INPUT-ALIGNED GRID-STATE HANDOFFS ENABLE VISUAL OPERATIONS TO COMPOSE

Anonymous authors
Paper under double-blind review

ABSTRACT

In visual programs closed over grid states, learned operations can execute accurately in isolation yet fail when composed. We study which boundary representation converts available primitive transitions into execution when the program is supplied. Re-entering each intermediate result through the shared spatial color interface raises exact accuracy on unseen primitive pairs from 43.5% to 88.2% under matched training, with the advantage persisting on longer programs. With every learned weight fixed, the same handoff raises held-out-pair accuracy from 29.2% to 87.6%. Matched position-wise probes recover the intermediate grid from both executors, and grid-state re-entry makes that information directly consumable by the next operation. The advantage also transfers to grounded-object operations and programs recovered from demonstrations. For closed grid-to-grid visual programs, familiar operations compose when every intermediate result returns through the grid representation consumed by the next operation.

1 INTRODUCTION

A visual executor can implement every required operation yet fail to compose them in one call. We study closed grid-to-grid programs, where each primitive consumes and produces a visual grid. Given familiar rotation and translation operations and their supplied order, where should rotation's output live when translation begins? This handoff matters across modular visual reasoners, recurrent executors, and planner–executor systems. Compositional generalization asks whether familiar components transfer to unfamiliar combinations (Lake & Baroni, 2018; Keysers et al., 2020; Kim & Linzen, 2020; Mondorf et al., 2026). Existing approaches add structured representations, modules, programs, recurrence, or intermediate computation (Andreas et al., 2016b; Johnson et al., 2017b; Reed & de Freitas, 2016; Nye et al., 2021), often changing depth, instruction sequencing, and state visibility together. We isolate the resulting execution question: *when every intermediate task state is expressible in the visual input space, should it remain implicit in hidden activations or return through that shared interface?*

Supplied programs make this boundary directly testable. We compare parameter-matched executors with the same primitive sequence, shared Transformer calls, training data, and final-grid supervision. On held-out pairs, grid-state re-entry raises exact accuracy from 43.5% under latent handoff to 88.2%, a 44.7-point advantage present in all five matched seeds and sustained on held-out three-operation programs; a separate high-competence chain evaluation remains above 96% through six operations. The interface returns each transition to the normalized, spatially indexed visual space consumed by the next primitive. The advantage transfers from global grid transformations to grounded object operations. A separately trained whole-task executor already contains reusable primitive transitions: with every learned weight fixed, supplied rollout through predicted grids raises held-out-pair accuracy from 29.2% to 87.6%.

Three findings define the result. First, input-aligned handoff produces the 44.7-point matched composition gain. Second, latent states already make the intermediate grid linearly readable; reliable composition appears when that state re-enters the next operation's input representation. Third, the interface remains effective with frozen weights, across six-operation chains, for grounded-object operations, and with programs recovered from demonstrations.

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

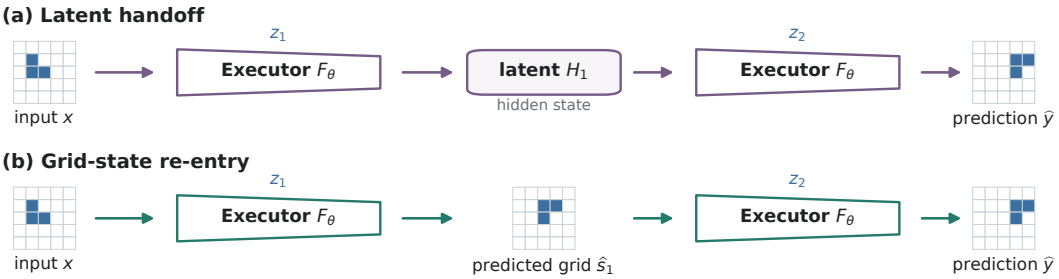

Figure 1: **The primary comparison isolates the state passed between primitives under matched computation and supervision.** The five-seed experiment separately trains parameter-matched latent-handoff and grid-state executors under the same supplied primitive sequence, data, update budget, and final-grid supervision. Within each executor, the same F_θ and weights are reused at both primitive calls. The latent executor passes H_1 directly; the grid-state executor predicts $\widehat{s}_1$ and re-enters it through the shared visual input interface.

2 Related Work

Compositional generalization. Compositional generalization tests whether familiar components remain executable in unfamiliar structures (Hupkes et al., 2020). Language benchmarks isolate novel instructions, structural combinations, and grounded action sequences (Lake & Baroni, 2018; Keysers et al., 2020; Kim & Linzen, 2020; Ruis et al., 2020). They also reveal a recurring separation between component-level competence and the ability to produce a composed answer (Press et al., 2023; Dziri et al., 2023). Our evaluation makes that separation measurable for visual execution.

Program-based visual reasoning. CLEVR and CLOSURE test systematic reasoning over visual programs and novel program structures (Johnson et al., 2017a; Bahdanau et al., 2019). ARC studies transformation discovery from demonstrations (Chollet, 2019; Lake & Baroni, 2023), while ConceptARC and compositional ARC extensions organize tasks around reusable concepts and their recombination (Moskvichev et al., 2023; Mondorf et al., 2026). Recent task-conditioned and visual–textual systems expand the learned interfaces used for ARC-style reasoning (Hu et al., 2026; Chen et al., 2026; Zhang et al., 2026). We supply an identifiable program, fixing recognition so the experiment tests how familiar visual operations exchange state during execution.

Modular neural execution. Neural module networks assemble learned visual operators into program-selected computation graphs (Andreas et al., 2016b;a; Hu et al., 2017; Johnson et al., 2017b; Mascharka et al., 2018), and conditional feature modulation steers a single visual network from an instruction embedding (Perez et al., 2018). Neural program interpreters and synthesis systems learn reusable subroutines or execute partial programs (Reed & de Freitas, 2016; Cai et al., 2017; Chang et al., 2019; Gondal et al., 2021; Yan et al., 2020; Ellis et al., 2021; 2019). Latent executors and tool-based visual programs extend this idea to learned execution traces and external modules (Chen et al., 2021; Shi et al., 2024; Gupta & Kembhavi, 2023; Surís et al., 2023). These approaches establish decomposition as a productive prior; we hold the supplied decomposition and processor calls fixed and isolate the representation passed across each primitive boundary.

Intermediate state interfaces. Intermediate generation, representation compatibility, and recurrent state all shape compositional reasoning. Explicit intermediate outputs improve unseen composition in synthetic autoregressive tasks, while CTL++ links failures on unseen compositions of known functions to incompatible learned representations (Ramesh et al., 2024; Csordás et al., 2022). Iterative decoding already returns predicted intermediate outputs to the model as its next input (Ruiz et al., 2021). Recurrent reasoning cells and latent language reasoners instead carry hidden state across steps: MAC passes control and memory between reasoning steps, Coconut feeds the final hidden state forward as the next input embedding, latent sentence reasoning compares continuous transfer with decode-and-reencode inference, and Soft Thinking re-enters probability-weighted mixtures of the input token embeddings (Hudson & Manning, 2018; Hao et al., 2025; Hwang et al., 2025; Zhang et al., 2025). In algorithmic execution, CLRS decodes intermediate hints and re-encodes them for the next recurrent step, and a systematic study of self-generated versus teacher-forced hints and of hard versus differentiable propagation establishes hint handling as a primary design axis (Veličković

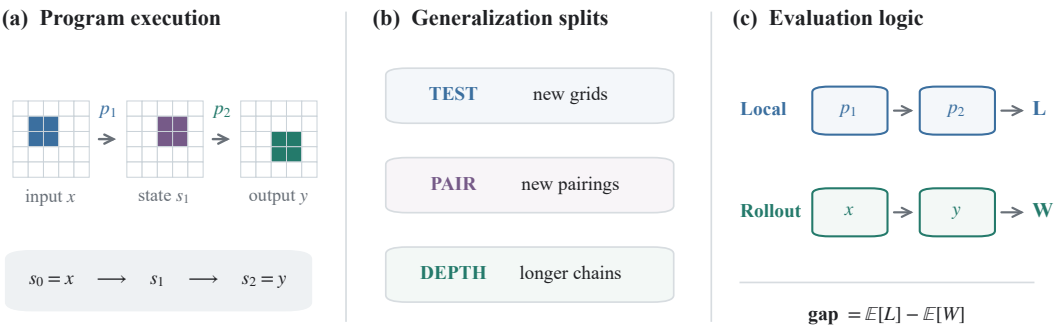

Figure 2: **Controlled visual programs expose execution and composition.** (a) Each specification defines a sequence of primitive-conditioned grid transitions. (b) Test, Pair, and Depth isolate generalization to new grids, held-out primitive pairings, and longer chains. (c) Reference-state diagnostics measure component availability, while end-to-end execution measures successful composition.

et al., 2022; Ibarz et al., 2022). Recurrent Transformers reuse parameters across depth, and textual and visual scratchpads externalize intermediate computation (Dehghani et al., 2019; Giannou et al., 2023; Nye et al., 2021; Lotfi et al., 2024; Hu et al., 2024; Li et al., 2025). Those comparisons vary supervision, decomposition, or modality alongside the interface. Our matched visual setting varies the interface alone: the primitive sequence, processor calls, parameters, data, update budget, and final-output supervision are all held fixed.

3 Method

Given a supplied program, we fix the primitive sequence, shared processor, data, and final-output supervision and vary only the state passed between primitive calls. Figures 1 and 2 summarize the interface comparison and controlled task design.

3.1 Problem Formulation and Closed Grid Programs

Task language. Let c be a task specification; K its number of operations; $t \in \{1, \dots, K\}$ a step index; p_t the operation at step t; $P(c) = (p_1, \dots, p_K)$ the ordered program; x and y the input and target grids; and s_t^* the reference grid after t operations. Then

$$s_0^* = x, \qquad s_t^* = p_t(s_{t-1}^*), \qquad y = s_K^*. \tag{1}$$

The family is closed over grid states: every p_t consumes and returns a grid on the same canvas, making the grid a task-sufficient state. Seven operation families cover rotation, reflection, translation, two forms of recoloring, masking, and cropping; nine categorical slots encode their parameters and canonical order in 38 dimensions.

Identifiable specifications. The language contains 10,920 legal specifications over grids of size 8–14. Each task provides four demonstrations and a query; generation retains sets explained by exactly one legal specification. Supplying it fixes program recovery and makes the state interface the experimental variable.

3.2 Execution Interfaces

We compare one-call whole-task execution with two primitive-aligned interfaces: latent handoff passes hidden pixel states, while grid-state re-entry returns each prediction through the shared visual input interface. We use *input-aligned handoff* for this return through the same representation consumed by the next operation.

Conditioning interface. Let $u(c)$ be the 38-dimensional encoding of c; E_ϕ the specification encoder; $z_c = E_\phi(u(c))$ the full-task token; $c^{(t)}$ the specification with every categorical slot outside p_t's operation family reset to its inactive value; and $z_t = E_\phi(u(c^{(t)}))$ its primitive token. Both occupy the same 512-dimensional space.

Shared visual interface. Let S be the canvas side length; $q \in \{0, \ldots, 10\}^{S \times S}$ a grid, with ARC colors 0–9 and padding 10; d the embedding width; $e_k \in \mathbb{R}^d$ the shared embedding of color class k; $r_{ij} \in \mathbb{R}^d$ the embedding of position (i, j); $o \in \mathbb{R}^{11 \times S \times S}$ the output logits; o_{kij} the logit for color k at (i, j); $\mathcal{I}$ the visual encoder; and $\mathcal{D}$ the discrete decoder. The shared input and output interfaces are

$$\mathcal{I}(q)_{ij} = e_{q_{ij}} + r_{ij}, \qquad \mathcal{D}(o)_{ij} = \arg \max_{k \in \{0,\ldots,10\}} o_{kij}. \tag{2}$$

For whole-task executor A with parameters θ_A, logit mapping F_{θ_A}, and output $\widehat{y}_A$, one call gives $\widehat{y}_A = \mathcal{D}(F_{\theta_A}(\mathcal{I}(x), z_c))$.

Latent handoff. Let T_θ be the shared Transformer block (Vaswani et al., 2017); H_t the hidden pixel state after step t; $H_0 = \mathcal{I}(x)$; W the shared color head; and $\widehat{y}_{\text{latent}}$ the final prediction. Without intermediate decoding or re-embedding, latent execution applies

$$H_t = T_\theta(H_{t-1}, z_t), \qquad \widehat{y}_{\text{latent}} = \mathcal{D}(W H_K). \tag{3}$$

Grid-state re-entry. Let F_θ be the shared primitive-conditioned executor; $\widehat{s}_t$ the predicted grid after step t; $\widehat{s}_0 = x$ the initial grid; o_t the step-t logits; and $\widehat{y}_{\text{grid}}$ the final prediction. Hard grid-state execution applies

$$\widehat{s}_0 = x, \qquad o_t = F_\theta(\mathcal{I}(\widehat{s}_{t-1}), z_t), \qquad \widehat{s}_t = \mathcal{D}(o_t), \qquad \widehat{y}_{\text{grid}} = \widehat{s}_K. \tag{4}$$

Differentiable re-entry. Let $\pi_{t,kij}$ be the step-t softmax probability of color k at position (i, j), and let $\widetilde{\mathcal{I}}(o_t)_{ij}$ be the corresponding expected visual embedding. The differentiable boundary is

$$\pi_{t,kij} = \text{softmax}(o_{t,\cdot ij})_k, \qquad \widetilde{\mathcal{I}}(o_t)_{ij} = \sum_{k=0}^{10} \pi_{t,kij} e_k + r_{ij}. \tag{5}$$

Training feeds $\widetilde{\mathcal{I}}(o_t)$ into the next primitive call; hard evaluation feeds $\mathcal{I}(\mathcal{D}(o_t))$. The same F_θ and weights are reused across all primitives and steps. The matched latent and grid-state executors share 12.9M parameters, task data, primitive sequence, processor calls, update budget, and final-output supervision, making the boundary state the comparison variable.

3.3 Learning from Final Outputs

Both primitive-aligned executors learn end to end from the final target grid, so the representation crossing each boundary defines their difference. Let $y \in \{0, \ldots, 10\}^{S \times S}$ be the target grid; y_{ij} its class at position (i, j); $o_{ij} \in \mathbb{R}^{11}$ the final logits there; $\text{softmax}(o_{ij})_{y_{ij}}$ the predicted probability of the target class; $\lambda(y_{ij})$ its weight (0.1 for padding, 1 for black, and 2 for colors 1–9); and $\ell_{\text{grid}}(o, y)$ the weighted grid loss. Then

$$\ell_{\text{grid}}(o, y) = \frac{\sum_{i,j} \lambda(y_{ij}) \left[-\log \text{softmax}(o_{ij})_{y_{ij}} \right]}{\sum_{i,j} \lambda(y_{ij})}, \tag{6}$$

We average this loss across the batch. Multi-operation tasks provide final targets, and single-operation tasks directly supervise primitive behavior. Neither primitive-aligned executor receives intermediate-grid targets; both backpropagate from the final grid through their respective state interfaces.

4 Experiments

4.1 Experimental Setup and Evaluation

Generalization splits. Training contains 1,510 tasks with one or two operations: 544 single-operation tasks establish primitive-conditioned transitions, and 966 two-operation tasks train familiar compositions. Test evaluates 458 familiar specifications on new grids, Pair holds out 204 operation co-occurrences, and Depth contains 180 three-operation programs whose constituent pairs occur in training. These splits separate input generalization from recombination and increased program length. Param-Color provides an auxiliary within-primitive recoloring diagnostic in Appendix A.

Table 1: **Grid-state re-entry supports reliable composition.** Exact-grid accuracy (%), mean ± sample standard deviation over five matched seeds. The interfaces share parameters, data, primitive-wise computation, update budget, and final-output supervision. Discrete scores evaluate the softmax-trained checkpoints with argmax feedback.

State passed between primitives	Seeds	Test	Pair	Depth
Latent pixel states	5	91.5 ± 2.2	43.5 ± 13.4	46.8 ± 15.0
Expected color-state re-entry (softmax)	5	94.7 ± 0.5	$\mathbf{88.2} \pm 3.6$	$\mathbf{90.9} \pm 1.3$
Discrete grid re-entry	5	94.6 ± 0.6	88.1 ± 3.4	91.0 ± 1.5

Training and statistical reporting. The main executor is a four-layer, width-512 visual Transformer with 12.9M parameters. Matched interface variants share the architecture, optimization schedule, training data, update budget, and validation-based checkpoint selection. Checkpoint selection uses held-in validation tasks disjoint from Pair and Depth. The primary comparison uses five matched training seeds and paired seed-level confidence intervals, while mechanism analyses use three independently trained seeds. Exact-grid accuracy requires every output cell to match. Full optimization and checkpoint details appear in Appendix A.

Component availability and composition. For task i, let x_i, y_i, and c_i be its input, target, and full specification; K_i its primitive count; $t \in \{1, \dots, K_i\}$ a step; p_{it} its step-t primitive; and s^*_{it} its reference grid after that step. Let $\widehat{F}_A(q, r)$ be executor A applied to grid q under instruction r; $\mathbf{1}[\cdot]$ the truth indicator; L_i^A the joint-local correctness indicator; and W_i^A the whole-task correctness indicator. Then

$$L_i^A = \prod_{t=1}^{K_i} \mathbf{1}\left[\widehat{F}_A(s^*_{i,t-1}, p_{it}) = s^*_{it}\right], \qquad W_i^A = \mathbf{1}\left[\widehat{F}_A(x_i, c_i) = y_i\right]. \tag{7}$$

L_i^A is one only when every primitive is correct for the same task and training seed. With deterministic decoding, $L_i^A = 1$ guarantees a correct rollout along the reference trajectory; its empirical mean therefore quantifies the primitive-conditioned transition competence available to rollout. Let $\mathbb{E}$ denote the empirical mean over task–seed trials; the composition gap is $\mathbb{E}[L_i^A] - \mathbb{E}[W_i^A]$. Let Pr denote empirical probability; $\Pr(W_i^A = 0 \mid L_i^A = 1)$ measures whole-task errors among trials with all components available.

4.2 Grid-State Re-entry Unlocks Held-Out Composition

The representation crossing the primitive boundary changes whether familiar operations compose. Matched parameters, primitive-wise computation, data, and supervision make the state passed between successive operations the experimental variable.

Grid-state re-entry improves Pair by 44.7 points (95% paired confidence interval, 26.3–63.1) and Depth by 44.1 points (24.2–64.0; Table 1). Every matched seed favors grid-state re-entry on both splits. Soft and hard feedback differ by at most 0.1 points, so the learned interface transfers directly to discrete grid execution.

Grid-state re-entry closes the composition gap that one-call execution leaves open (Table 2). Appendix C provides the full breakdown and last-state oracle control.

Table 2: **Grid-state re-entry closes the composition gap that one-call execution leaves open.** Exact-grid accuracy (%). "Local competence" is measured primitive-conditioned competence: per-step accuracy when each primitive is fed its reference input grid.

Execution	Pair	Depth
One-call execution (whole-task)	29.2	61.3
Grid-state re-entry, discrete	88.1	91.0
Primitive-conditioned local competence	87.4	91.7

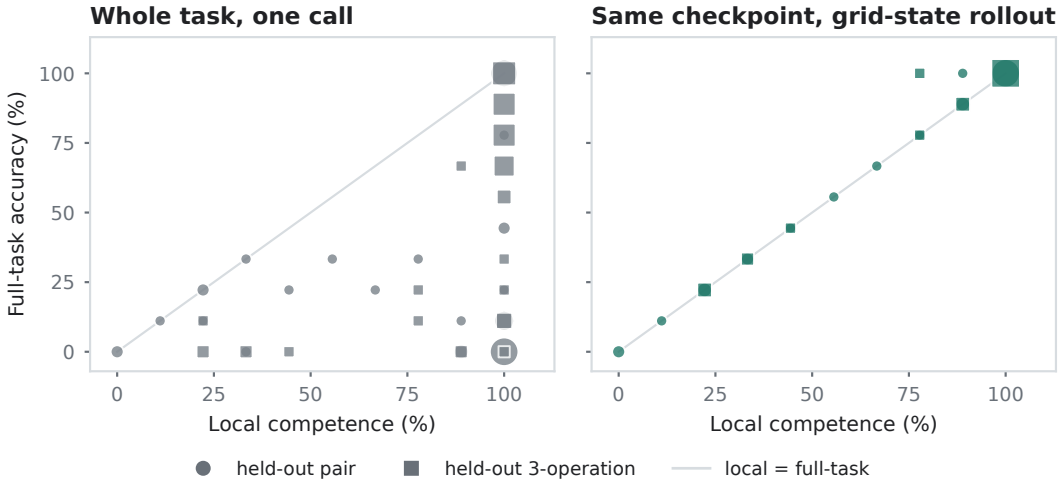

Figure 3: **Grid-state rollout converts local competence into composed accuracy.** Each marker aggregates unseen programs that share the same local-competence and full-task rates; marker area indicates how many programs coincide. Both panels use the identical three frozen whole-task checkpoints: left, one call on the whole specification; right, the same checkpoint invoked once per primitive through its own predicted grids. The diagonal marks equality between local and full-task accuracy.

4.3 Frozen Rollout Reuses Learned Primitive Transitions

The frozen intervention asks whether a whole-task-trained model already contains primitive-conditioned transitions that can be reused when the supplied program is externally unfolded at inference: with $\theta = \theta_A$ fixed, it replaces one full-program call with a sequence of primitive-conditioned calls under the discrete interface in Equation 4, each fed the previous predicted grid.

Table 3: **The same frozen executor recovers composition when unfolded over grid states.** Exact-grid accuracy (%), three-checkpoint mean $\pm$ sample standard deviation. Both rows use identical weights. The rollout receives the supplied primitive identities and order, and re-enters one predicted grid per primitive.

Execution interface	Test	Pair	Depth
Whole-task (one-pass)	93.2 ± 0.4	29.2 ± 2.0	61.3 ± 2.2
Same frozen model, supplied rollout	95.3 ± 0.3	87.6 ± 2.5	92.0 ± 1.2

With identical weights, grid-state rollout raises Pair accuracy by 58.4 points and Depth accuracy by 30.7 points (Table 3). This same-weight result shows that the learned primitive transitions support complete execution when invoked through the supplied rollout.

4.4 What Makes an Intermediate State Reusable

Primitive alignment and input-aligned state parameterization account for gains that supervised readability alone does not reproduce. The following three-seed controls distinguish these contributions; Table 1 reports the five-seed primary comparison.

Primitive alignment, not depth or recurrence alone, explains the gain. At $K = 3$, three shared four-layer calls and a twelve-layer one-pass executor each require approximately 15.3 GFLOPs; a generic recurrent scratchpad receiving the full specification and a step-position embedding at each of K calls, without per-call primitive identity, shares the same final-output supervision. Neither matched alternative recovers composition: the twelve-layer one-pass executor reaches 30.7%/54.4% on Pair/Depth, and the generic scratchpad collapses to 0.8–27.5%/0.0% under hard and soft feedback, while primitive-aligned grid-state re-entry reaches 88.2%/91.5% with one third as many unique parameters as the deeper one-pass executor (Table 4). Depth, recurrence, and final-output supervision are therefore insufficient; assigning the correct primitive at each state update is what recovers composition (Appendix H).

Table 4: **Depth, parameter count, and unaligned recurrence do not reproduce primitive alignment.** Exact-grid accuracy (%), mean ± sample standard deviation over three seeds. "Unique parameters" counts stored trainable weights, not parameters multiplied by calls. At $K = 3$, the twelve-layer one-pass executor and three calls to the shared four-layer grid-state executor each require approximately 15.3 GFLOPs. The generic scratchpad receives the full specification and a step-position embedding at each recurrent call, sharing final-output supervision but not primitive alignment.

Execution design	Unique parameters	Calls	Test	Pair	Depth
Whole-task (one-pass), 8 layers	25.5M	1	92.5 ± 2.0	27.8 ± 4.9	55.7 ± 16.9
Whole-task (one-pass), 12 layers	38.1M	1	91.3 ± 1.1	30.7 ± 2.5	54.4 ± 8.9
Generic scratchpad, soft feedback	12.9M	K	91.7 ± 2.0	27.5 ± 5.5	0.0 ± 0.0
Generic scratchpad, hard feedback	12.9M	K	29.4 ± 0.8	0.8 ± 1.4	0.0 ± 0.0
Primitive-aligned grid-state re-entry	12.9M	K	$\mathbf{94.9} \pm 0.3$	$\mathbf{88.2} \pm 4.8$	$\mathbf{91.5} \pm 1.4$

Intermediate supervision and re-entry provide distinct gains. Supervising intermediate readability recovers part of the Pair gain while retaining latent-to-latent handoff at inference. The control adds an auxiliary grid loss after each nonfinal latent segment.

Table 5: **Intermediate supervision and grid-state re-entry provide distinct gains.** Exact-grid accuracy (%) over three matched seeds; the auxiliary condition retains latent-to-latent handoff at inference.

	Pair		Depth	
Seed	Latent	+ Auxiliary grid loss	Latent	+ Auxiliary grid loss
0	23.5	53.9	25.6	46.1
1	44.6	47.5	50.0	56.7
2	45.1	59.3	55.0	51.7
Mean ± SD	37.7 ± 12.3	53.6 ± 5.9	43.5 ± 15.8	51.5 ± 5.3

The auxiliary loss improves Pair in every seed and raises its mean by 15.8 points. Grid-state re-entry adds another 34.6 Pair points and 39.4 Depth points, separating supervised readability from an input-aligned execution boundary.

Input-aligned color states. The boundary controls successively restore spatial positions, project hidden states through the color dictionary, and map spatial color weights onto the probability simplex. Normalized, input-aligned color states deliver 81.2–88.2% Pair accuracy, while intermediate reconstruction or restored spatial positions reach 53.6–56.0% (Table 6). The strongest boundary therefore combines spatial indexing with a bounded color-state representation in the coordinate system and vocabulary consumed by the next operation. Tying the boundary dictionary to the visual input stabilizes all five runs (Appendix A.3).

Table 6: **The composition gain grows as the boundary approaches reusable visual state.** Exact-grid accuracy (%), mean ± sample standard deviation over three matched seeds. Each row restores one more property of the shared visual interface, from supervised-only intermediates to full spatial and color re-entry. Full component controls appear in Appendix A.

Boundary state	Seeds	Pair	Depth
Latent handoff	3	37.7 ± 12.3	43.5 ± 15.8
Latent handoff, supervised intermediates	3	53.6 ± 5.9	51.5 ± 5.3
Latent handoff, spatial positions restored	3	56.0 ± 11.3	52.2 ± 15.9
Simplex-coded boundary, sigmoid–L1	3	81.2 ± 9.2	82.0 ± 9.7
Grid-state re-entry, softmax	3	$\mathbf{88.2} \pm 4.8$	$\mathbf{91.5} \pm 1.4$

Marginal distribution matching does not explain the gain. A further control asks whether input-aligned re-entry succeeds simply because it moves the boundary state's marginal statistics closer to those of ordinary visual input, independent of any structural commitment to the color simplex. Replacing the boundary state with an unconstrained per-position MLP and training it with a multi-scale

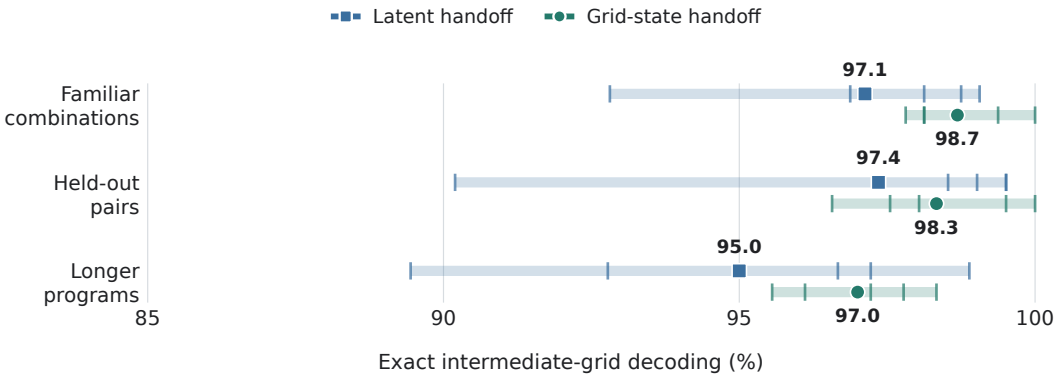

Figure 4: **Grid-state re-entry places readable intermediate information in the next operation's input representation.** Exact first-intermediate-grid decoding after one primitive-conditioned block. Markers show five-seed means, ticks show individual seeds, and pale segments show the observed range; the shared position-wise readout cannot move pixels.

maximum mean discrepancy (MMD) penalty (Gretton et al., 2012) to match the marginal distribution of real input-grid embeddings drives that mismatch to within $4\times$ of the null floor between two independent real samples—a roughly 400-fold reduction relative to unconstrained latent handoff—while Pair accuracy remains at 25.1%, far below grid-state re-entry's 88.2% (Appendix A.2). Matching marginal statistics alone therefore does not reproduce the structured, input-aligned state that composition requires.

Table 7: **Marginal distribution matching does not reproduce input-aligned re-entry.** Exact-grid accuracy (%) and the MMD^2 diagnostic, five seeds. "Floor" is the MMD^2 between two disjoint samples of the real embedding population itself.

Boundary condition	Test	Pair	Depth	Observed MMD^2	Floor MMD^2
Distribution-matched MLP	79.2 ± 8.7	25.1 ± 6.4	0.1 ± 0.2	0.0036 ± 0.0021	0.0009 ± 0.0002
Raw latent handoff (no penalty)	91.5 ± 2.2	43.5 ± 13.4	46.8 ± 15.0	1.44 ± 0.20	0.0009 ± 0.0002

Readable intermediate states require an aligned execution boundary. A matched probe compares the latent and grid-state executors after their first primitive-conditioned block (Figure 4). The same shared position-wise linear color readout is fitted to each frozen state; because it cannot move information between positions, exact recovery requires the transformed grid to be locally readable in canvas coordinates. Full fitting and evaluation counts appear in Appendix G.2.

Both interfaces encode the first transformed grid with high exact readability: $97.4 \pm 4.0\%$ versus $98.3 \pm 1.4\%$ on Pair and $95.0 \pm 3.8\%$ versus $97.0 \pm 1.2\%$ on Depth (Figure 4). Their 44.7-point execution difference separates intermediate-state readability from successful compositional reuse: grid-state re-entry returns the readable information through the representation family consumed by the next call. Complete family slices and commuting/noncommuting studies appear in Appendices C and G.

4.5 Scaling Across Program Length and Visual Domains

Composition through six operations. We evaluate repeated grid-state re-entry across 33 atomic operations that reach at least 95% pooled validation accuracy and programs of up to six operations. Each length contains 240 programs; the construction protocol and per-length results appear in Appendix B.

Across chains of high-competence primitives, frozen rollout tracks measured component competence through six operations, and matched grid-state executors retain 96.2–99.2% exact accuracy. The widening separation from one-call and latent execution shows that input-aligned re-entry sustains accurate execution across repeated primitive boundaries (Figure 5).

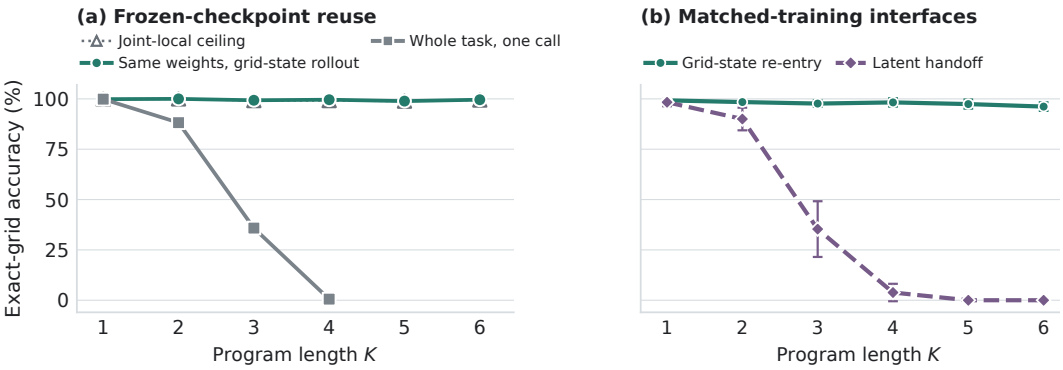

Figure 5: **Input-aligned handoff remains stable across six-operation chains of learned primitives.** Frozen rollout tracks joint local competence; matched grid-state executors maintain 96.2–99.2% exact accuracy across lengths one through six. Error bars show sample standard deviations.

Table 8: **The interface advantage transfers to grounded-object operations.** Exact-grid accuracy (%), mean $\pm$ sample standard deviation over three matched second-stage runs initialized from the primitive checkpoint.

Execution interface	Cross-step	S–A–T (2 calls)
Whole-task (one-pass)	18.7 ± 1.1	85.4 ± 0.7
Grid-state re-entry	$\mathbf{98.4} \pm 0.5$	$\mathbf{99.9} \pm 0.2$

Grounded object operations. The grounded-object domain composes selectors, object actions, rotations, and reflections. Its 18 primitives reach 98.2% standalone exact accuracy; Cross-step and the selector–action–transform (S–A–T) test unseen combinations. Appendix Table 26 specifies the architecture, splits, and representations.

Cross-step pairs 98.6% joint local competence with 18.7% one-call accuracy; grid-state re-entry reaches 98.4%. Frozen rollout of the same checkpoints reaches $98.6 \pm 0.5\%$ on Cross-step and $98.9 \pm 1.2\%$ on S–A–T; the shared primitive initialization reaches 97.6%/97.9% (Appendix Tables 22 and 27).

Recognized programs. With the same 246 recognized programs, changing only the execution backend raises matched-seed accuracy from 65.4% to 88.2%, converting 161 correct outputs to 217; three grid-state executors reach $87.0 \pm 2.8\%$. The fixed recognizer locates the gain at execution. Full recognition and backend protocols appear in Appendix D.

5 Conclusion

We introduced input-aligned grid-state handoff: every learned operation returns its predicted grid through the visual input representation consumed by the next operation. A compute-matched one-pass executor cannot recover this composition, establishing the cross-primitive state interface as a decisive factor beyond depth and capacity. The same handoff remains effective through six-operation chains, transfers from global transformations to grounded-object operations, and preserves more recognized programs when paired with an upstream recognizer. The resulting design principle is direct: when a task has a sufficient state representation, return each learned operation's result through that representation before invoking the next operation, turning available primitive transitions into compositional execution.

AI USE STATEMENT

AI tools were used to aid and polish writing, retrieve and discover relevant literature, support research ideation and experimental discussion, and draft and reorganize sections of the paper. They also assisted with figure preparation. All experimental results were produced from the authors' code and experimental records. The authors manually verified numerical values, citations, scientific claims, and AI-assisted text and figures against the underlying sources and artifacts. The authors take full responsibility for the final content of the paper, including all AI-assisted material.

REPRODUCIBILITY STATEMENT

The task language, model configurations, optimizer, training budget, checkpoint selection, evaluation units, seed-level results, and intervention protocols are specified in Section 3 and the appendices. Checkpoint selection uses held-in validation tasks disjoint from Pair and Depth. Supplied specifications fix program structure, while learned networks predict every reported grid transition. Code for the task language, executors, training, and evaluation is included in the supplementary material.

REFERENCES

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1545–1554. Association for Computational Linguistics, 2016a. doi: 10.18653/v1/N16-1181.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016b.

Dzmitry Bahdanau, Harm de Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. CLOSURE: Assessing systematic generalization of CLEVR models. *arXiv preprint arXiv:1912.05783*, 2019.

Jonathon Cai, Richard Shin, and Dawn Song. Making neural programming architectures generalize via recursion. In *International Conference on Learning Representations*, 2017.

Michael B. Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. Automatically composing representation transformations as a means for generalization. In *International Conference on Learning Representations*, 2019.

Xinwang Chen, Xiuxing Li, Qing Li, Ziyue Zhuang, Yutong Wu, Ziyu Li, Zhuo Wang, Kai Li, Jianye Hao, and Xia Wu. Human-like abstract visual reasoning via understanding and solving reasoning loop. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41235–41244, 2026.

Xinyun Chen, Dawn Song, and Yuandong Tian. Latent execution for neural program synthesis beyond domain-specific languages. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. CTL++: Evaluating generalization on never-seen compositional patterns of known functions, and compatibility of neural representations. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9758–9767. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.662.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. volume 36, pp. 70293–70332, 2023.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Joshua B. Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a REPL. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 835–850. Association for Computing Machinery, 2021. doi: 10.1145/3453483.3454080.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023.

Muhammad Waleed Gondal, Nasim Rahaman, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic inference with neural interpreters. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling*, 2025.

Keya Hu, Ali Cy, Linlu Qiu, Xiaoman Delores Ding, Runqian Wang, Yeyin Eva Zhu, Jacob Andreas, and Kaiming He. Arc is a vision problem! In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2537–2546, 2026.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 804–813, 2017.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. volume 37, pp. 139348–139379, 2024.

Drew A. Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations*, 2018.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020. doi: 10.1613/jair.1.11674.

Hyeonbin Hwang, Byeongguk Jeon, Seungone Kim, Jiyeon Kim, Hoyeon Chang, Sohee Yang, Seungpil Won, Dohaeng Lee, Youbin Ahn, and Minjoon Seo. Latent reasoning via sentence embedding prediction. *arXiv preprint arXiv:2505.22202*, 2025. Previously titled "Let's Predict Sentence by Sentence"; presented at the COLM RAM 2 Workshop (Oral), which is not a proceedings venue.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner. In *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pp. 2:1–2:23. PMLR, 2022.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, 2017a.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision*, pp. 2989–2998, 2017b.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*, 2020.

Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 9087–9105. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.731.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2873–2882. PMLR, 2018.

Brenden M. Lake and Marco Baroni. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623:115–121, 2023. doi: 10.1038/s41586-023-06668-3.

Chengzu Li, Wenshan Wu, Huanyu Zhang, Yan Xia, Shaoguang Mao, Li Dong, Ivan Vulić, and Furu Wei. Imagine while reasoning in space: Multimodal visualization-of-thought. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36340–36364. PMLR, 2025.

Aryo Lotfi, Enrico Fini, Samy Bengio, Moin Nabi, and Emmanuel Abbe. Chain-of-sketch: Enabling global visual reasoning. *arXiv preprint arXiv:2410.08165*, 2024. Previously titled "Visual Scratchpads: Enabling Global Reasoning in Vision" (v1, 2024-10-10); retitled "Chain-of-Sketch: Enabling Global Visual Reasoning" in v2 (2025-06-26).

David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4942–4950, 2018.

Philipp Mondorf, Shijia Zhou, Monica Riedler, and Barbara Plank. Compositional-ARC: Assessing systematic generalization in abstract spatial reasoning. In *International Conference on Learning Representations*, 2026.

Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The ConceptARC benchmark: Evaluating understanding and generalization in the ARC domain. *Transactions on Machine Learning Research*, 2023.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11671.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Rahul Ramesh, Ekdeep Singh Lubana, Mikail Khona, Robert P. Dick, and Hidenori Tanaka. Compositional capabilities of autoregressive transformers: A study on synthetic, interpretable tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42074–42103. PMLR, 2024.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, 2016.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Luana Ruiz, Joshua Ainslie, and Santiago Ontañón. Iterative decoding for compositional generalization in transformers. *arXiv preprint arXiv:2110.04169*, 2021.

Kensen Shi, Joey Hong, Yinlin Deng, Pengcheng Yin, Manzil Zaheer, and Charles Sutton. Exedec: Execution decomposition for compositional generalization in neural program synthesis. In *International Conference on Learning Representations*, 2024.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11888–11898, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 22084–22102. PMLR, 2022.

Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Hashemi. Neural execution engines: Learning to execute subroutines. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Beichen Zhang, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Think visually, reason textually: Vision-language synergy in abstract reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41203–41212, 2026.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of LLMs in continuous concept space. In *Advances in Neural Information Processing Systems*, pp. 168990–169012, 2025.

Appendix

A Execution and Supervision Controls

Evidence units. The primary trained state-interface endpoint uses five matched training seeds. Frozen-checkpoint analyses use three independently trained whole-task checkpoints, and auxiliary ablations use three training seeds unless labeled otherwise. Grounded-object second-stage results report means and sample standard deviations over three independently optimized runs initialized from one shared primitive checkpoint. Rollout from the pre-second-stage primitive initialization uses that single checkpoint and is reported without a standard deviation. External-recognition comparisons reuse one cached generation set across executors. We refer to the original global-grid DSL and the grounded-object DSL by their domain names rather than experiment-version labels.

Table 9 reports the five-seed grid-state endpoint together with three-seed intermediate-supervision and transition-augmentation controls; the seed count is explicit for every row. All procedures use the supplied task specification. Multi-operation final-target training and teacher-forced intermediate-target training are distinct protocols.

Param-Color is retained as an auxiliary within-primitive recoloring diagnostic. All five primary grid-state seeds solve all 48 Param-Color tasks, giving 48/48 for every seed and 240/240 task–seed trials.

Table 9: Supervision controls, mean and sample standard deviation (%). The seed count is reported per row. The grid-state executor learns from final targets on composed tasks; the teacher-forced variant uses intermediate targets; the transition-augmented whole-task executor adds standalone transition examples.

Condition	Seeds	Test	Param-Color	Pair	Depth
Grid-state, final targets (hard)	5	94.6±0.6	100.0±0.0	88.1±3.4	91.0±1.5
Teacher-forced intermediate targets	3	94.2±0.8	97.2±2.4	81.9±2.9	88.1±2.3
Whole-task + transition examples	3	91.0±2.8	99.3±1.2	32.5±5.6	55.6±9.1

The separate seed-0 training-budget control extends whole-task training from 15,000 to 24,600 updates. Both select the same update-13,500 checkpoint and retain 93.0/100.0/31.4/63.3% on Test/Param-Color/Pair/Depth. Under matched whole-task transition supervision, grid-state re-entry provides the strongest composition endpoint. Intermediate supervision reaches $81.9 \pm 2.9\%$ Pair and $88.1 \pm 2.3\%$ Depth, and additional transition examples preserve the one-call execution profile at $32.5 \pm 5.6\%$ and $55.6 \pm 9.1\%$.

A.1 Primitive-aligned latent depth

The latent-segmented executor is the matched state-interface control from Table 1. It uses the same four-layer, width-512 shared Transformer block as the grid-state executor and applies that block once per supplied primitive. Pixel hidden states pass directly from segment t to segment $t+1$; no intermediate color logits are decoded or re-embedded. Training uses the same 1,510 tasks, exact primitive sequence, 15,000 updates, batch size 64, final-grid loss, and validation-loss checkpoint selection as the grid-state executor. Both contain 12,911,627 parameters and receive no intermediate-grid targets.

Table 10: Primitive-aligned latent execution across five seeds, exact-grid accuracy (%). Hidden pixel states cross primitive boundaries without intermediate decoding.

Seed	Best update	Test	Param-Color	Pair	Depth
0	11,250	88.2	68.8	23.5	25.6
1	15,000	93.9	100.0	44.6	50.0
2	13,000	93.0	100.0	45.1	55.0
3	13,000	91.9	100.0	61.3	64.4
4	9,500	90.4	75.0	43.1	38.9
Mean ± SD	–	91.5 ± 2.2	88.8 ± 15.6	43.5 ± 13.4	46.8 ± 15.0

Table 11: Boundary-component controls, exact-grid accuracy (%). The reset condition preserves hidden states while re-adding spatial positions and applying LayerNorm. Linear re-entry projects to color logits and back through learned color embeddings without softmax or argmax.

Boundary interface	Seed	Best update	Test	Pair	Depth
Positions + LayerNorm	0	10,250	93.2	60.8	61.1
	1	9,000	92.4	64.2	61.7
	2	10,250	88.6	43.1	33.9
	Mean $\pm$ SD	–	91.4 ± 2.4	56.0 ± 11.3	52.2 ± 15.9
Linear logit re-entry	0	2,500	76.4	12.3	0.0
	1	2,250	65.1	16.2	0.0
	2	2,750	72.9	15.7	0.0
	Mean $\pm$ SD	–	71.5 ± 5.8	14.7 ± 2.1	0.0 ± 0.0

Re-adding positions and LayerNorm raises mean Pair accuracy from 37.7% to 56.0%. Normalized color-state re-entry then raises it to 81–88%, associating spatial anchoring with a modest gain and normalized color representation with the largest additional gain.

Table 12: **Normalized color-state re-entry supports composition across parameterizations.** Exact-grid accuracy (%), mean $\pm$ sample standard deviation over three matched seeds. Linear + LayerNorm controls the scale of an unconstrained logit projection. Sigmoid–L1 and softmax instead map color weights to the probability simplex before re-embedding them with the shared color dictionary. The $T = 1$ row reports soft feedback from the first three primary grid-state seeds.

Boundary state	Seeds	Test	Param-Color	Pair	Depth
Latent handoff	3	91.7 ± 3.1	89.6 ± 18.0	37.7 ± 12.3	43.5 ± 15.8
Linear re-entry + LayerNorm	3	90.5 ± 4.2	89.6 ± 16.3	41.3 ± 18.4	39.4 ± 21.9
Sigmoid–L1 convex re-entry	3	92.4 ± 4.4	98.6 ± 1.2	81.2 ± 9.2	82.0 ± 9.7
Softmax re-entry, $T = 2$	3	94.1 ± 1.9	96.5 ± 6.0	82.5 ± 4.6	86.9 ± 3.6
Softmax re-entry, $T = 1$	3	94.9 ± 0.3	100.0 ± 0.0	88.2 ± 4.8	91.5 ± 1.4

Convex color-state boundaries provide the largest gain in this comparison: sigmoid–L1 and both softmax parameterizations exceed 80% on Pair and Depth, whereas the scale-normalized linear projection reaches 41.3%/39.4%. Across the successful interfaces, each spatial location is represented by normalized convex weights over the input color dictionary and re-entered through the visual input interface.

A.2 Distribution-matched latent control

This control asks whether the composition gain in Table 6 reduces to resembling the marginal distribution of ordinary visual input, independent of any structural commitment to the color simplex. At each non-final primitive boundary, an otherwise unconstrained two-layer MLP (`Linear`→`GELU`→`Linear`, applied position-wise) replaces the hidden pixel state; nothing in the architecture restricts its output to the color dictionary's span or to a probability simplex. Training adds a multi-scale RBF maximum mean discrepancy (MMD) penalty (Gretton et al., 2012) between the transformed boundary state and a freshly sampled population of real input-grid embeddings $\mathcal{I}(x)$, drawn from ordinary task inputs at every update—never the current example's true intermediate grid, so the penalty carries only population-level, not per-example, information. The kernel bandwidth (a five-scale multiple of the median heuristic) is recomputed every step from the current batch; boundary and reference vectors are each subsampled to 512 points per update for the kernel evaluation. The boundary MLP adds 525,312 parameters to the 12,911,627-parameter grid-state architecture (+4.1%).

We select the MMD weight ($\lambda = 20$) from a short calibration sweep ($\lambda \in \{5, 20, 60\}$, reduced step budget) trading off matching strength against task convergence, then train five full seeds at 15,000 updates with the same data, batch size, and validation-loss checkpoint selection as every other boundary control.

The MMD objective brings mean observed MMD^2 from 1.44 ± 0.20 to 0.0036 ± 0.0021, within a factor of four of the matched null floor and roughly 400 times closer to the real embedding popu-

Table 13: **Marginal distribution matching does not reproduce input-aligned re-entry.** Exact-grid accuracy (%) and the MMD^2 diagnostic, five seeds. "Observed" is the MMD^2 between the trained boundary state and real input-grid embeddings on held-out two-primitive tasks; "Floor" is the same statistic between two disjoint samples of the real embedding population itself. The raw latent-handoff row applies the identical diagnostic to the Table 10 checkpoints, with no matching penalty.

Seed	Test	Pair	Depth	Observed MMD^2	Floor MMD^2
0	87.1	31.4	0.0	0.0027	0.0010
1	83.0	25.5	0.0	0.0024	0.0007
2	64.4	15.7	0.0	0.0074	0.0008
3	80.1	22.5	0.0	0.0024	0.0011
4	81.4	30.4	0.6	0.0029	0.0008
Mean $\pm$ SD	79.2 ± 8.7	25.1 ± 6.4	0.1 ± 0.2	0.0036 ± 0.0021	0.0009 ± 0.0002
Raw latent handoff (no penalty)	see Table 10			1.44 ± 0.20	0.0009 ± 0.0002

lation, while Pair/Depth remain at 25.1%/0.1%. Matching marginal embedding statistics therefore does not supply the structured state interface recovered by grid-state re-entry.

A.3 Untied boundary codebooks

This control keeps spatial indexing and softmax normalization and changes only the dictionary that the boundary weights index. Instead of the shared color embedding consumed at the visual input interface, a separate head predicts weights over a freely learned codebook, which is then re-embedded and re-positioned exactly as in grid-state re-entry. The 11-entry variant matches the grid-state codebook size and adds 11,275 parameters to 12,911,627 (0.09%); the 64-entry variant adds 65,600 (0.51%). Training data, side length, width, depth, batch size, 15,000 updates, and validation-loss checkpoint selection are unchanged.

Table 14: **Tying the boundary dictionary to the input vocabulary stabilizes training.** Exact-grid accuracy (%) per seed. The 11-entry variant is extended to five seeds because it substitutes directly for the primary grid-state interface; the 64-entry variant uses three seeds and tests codebook capacity. Grid-state re-entry reaches $94.7 \pm 0.5\%$ on Test across five seeds with every run converging.

Codebook	Seed	Best update	Best val. loss	Test	Pair	Depth
Untied, 11 entries	0	13,000	0.0283	91.9	78.4	60.0
	1	2,000	0.4115	52.4	25.0	0.0
	2	6,250	0.0243	92.1	76.0	71.7
	3	3,250	0.1863	58.5	33.8	8.9
	4	5,250	0.1707	64.4	40.7	6.7
	Mean $\pm$ SD	–	–	71.9 ± 18.9	50.8 ± 24.8	29.4 ± 33.6
Untied, 64 entries	0	7,000	0.0254	90.6	76.5	28.3
	1	4,000	0.2574	56.8	27.9	6.1
	2	13,000	0.0906	87.8	64.7	50.6
	Mean $\pm$ SD	–	–	78.4 ± 18.8	56.4 ± 25.3	28.3 ± 22.2

All five tied runs converge, reaching $94.7 \pm 0.5\%$ familiar-task accuracy and the strongest composition endpoint. The shared input dictionary therefore provides a substantially more stable training endpoint. Untied dictionaries produce a wider seed-level distribution, with the two converged 11-entry runs reaching 76.0–78.4% Pair. The tied interface places the boundary state in the vocabulary consumed by the visual input and couples its gradients to that embedding.

A three-seed capacity control increases the one-pass whole-task executor from four layers to eight and twelve layers. The eight-layer model (25.5M parameters) reaches $92.5 \pm 2.0/100.0 \pm 0.0/27.8 \pm 4.9/55.7 \pm 16.9\%$ on Test/Param-Color/Pair/Depth; the twelve-layer model (38.1M parameters, approximately 15.3 GFLOPs) reaches $91.3 \pm 1.1/100.0 \pm 0.0/30.7 \pm 2.5/54.4 \pm 8.9\%$. In this three-seed matched-compute control, primitive-aligned re-entry reaches 88.2%/91.5% on Pair/Depth, establishing the execution boundary as a decisive factor beyond depth and compute.

Training uses 1,510 tasks, including 544 single-operation and 966 two-operation tasks. Final-output targets on the single-operation subset directly teach the primitive-conditioned behavior reused by frozen-checkpoint rollout. At test time, the intervention replaces the whole-program interface with primitive-specific conditioning coupled to successive predicted states. On multi-operation tasks, the separately trained grid-state executor learns through the final-grid objective and passes its own predicted states between invocations. Intermediate targets enter the teacher-forced, extra-transition, and supervised-latent controls, but not the primary grid-state executor's multi-operation objective.

The operation-specific control replaces the shared executor with seven complete 12.9M-parameter networks, one for each operation family. At every supplied step, the external loop selects the corresponding family network, provides the current predicted grid and primitive specification, and passes its output to the next selected network using the same feedback rule as grid-state re-entry. The seven networks contain 90.4M parameters in total but divide the available training transitions by family: 232 nonblack-recolor, 804 exact-recolor, 340 reflection, 632 translation, 76 crop, 173 mask, and 219 rotation transitions. With hard feedback, the collection reaches 85.2/69.4/64.1/74.1% on Test/Param-Color/Pair/Depth across three seeds. The shared executor converts cross-family data sharing into higher composition accuracy despite using one seventh of the aggregate parameters.

B High-Competence Chain-Length Scaling

The scaling suite is evaluation-only and reuses the three frozen whole-task checkpoints. Primitive eligibility is fixed using the original validation split: an atomic vector must reach at least 95% pooled exact-grid accuracy across the checkpoints. All 33 non-crop, training-supported atomic vectors satisfy this criterion. New evaluation tasks do not participate in selection. For each $K \in \{1, \ldots, 6\}$, the suite contains 240 unique inputs generated with seed 20260921.

For $K = 1$–4, we sample valid no-crop composite vectors whose constituent atomic vectors pass the validation criterion. These programs have both the original whole-program token and its supplied primitive decomposition. The original language has one slot per operation family and mutually excludes rotation with reflection and exact with nonzero recoloring; consequently, four is the maximum no-crop length expressible by one whole-program token. For $K = 5$–6, we sample explicit sequences from the same atomic pool, disallow immediate duplicate requests, and retain programs in which every reference transition changes the grid, every intermediate state is nonempty, and the final program is non-identity. These longer rows test the stability of the primitive interface and have no whole-token score.

Figure 5 in the main text plots this suite. Table 15 gives the frozen-checkpoint values; Table 16 evaluates the five matched latent/grid-state checkpoint pairs on the identical tasks.

Table 15: Exact-grid accuracy (%, mean $\pm$ sample standard deviation across the three frozen whole-task checkpoints) on 240 programs per length. Joint local requires every primitive to be correct on its reference input state. Lengths 1–4 are representable by one whole-program token; lengths 5–6 exceed what the task language can express in a single specification and have no one-call score.

Length K	Joint local	Whole task, one call	Same weights, grid-state rollout
1	99.9 ± 0.2	99.9 ± 0.2	99.9 ± 0.2
2	100.0 ± 0.0	88.2 ± 1.3	100.0 ± 0.0
3	99.2 ± 0.8	35.8 ± 1.8	99.3 ± 0.6
4	99.2 ± 1.1	0.6 ± 0.5	99.6 ± 0.7
5	98.9 ± 0.9	–	98.9 ± 0.9
6	99.4 ± 0.5	–	99.6 ± 0.4

C Component Competence and Counterfactual Programs

C.1 Joint component evaluation

The local diagnostic in Equation 7 evaluates each primitive on the exact state produced by the reference prefix. The conjunction is taken within each task and seed. Whole-task accuracy is measured

Table 16: Matched-training chain-length scaling on the same 240 programs per length (%, mean ± sample standard deviation across five paired seeds). Grid-state checkpoints use hard predicted-grid feedback; latent checkpoints pass hidden pixel states directly between shared-block applications. All checkpoints are selected on the original validation split before this evaluation.

Length K	Grid-state re-entry	Latent handoff
1	99.2 ± 0.7	98.3 ± 2.0
2	98.4 ± 0.7	90.0 ± 5.6
3	97.7 ± 1.7	35.3 ± 13.8
4	98.2 ± 2.1	3.8 ± 4.3
5	97.4 ± 2.3	0.0 ± 0.0
6	96.2 ± 2.1	0.0 ± 0.0

separately using the combined specification. This pairing identifies compositions for which constituent predictions are available even when the full-specification prediction fails. The corresponding local-competence-versus-composed-accuracy scatter appears as Figure 3 in the main text.

Table 17: Measured component availability and composed accuracy (%). Whole-task columns pool three checkpoints; grid-state columns pool all five grid-state checkpoints. "Last-state oracle" replaces the input to the final primitive with the reference intermediate grid. "Two-step familiar" is the two-operation subset of Test, not the full Test split. The gap is a difference between measured task-level rates.

Evaluation subset	Local	Whole task	Gap	Grid state	Last-state oracle
Two-step familiar compositions	94.5	91.5	3.0	93.5	94.9
Two-step held-out pairs	87.4	29.2	58.2	88.1	89.2
Three-step held-out compositions	91.7	61.3	30.4	91.0	94.0

For rotation–translation, the fraction of all-local-success trials whose whole-task prediction is incorrect is 91.6%. This conditional composition gap is 100.0% for translation–mask, 1.4% for reflection–nonblack recoloring, and 57.1% for source recoloring–cropping. These rates separate composition-dominated families from variation in component competence.

Pair contains exactly four held-out ordered family pairs. Table 18 reports each pair for the same five matched seeds as Table 1. Grid-state re-entry matches the near-saturated latent result on reflection–recoloring and improves the other three families, so the aggregate advantage is not produced by one held-out pair.

Table 18: **The matched interface advantage spans the four held-out Pair types.** Exact-grid accuracy (%), mean ± sample standard deviation across five seeds. Counts are distinct evaluation tasks.

Held-out ordered pair	Tasks	Latent handoff	Grid-state re-entry
Exact recolor → crop	36	2.2 ± 2.3	42.2 ± 15.9
Reflect → nonblack recolor	48	96.2 ± 5.2	97.1 ± 3.5
Rotate → translate	72	41.4 ± 26.5	97.5 ± 1.8
Translate → mask	48	25.0 ± 13.3	99.6 ± 0.9

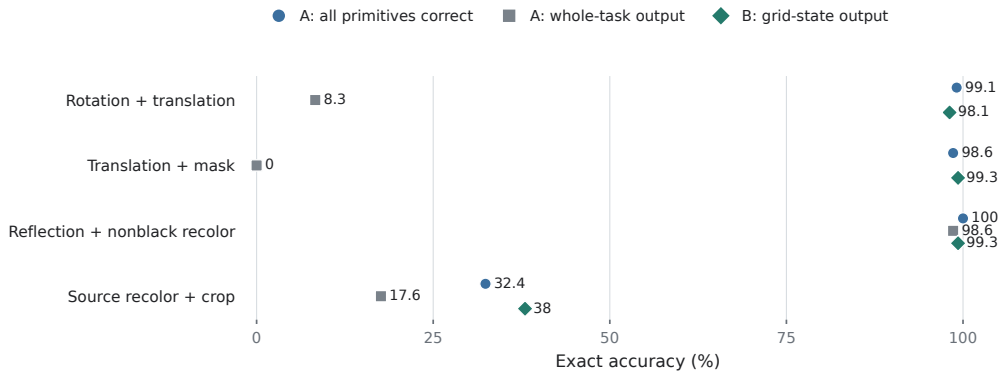

Figure 6: **Operation-family slices separate composition-dominated and component-acquisition regimes.** Rates pool task–seed trials across three seeds. Local success requires the whole-task checkpoint to execute both primitives correctly on reference intermediate inputs; whole-task output uses the combined specification; aligned output uses the separately trained grid-state executor with hard feedback.

C.2 Changed-order evaluation

Counterfactual testing retains each eligible permutation for which the reference program changes the query answer. Changed-order scores use permutations; the same underlying task may contribute more than one permutation. Original-order accuracy uses distinct eligible source tasks (33 Test, 120 Pair, and 90 Depth). Each trained seed is evaluated on the same eligible set.

Table 19: Counterfactual order fidelity of shared grid-state execution. Means and sample standard deviations are over five training seeds. Original-order accuracy refers to the eligible intervention subset.

Split	Permutations	Original-order accuracy	Old target after change	New target
Test	33	75.8 ± 5.7	0.0	82.4 ± 5.8
Pair	120	98.3 ± 1.3	0.0	98.7 ± 1.5
Depth	278	88.2 ± 3.3	0.0	86.8 ± 1.2

D Recognition and Execution as Separate Stages

This stage-specific evaluation asks how effectively alternative execution interfaces preserve a program recovered from demonstrations. Every backend receives the same cached description, making execution the comparison variable.

D.1 Cached Claude Opus 5 descriptions

The external-recognizer experiment uses 246 tasks: 100 from Test, 26 from the auxiliary ParamColor diagnostic, and 60 each from Pair and Depth. Claude Opus 5 (`claude-opus-5`, high effort) is given four demonstrations and the operation vocabulary. One set of descriptions is generated, parsed into concept vectors, and cached for all backend comparisons. The recognizer produces parseable specifications for 232/246 tasks; 231 are exactly correct, comprising 14 unparsable responses and one parsed specification error. All 246 tasks remain in the execution denominator.

The paired seed-0 backend replacement changes 161 correct queries to 217. The three grid-state executor seeds solve 217, 206, and 219 queries. Because the descriptions recover 231 of 246 complete specifications, this comparison measures how effectively each learned neural backend converts the same recognized programs into output grids.

E Inference with the Same Frozen Whole-Task Model

For each seed, we load one trained whole-task executor and its concept encoder, then compare a single full-specification call with primitive-conditioned calls using those identical weights. Self-

rollout decodes each output to a discrete grid, then re-encodes that prediction for the next call. The same frozen second-stage checkpoint supplies every transition from its own predicted states. The supplied decomposition specifies primitive identities and order. All scores below report means and sample standard deviations over three independently optimized second-stage checkpoints.

Table 20: Global-grid-domain exact-grid accuracy (%).

Split	Tasks	Whole task, one call	Same checkpoint, grid states
Test	458	93.2 ± 0.4	95.3 ± 0.3
Pair	204	29.2 ± 2.0	87.6 ± 2.5
Depth	180	61.3 ± 2.2	92.0 ± 1.2

Conditioning sanity check. The following control holds the global-grid checkpoint, call count, and hard feedback fixed while changing only the program supplied at each recurrent call.

Table 21: Conditioning sanity check for frozen global-grid checkpoints. Exact-grid accuracy (%), mean $\pm$ sample standard deviation over three checkpoints. Every condition uses the identical checkpoint, K calls, and hard discrete feedback; only the program requested at each call changes.

Program requested at each of K calls	Pair	Depth
Supplied primitive sequence	$\mathbf{87.6} \pm 2.5$	$\mathbf{92.0} \pm 1.2$
Full program repeated	3.3 ± 1.7	22.6 ± 0.8
First primitive repeated	0.0 ± 0.0	0.0 ± 0.0
Last primitive repeated	0.0 ± 0.0	0.0 ± 0.0

The control verifies that the recurrent evaluation follows the operation supplied at each update. Changed-order results in Table 24 separately test whether the same checkpoint follows a new target when the requested order changes.

Table 22: **The same three second-stage whole-task checkpoints improve under grid-state rollout.** Grounded-object exact-grid accuracy (%), mean $\pm$ sample standard deviation. Each rollout reuses the weights of its one-call counterpart. S–A–T denotes the selector–action–transform holdout, which has two executor calls.

Split	Tasks	Second-stage model, one call	Same checkpoint, grid-state rollout
Test	299	70.2 ± 1.5	97.9 ± 0.2
Cross-step	210	18.7 ± 1.1	98.6 ± 0.5
S–A–T (2 calls)	240	85.4 ± 0.7	98.9 ± 1.2

Across the global-grid and grounded-object domains, each frozen second-stage checkpoint converts supplied primitive sequences into substantially higher composed accuracy. Table 21 holds the global-grid checkpoint and number of calls fixed while varying the operation requested at each update; the following table applies the same control to the three grounded-object second-stage checkpoints.

Table 23: Matched controls over three frozen second-stage whole-task checkpoints (%), mean $\pm$ sample standard deviation. Cross-step and the selector–action–transform (S–A–T) holdout test recombination across supplied primitives.

Conditioning of each second-stage checkpoint	Calls	Cross-step	S–A–T (2 calls)
Full concept, one call	1	18.7 ± 1.1	85.4 ± 0.7
Correct primitives	K	98.6 ± 0.5	98.9 ± 1.2
Full concept repeated	K	0.0 ± 0.0	0.0 ± 0.0
First primitive repeated	K	1.4 ± 0.0	0.0 ± 0.0
Last primitive repeated	K	0.0 ± 0.0	0.0 ± 0.0

In the grounded-object domain, full-program repetition reaches 0% on Cross-step and the S–A–T holdout, compared with 98.6%/98.9% with the supplied sequence. Together with the global-grid

controls in Table 21, these interventions show that each frozen second-stage executor follows the operation assigned to each visual-state update.

Table 24: Frozen-checkpoint changed-order evaluation (%). Original-order accuracy uses distinct source tasks; old/new-target scores use changed-answer permutations. Depth has 90 source tasks and 278 permutations, so the denominators differ.

Split	Tasks	Permutations	Original order	Old target	New target
Test	33	33	72.7 ± 0.0	0.0 ± 0.0	74.7 ± 3.5
Pair	120	120	99.2 ± 0.8	0.0 ± 0.0	99.2 ± 1.4
Depth	90	278	88.5 ± 2.3	0.0 ± 0.0	86.2 ± 1.2

Changed-order evaluation shows the same transition-level fidelity: when the requested order changes the target, matches to the old target are 0% across Test, Pair, and Depth, while new-target accuracy is 74.7%, 99.2%, and 86.2%, respectively.

What full-concept recurrence computes. Repeating a complete program token does not request additional computation toward the same target: it requests the complete transformation again. We therefore construct the symbolic trajectory $P(x), P^2(x), \ldots, P^K(x)$ and score every frozen-checkpoint recurrent state against its corresponding counterfactual target.

Table 25: Repeated-program diagnosis for frozen whole-task checkpoints (%, three seeds). "Final vs. $P(x)$" retains the original task target after K full-concept calls; "Final vs. $P^K(x)$" uses the symbolic repeated-program target; "Full trajectory" requires every recurrent prediction to equal $P^t(x)$. The analysis covers the 179 Depth programs for which all repeated symbolic transitions remain defined.

Split	Valid/all	$P^K(x) \neq P(x)$	Final vs. $P(x)$	Final vs. $P^K(x)$	Full trajectory
Test	458/458	286	36.0 ± 0.2	92.8 ± 0.4	92.6 ± 0.2
Pair	204/204	168	3.3 ± 1.7	32.7 ± 3.7	26.0 ± 1.8
Depth	179/180	128	22.7 ± 0.9	58.1 ± 0.6	54.2 ± 2.0

Repeated full-program calls follow the symbolic target $P^K(x)$ on 90.9% of the 286 changed-target familiar tasks, while matches to the original target fall to 0%. The same target shift appears on Pair: among 168 changed-target programs, repeated-target matches reach 35.7% and original-target matches reach 0%. On the 179 Depth programs with fully defined repeated trajectories, the corresponding rates are 49.7% and 0.3%. Grounded-object analysis likewise uses the tasks for which another complete application preserves the required selection and action preconditions.

These controls identify primitive-conditioned state updates as the mechanism that converts the frozen executor's learned transitions into composed programs.

F Grounding Object Primitives Before Testing Composition

The grounded-object study uses a two-stage design: standalone primitive acquisition followed by composition in a language of color and border-contact selectors, object actions, rotation, and reflection. This construction creates a high-competence population in which cross-step composition can be measured directly.

Primitive acquisition. The 18 categories used for composition reach 98.2% exact-grid accuracy on 3,409 standalone test examples; 17 exceed 90%, and border-touching movement reaches 89.4%. This high-competence population isolates cross-step composition from primitive acquisition in the evaluation below.

Composition data and training. The narrowed dataset contains 4,419 training, 300 validation, and 299 Test tasks. Cross-step holds out bounding-box filling with rotation and object retention with reflection (210 tasks). The selector–action–transform (S–A–T) split contains 240 held-out combinations with observed constituent pairs. It uses two executor calls because selector–action forms

Table 26: **Representation and architecture across the two visual domains.** Both domains use the same cell-level visual vocabulary and input-aligned state interface; the grounded-object domain enlarges the canvas and replaces global-operation slots with selector–action slots.

Component	Global-grid domain	Grounded-object domain
Canvas	14×14 fixed canvas; source grids are 8–14 cells per side	22×22 fixed canvas; source grids are 14–22 cells per side
Visual tokens	One token per canvas cell; 11-entry embedding for colors 0–9 and padding, plus learned row/column position embeddings	Same cell tokenization, color vocabulary, padding symbol, and positional format
Specification vector	9 categorical slots; 38-dimensional slot-wise one-hot vector mapped to one 512-dimensional rule token	7 categorical slots; 50-dimensional slot-wise one-hot vector mapped to one 512-dimensional rule token
Primitive vector	One active global operation family: rotate, reflect, translate, exact recolor, nonblack recolor, mask, or crop	One selector–action step, optionally followed by one rotate/reflect step
Executor	Four-layer, width-512 shared Transformer	Eight-layer, width-512 shared Transformer
Boundary state	14×14 grid over the 11 visual classes	22×22 grid over the same 11 visual classes

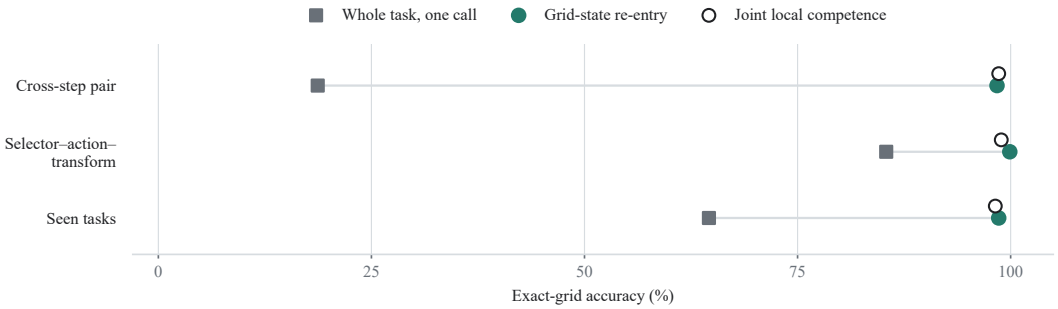

Figure 7: **Grid-state re-entry converts available component competence into composed accuracy.** Each row pools task–seed trials on one evaluation split. Open circles show joint local competence of the whole-task executor on reference intermediate states; squares show its one-call accuracy; filled circles show the separately trained grid-state executor with hard feedback. Cross-step and the selector–action–transform holdout test composition; Test contains familiar composed tasks.

one learned operation and rotation or reflection forms the second. The generator filters ambiguous demonstrations and checks evaluation input overlap with training.

Both variants start from the same eight-layer, width-512 primitive checkpoint (25.5M parameters; 22-cell canvas). Each update combines 32 composition examples and 32 balanced standalone transitions, with equal loss weights. AdamW uses learning rate 10^{-4}, weight decay 0.01, 1,000 warm-up updates, gradient clipping at 1, and 10,000 updates. Validation composition loss selects a checkpoint every 500 updates. The grid-state executor trains with soft feedback and final targets on composed tasks; evaluation below uses hard feedback. Seeds 0–2 vary second-stage optimization while sharing the primitive initialization and dataset; the shared primitive initialization is also evaluated directly, before composition training.

Figure 7 tests whether high standalone primitive competence is converted into composed execution.

On Cross-step, the second-stage whole-task executor has 98.6% joint local competence but 18.7% one-call accuracy, a 79.8-point composition gap before rounding. Grid-state re-entry reaches 98.4%, and rollout from the single shared primitive initialization reaches 97.6%. The $70.2\pm1.5\%$ Test result in Table 27 is one-call accuracy on familiar composed tasks, whereas standalone primitive accuracy is 98.2% over 3,409 examples; it therefore measures composition rather than primitive acquisition. Standalone retention after second-stage training is 98.7–98.9% for the whole-task executor

Table 27: **Grounded operations compose through grid-state re-entry.** Exact accuracy (%). The first two rows report mean ± sample standard deviation over three independently optimized second-stage runs initialized from the same primitive checkpoint. "Shared primitive initialization" is that single checkpoint before second-stage training and therefore has no seed variation. Cross-step and the selector–action–transform (S–A–T) holdout test composition.

Execution	Test	Cross-step	S–A–T (2 calls)
Whole task, one call	70.2 ± 1.5	18.7 ± 1.1	85.4 ± 0.7
Grid-state re-entry, hard	98.1 ± 0.4	98.4 ± 0.5	99.9 ± 0.2
Shared primitive initialization, rollout	96.7	97.6	97.9

Table 28: Object-domain composition diagnostics, pooled across three seeds. Gap is computed before rounding; the last-state oracle supplies the reference input to the grid-state executor's final step.

Subset	Trials	Local	Whole task	Gap	Grid state	Last-state oracle
Test (two steps)	738	98.2	64.6	33.6	98.6	100.0
Cross-step	630	98.6	18.7	79.8	98.4	100.0
S–A–T holdout	720	98.9	85.4	13.5	99.9	100.0

and 98.4–98.9% for the grid-state executor, with every evaluated category above 90%. The domain extends the execution gap from global grid transformations to grounded object operations.

G Matched Commutativity and Intermediate-State Readout

G.1 Matched reflection–translation tasks

A separate 3,002-task dataset holds out reflection–translation. Its two core strata balance marginal reflection axes, translation directions, and distances while varying their joint association and whether the operations commute. Each core stratum contains 96 tasks. The additional noncommuting family contains 120 rotation–translation tasks; the commuting families contain 108 geometry–color or masking–recoloring tasks. The matched core strata isolate commutativity, while the broader families test transfer across operation identities. Models are trained separately on this dataset with three seeds; its Test split is distinct from the primary global-grid Test split.

Table 29: Exact accuracy (%) in matched and other-family strata. NC: noncommuting; C: commuting. Execution rows are three-seed mean and standard deviation; joint local competence pools task–seed trials.

Execution	Core NC	Core C	Family NC	Family C
Whole task, one call	0.0 ± 0.0	10.8 ± 6.8	0.0 ± 0.0	93.8 ± 1.4
Ordered tokens	0.0 ± 0.0	0.3 ± 0.6	0.0 ± 0.0	96.9 ± 1.4
Grid-state re-entry, hard	95.1 ± 2.6	97.6 ± 1.6	70.6 ± 22.9	94.8 ± 1.4
Joint local competence	99.3	99.7	97.8	99.1

High local competence coexists with a large local-to-whole execution gap in both core strata, establishing the phenomenon for commuting as well as noncommuting operations. Alternative-output classification gives the correct target precedence when candidate programs coincide. On noncommuting tasks, zero predictions match the reverse-order reference, localizing the observed gap away from simple order reversal. The matched strata identify input-aligned handoff as the common execution advantage across both algebraic regimes.

G.2 Matched position-wise probes

The matched probe freezes the five latent-handoff and five grid-state checkpoints from Table 1. For each two-or-more-step program, it reads the pixel features after the first primitive-conditioned block and targets the reference intermediate grid. The same position-wise linear color readout, optimizer, fitting budget, and examples are used for both sources; the readout cannot move information between canvas positions. Cases in which the intermediate equals the input or final grid are excluded.

Training uses 3,064 demonstrations from 766 multi-operation training tasks, validation uses 191 queries, and evaluation retains 297 Test, 204 Pair, and 180 Depth queries.

Table 30: **The intermediate grid is readable from both matched state families.** Exact-grid accuracy (%), mean ± sample standard deviation across five frozen executor seeds. Figure 4 shows individual seeds.

Feature source after first primitive	Test	Pair	Depth
Latent handoff, H_1	97.1±2.6	97.4±4.0	95.0±3.8
Grid-state executor	98.7±0.9	98.3±1.4	97.0±1.2

The matched result separates state availability from state use. Both executors retain a locally decodable first intermediate grid, and their end-to-end Pair and Depth accuracies differ by roughly 44 points. The grid-state interface makes the already readable task state directly usable by the next shared block.

Table 31: Scalar readouts from mean-pooled visual tokens (%), three seeds. Geometry describes the first transformation; primitive ID is a different classification target.

Source / target	Test	Core NC	Core C	Family NC	Family C
Whole task: geometry	99.8±0.4	86.5±14.6	90.6±6.3	98.9±1.3	96.9±2.3
Whole task: primitive ID	95.5±0.7	41.7±34.3	40.3±23.8	63.9±10.9	11.7±12.6
Ordered: geometry	99.6±0.2	76.7±22.6	62.5±33.1	65.6±17.8	99.4±1.1
Ordered: primitive ID	94.1±2.0	70.8±21.3	66.7±19.0	51.7±9.3	16.7±10.0
Grid-state first call: geometry	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
Grid-state first call: primitive ID	99.4±0.2	100.0±0.0	99.7±0.6	98.3±2.2	99.4±1.1

The pooled readouts provide a complementary comparison with one-pass executors: transformation attributes can remain accessible even when a complete intermediate canvas is not the model's execution boundary.

H Primitive Alignment and Program Recovery

A recurrent-grid control instantiates the intermediate-state idea from visual scratchpads (Lotfi et al., 2024); a learned primitive predictor separates program recovery from a frozen backend, following ExeDec's high-level decomposition (Shi et al., 2024). Each control isolates one mechanism in the fixed task language.

Table 32: Exact-specification generic visual scratchpad, three seeds (%). Training uses soft recurrent states and final-output loss; soft and hard rows report their corresponding feedback conditions.

Feedback condition	Test	Pair	Depth
Generic scratchpad, soft	91.7±2.0	27.5±5.5	0.0±0.0
Generic scratchpad, hard	29.4±0.8	0.8±1.4	0.0±0.0

The generic scratchpad supplies the full concept plus a learned step-position embedding on each of K recurrent calls. It shares the final-output supervision regime but not the per-call primitive assignment. The three-seed primitive-aligned grid-state control reaches 88.2% on Pair and 91.5% on Depth, establishing the value of assigning the corresponding primitive at each state update.

The sequential decomposer receives demonstrations and the current grid, predicts one of 34 training primitives or a stop symbol, and feeds a frozen aligned backend for at most three calls. Teacher-forced decomposition training uses reference prefixes and 20,000 updates. Reference sequences unlock 91.7/82.8% Pair accuracy and 93.3/90.0% Depth accuracy on seeds 0/1, locating this pipeline's decisive stage in program recovery while keeping the backend fixed.