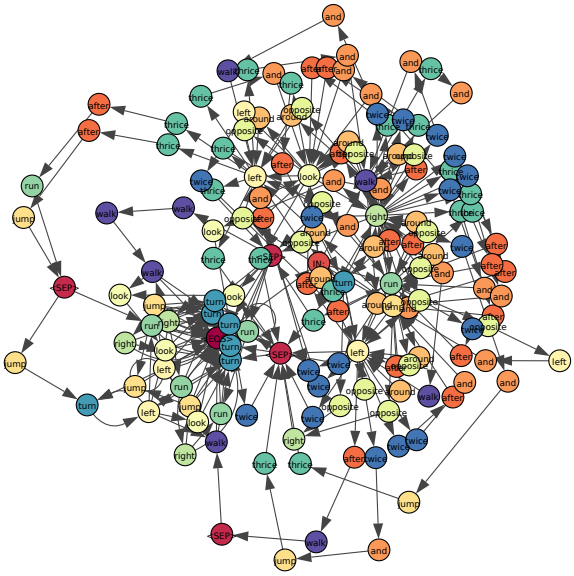

Figure 2: We apply a Clone-Structured Causal Graph with 100 clones directly on a concatenated subset of SCAN sequences and visualize the resulting model as a simple directed graph.

information. We further experienced with alternative variants that take the state across all clones of a single emission into account, increasing the number of potential configurations that the state can exhibit, but these modifications still failed to meaningfully solve any SCAN split.

D Compute Resources

Experiments were performed on readily available research hardware: single recent GPUs (e.g., A100 or H100) on a campus cluster and, occasionally, cloud services. Meta-training the Transformer required a few GPU-hours on one card, and each extra seed or ablation consumed a similar budget. Both schema extractors finish in under a minute on a CPU, and evaluating the full SCAN test set completes in under five minutes on a single GPU.

E CSCG Extractor Schemas

The CSCG extractor produces both, a textual representation of the extracted schemas, as well as a graph-based visualization, showcasing the direct correspondence between input and output variables. This visualization approach reveals potential for schema comparison through graph-based representations, as atomic schema demonstrations share consistent structural components. Additionally, the extractor correctly identifies special cases in the "turn" and "around" schemas, recognizing that these produce no additional output when paired with the turn primitive, unlike other primitives such as jump or run.

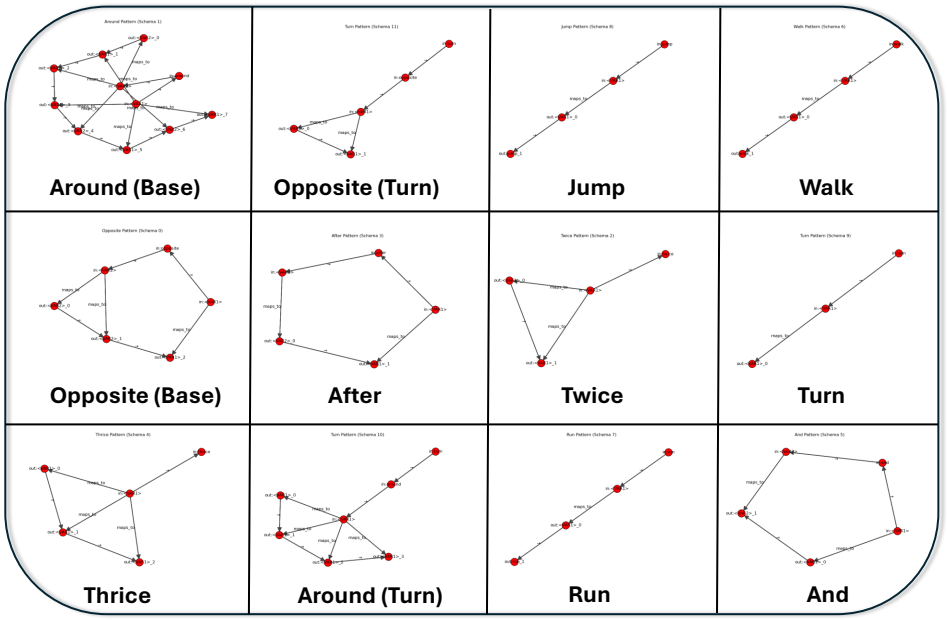

Figure 3: Overview of extracted schemas by the CSCG extractor, visualized as directed sequence graphs. Note that turn is a special case, which evokes different behavior when combined with around or opposite.