PROBING TRANSFORMER REASONING TRANSFER THROUGH ALPHABET EQUIVALENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

A reasoner is *abstract* if it applies a general procedure to a problem's structure, indifferent to how that structure is concretely instantiated; it is *concrete* if what it learned is tied to the particular instantiation trained on. This provides a testable marker: an abstract reasoner should be able to transfer mastery of one problem instance to another. We study equivalence between problems via alphabet renaming, which can be defined on any symbolic problem, and instantiate it on Sudoku. A transformer trained on one such puzzle transfers to an equivalent one no better than an untrained network, yet a control that copies over its trained parameters instead of retraining recovers its original performance exactly. Reasoning, in that narrow sense, is abstract; recognizing that a new instance calls for it is not. We probe this recognition gap three ways. Freezing the shared network and adapting only a new alphabet's parameters fares worse than training an unfrozen model from scratch on the same data: prior exposure to other alphabets does not ease learning a new one. Recombining already-known symbols into unfamiliar combinations yields graded, partial success: above chance, short of the seamless transfer an abstract reasoner should show, degrading as combinations diverge from training. Replacing the model's fixed, per-alphabet output vocabulary with a mechanism that selects among a puzzle's own visible symbols helps substantially with recombination, but only marginally with transfer to a truly unseen alphabet. These results isolate a specific failure to recognize alphabet-equivalent instances as such, distinct from a general failure to reason, and show several natural interventions narrow, but do not close, this gap.

1 INTRODUCTION

A model that has learned to solve a problem can be said to reason *abstractly* about it if the procedure it applies operates on the problem's structure, indifferent to the particular form in which that structure is presented; it reasons only *concretely* if what it has learned is tied to the specific surface form it was trained on, instead of to the structure underneath (Fodor & Pylyshyn, 1988). This distinction is central to any claim that a learned system reasons instead of merely fitting patterns in its training distribution, but such claims are frequently made without a test that could actually distinguish the two possibilities. Benchmarks such as SCAN (Lake & Baroni, 2018) and COGS (Kim & Linzen, 2020) test a closely related property, generalizing to unseen *combinations* of familiar components, and have repeatedly found large gaps between in-distribution and compositional-generalization accuracy. We study a different, more basic notion of generalization: not new combinations of familiar parts, but the exact same combinatorial structure presented under a purely relabeled vocabulary, asking a question any account of compositional generalization presupposes an answer to: does the model recognize the relabeled instance as the problem it already knows how to solve at all.

We propose a concrete operationalization of this distinction: an abstract reasoner should treat two different instances of the same underlying problem interchangeably, since its procedure does not depend on which particular instantiation it happens to face. We instantiate "different instances of the same problem" through the simplest transform that leaves a problem's structure completely intact while changing its surface form entirely: renaming its set of symbols.

This gives a general recipe applicable to any symbolic or combinatorial domain: define equivalence via a checkable relabeling, measure whether a model trained on one instance transfers, unaided,

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.

to an equivalent one, and use a targeted *weight-transplant* control, copying a trained model's per-symbol parameters directly onto their counterparts in the new alphabet, to separate a true failure to generalize from the simpler confound that the new instance's symbols were never trained at all.

We instantiate this recipe on Sudoku puzzles, whose instances can be generated inexpensively with a verified unique solution. A transformer trained on one alphabet transfers to an equivalent puzzle in a different alphabet at chance level, indistinguishable from an untrained network, even though the weight-transplant control recovers its original accuracy *exactly*: the model's procedure reuses across equivalent instances once told the symbol correspondence, but it never discovers that correspondence on its own. Probing this recognition gap along three further axes tells a more nuanced story. Pretraining on several known alphabets, and an architecture that replaces the model's fixed output vocabulary with a mechanism selecting among a puzzle's own visible symbols, each narrow the gap somewhat but leave it open. Scaling up training-time diversity then resolves it unevenly: recombining symbols the model already knows becomes essentially solved, while transfer to a truly novel symbol remains just as broken as at the outset.

Contributions.

- An operationalized, reusable methodology for testing knowledge transfer between equivalent instances of a symbolic problem, including a weight-transplant control isolating true reasoning-transfer failure from never-trained parameters.
- Applied to Sudoku, naive transfer across alphabet-equivalent puzzles fails completely, while the transplant control recovers full performance exactly: the failure is one of recognition, not of reusable computation (Section 4.1).
- This recognition gap persists across pretraining diversity and an architecture removing a structural bottleneck that prevents zero-shot generalization by construction; each narrows but does not close it (Sections 4.2–4.3).
- Training-time data diversity, at much larger scale, resolves the gap into two sub-problems: recombining already-known symbols is essentially solved, while transfer to a truly novel symbol remains unsolved by every intervention tested (Section 4.4).

2 Related Work

Neural Sudoku solving. Palm et al. (2018) introduced *Recurrent Relational Networks*, solving Sudoku via iterative message passing between cell-level representations on a graph, the architectural lineage our encoder-based model belongs to. Wang et al. (2019)'s *SATNet* instead embeds a differentiable MAXSAT solver directly in the network; Chang et al. (2020) showed it fails at visual Sudoku without intermediate labels tying each digit image to its symbol, a symbol-grounding failure related to the one we document, though via a different architecture and a different route to severing the symbol-to-meaning link. That work, and subsequent neural combinatorial solvers, target solving accuracy within a single fixed vocabulary; we instead hold that accuracy fixed near ceiling and ask what happens under a symbol relabeling that leaves the problem unchanged.

Compositional and systematic generalization. SCAN (Lake & Baroni, 2018) and COGS (Kim & Linzen, 2020) test whether sequence models generalize to novel *combinations* of familiar primitives, both finding persistent, large gaps between in-distribution and compositional accuracy; Hupkes et al. (2020) formalize this style of test into a general methodology for probing compositionality, and Csordás et al. (2021) show the gap narrows substantially for Transformers specifically under the right training details. Our question is more basic: not combining familiar parts in new ways, only recognizing the exact same combination already trained on, wearing a different vocabulary. That we still find total failure suggests the difficulty these benchmarks document may in part reflect an even more basic failure to abstract away from surface symbol identity at all.

Symbol binding and variable binding. Santoro et al. (2017)'s *Relation Networks* were an early architecture for relational reasoning by construction, though over a fixed input set instead of symbol identity. Webb et al. (2021) is the closest direct precedent for our framing: their *Emergent Symbol Binding Network* uses an indirection mechanism over external memory that lets learned rules

generalize near-perfectly to entirely novel filler entities from a handful of examples, because its computation operates on relations between representations instead of their identity. Altabaa et al. (2024)'s *Abstractors* formalize a version of this idea for Transformers directly, via a relational cross-attention mechanism constrained to depend only on similarity between representations, never their content. Our Section 4.3 pointer-based redesign (predicting a blank cell's value by selecting among symbols visible elsewhere in the puzzle, a direct application of Pointer Networks (Vinyals et al., 2015) and pointer-generator copy mechanisms (See et al., 2017)) asks a narrower question: whether removing a fixed, identity-keyed output vocabulary, without also constraining the encoder's internal computation to be relational, is enough on its own. Our results (helping substantially with one sub-problem, only marginally with the other) indicate it is not, motivating Abstractors' more explicit mechanism as a next step (Sections 4.4, 5).

Symbolic mechanisms in pretrained language models. Contemporaneously, Yang et al. (2025) identify an emergent symbolic architecture inside pretrained large language models (symbol abstraction, symbolic induction, and retrieval heads operating on abstract variables instead of raw tokens) supporting abstract reasoning in a related sense, and Feng & Steinhardt (2024) similarly find LLMs bind entities to attributes in context via dedicated binding-identity representations. Their results establish that such mechanisms can exist inside an already-trained model; our question is different: which interventions cause the analogous recognition gap to appear or close, which requires controlling training itself, in the spirit of work training small transformers from scratch on algorithmic tasks to study generalization in isolation (Power et al., 2022; Veličković & Blundell, 2021). We train small transformers under fully controlled data, so that pretraining diversity, architecture, and training-time mixing can each be manipulated one at a time and any resulting behavior change attributed directly to it.

3 Problem Setup and Method

We fix Sudoku terminology used throughout the paper: a puzzle fixes some cells (the *fixed cells*) of a 9×9 grid, partitioned into nine 3×3 boxes, and leaves the rest blank; a *solution* fills every blank cell so that each of 9 symbols appears exactly once in every row, column, and box, and we only use puzzles with a unique solution. Section 3.1 below defines, for combinatorial problems in general, what it means for two puzzles to be equivalent under a relabeling of their symbols, and the notation we reuse throughout the paper; the remaining subsections instantiate the rest of our setup for Sudoku specifically.

3.1 Equivalence of combinatorial problems

A combinatorial problem is defined over n roles subject to some rules; a puzzle is stated using an *alphabet*: an injective map A from the n canonical roles $\{1, \ldots, n\}$ to n distinct tokens, together with a fixed token 0 reserved as blank and shared by every alphabet (for Sudoku, $n = 9$). For a puzzle P stated in the canonical alphabet (roles $1, \ldots, n$ used directly as tokens), write $A(P)$ for the puzzle obtained by relabeling every role d to token $A(d)$, leaving every other aspect, positions, structure, and rules, unchanged; we call $A(P)$ "puzzle P in alphabet A."

This relabeling preserves validity unconditionally: applying the same substitution everywhere carries any solution of P to a solution of the correspondingly relabeled puzzle. Two puzzles are *equivalent* exactly when they are $A(P)$ and $B(P)$ for the same underlying P and two alphabets $A \neq B$: solving one is the same problem as solving the other, under a different alphabet.

3.2 Data

We generate 200,000 puzzles once, in a canonical alphabet (tokens 1 to 9), each with 40 fixed cells and a solution verified unique by constraint propagation with backtracking, and split them once into a fixed 90/5/5 train/val/test partition (180,000 / 10,000 / 10,000 puzzles; full generation settings in Table 7, Appendix A.1). For any other alphabet, we obtain its puzzle set by applying that alphabet to this same partition, so every alphabet's split covers the same underlying puzzles, and comparisons across alphabets are over *identical* underlying puzzles.

Each alphabet is a disjoint block of 9 token ids: the k-th letter (A, B, C, ...) simply shifts every token id by $9(k-1)$, so alphabet A is the canonical alphabet itself, B shifts every id by $+9$, and so on. We designate one letter, E, as held out from every model's training data in every experiment, so it consistently serves as the truly-unseen-symbol test throughout.

3.3 MODELS

Both architectures we study share an encoder body: token embeddings, additive row/column/box positional embeddings, and a standard pre-norm Transformer encoder (Vaswani et al., 2017) (8 layers, $d_{\text{model}} = 384$, 8 attention heads; full architecture hyperparameters in Table 7, Appendix A.1). They differ only in how a blank cell's encoded representation is turned into a predicted token, illustrated side by side in Figure 1.

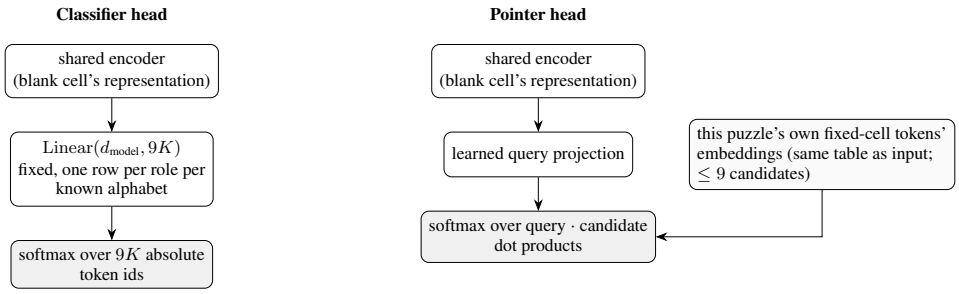


Figure 1: The two architectures share an encoder body and differ only in how a blank cell's representation becomes a predicted token. The classifier head (left) scores a fixed set of $9K$ absolute token ids, requiring dedicated trained rows for every alphabet ever seen. The pointer head (right) instead scores only the token ids actually present among that specific puzzle's fixed cells, so its output space never depends on how many alphabets were seen during training.

Classifier head (baseline). A linear layer maps each cell's representation to $9K$ logits, one per role per alphabet among the K alphabets ever included in training, and predicts the literal token id directly (never a "role within some alphabet," which would presuppose the model already knows its own alphabet's identity). This head's size is fixed at training time: a novel alphabet's 9 logit rows do not exist yet and cannot be added without retraining, which is the structural bottleneck motivating the pointer head described next.

Pointer head. For each blank cell, we instead identify the (at most 9) distinct token ids that appear among that specific puzzle's own fixed cells, and predict which of them belongs in the cell. A learned query, computed from the blank cell's encoded representation, is compared via dot product against each candidate's raw token-embedding vector (the same embedding table used on the input side; no separate output projection exists), and the resulting scores are softmaxed over the present candidates. This candidate set is always drawn from the puzzle's own fixed cells, so it never exceeds 9 candidates, unlike the classifier head's $9K$-sized output, which grows with K. A puzzle's target is only well-defined for a blank cell if the correct token is itself among that puzzle's own fixed cells, not merely somewhere in the completed grid; measured directly on our generator, 0.80% of puzzles violate this. We filter such puzzles out for training, and deliberately leave them in for evaluation, so reported numbers reflect this real (if small) architectural limitation.

Both architectures optionally support RRN/IREM-style unrolling (Palm et al., 2018; Du et al., 2022) of the shared encoder; all experiments here use a single pass ($T = 1$).

3.4 THE WEIGHT-TRANSPLANT CONTROL

Naive transfer between equivalent puzzles $A(P)$ and $B(P)$ (Section 4.1) cannot tell whether a model's failure to solve $B(P)$ reflects reasoning that does not generalize across alphabets, or simply that B's tokens never received a training signal, leaving their parameters at random initialization regardless of how good the model's reasoning otherwise is. The weight-transplant control isolates the

second explanation: for the classifier head, we copy, for every role $d \in \{1, \ldots, 9\}$, A's token-d row of the token embedding table and of the $\mathrm{Linear}(d_{\text{model}}, 9K)$ output head (row and bias) directly onto B's token-d row (a plain tensor copy, no gradient step or retraining) before evaluating on B. This tells us how the model performs on B once it has B's own trained parameters, without B itself ever appearing during training. The pointer head needs no analogous control: it never has per-alphabet output parameters to copy in the first place, which is precisely its motivation.

3.5 Metrics

We report two metrics. *Cell accuracy* is the fraction of blank cells filled with the token recorded in our data. *Exact match rate* is the fraction of boards where every blank cell matches.

The weight-transplant control (Section 3.4) copies one alphabet's row onto another's, creating a tie an unrestricted argmax would break by favoring whichever alphabet has the lower token offset, an implementation artifact, not anything the model computed. We therefore restrict the classifier head's argmax to the evaluated alphabet's own 9 logits (instead of all $9K$) whenever the transplant control is in play; this does not hand the model information it lacks, since which alphabet a puzzle belongs to is already visible in its own fixed cells. The pointer head needs no such fix, its candidate set already restricted to the puzzle's own tokens by construction.

Because prediction is always restricted to the specific tokens a puzzle's own fixed cells establish as belonging to it, and every puzzle has a verified unique solution, exact match rate directly measures whether the model produced a legal completion (no repeated symbol in any row, column, or box). Cell accuracy remains a distinct, finer-grained per-cell measure.

3.6 Training protocol

All models are trained with AdamW (Loshchilov & Hutter, 2019), batch size 256, for 20 epochs unless otherwise noted, on a single A100 GPU per run. Table 7 (Appendix A.1) lists the full hyperparameters, including every condition's epoch count and seed count and the exceptions to these defaults.

4 Experiments and Results

Unless noted otherwise, every experiment below uses the data, model, and metrics of Section 3, trained on a single A100 GPU per run.

4.1 Does the model discover the equivalence on its own?

The classifier head's disjoint per-alphabet output rows mean it cannot use a row it never trained; that much is architectural. What is not settled is how total the resulting failure is, since a shared encoder processing both alphabets could plausibly leak some above-chance signal even through untrained output rows. We train the classifier head on alphabet A only, separately on alphabet B only, and (as a reference check) on their union, and evaluate each single-alphabet model three ways: on its own alphabet (*self*); on the other, unaided (*naive cross*); and on the other with the weight-transplant control applied (*transplanted cross*).

Model	Eval alphabet	Mode	Cell acc.	Exact match
A	A	self	99.41%	92.2%
A	B	naive cross	11.28%	0%
A	B	transplanted cross	99.41%	92.2%
B	B	self	99.46%	92.3%
B	A	naive cross	10.95%	0%
B	A	transplanted cross	99.46%	92.3%
random init	(either)	floor	11.10–11.17%	0%

Table 1: Naive cross-alphabet transfer lands exactly at the random initialization floor; the transplant control recovers the *self* mode's number exactly.

Table 1 shows naive cross-alphabet transfer landing exactly at the random initialization floor: not somewhat worse than *self*, but statistically indistinguishable from a network never trained at all. The transplant control recovers the self mode's number to four significant figures, with no retraining. The only thing separating 11% from 99% is knowing the symbol correspondence, not some deeper representational gap: solving competence is bound to the specific tokens trained on, and does not spontaneously discover that a renamed puzzle is the same problem already known how to solve.

4.2 Does more alphabet diversity help?

Experiment 4.1 used one alphabet per model. We ask two further questions with a classifier pretrained on $K = 6$ alphabets (A,B,C,D,F,G; E held out), 60,000 puzzles per alphabet, 20 epochs (99.57% cell accuracy across all six by the end of training; full settings in Table 7, Appendix A.1).

Few-shot adaptation. For held-out E, given $n \in \{100, 1000\}$ labeled E-puzzles (3 seeds each), *finetune* (freezing the pretrained body, adapting only E's own rows) and *scratch* (a fresh, unfrozen model) are indistinguishable at $n = 100$ (26.8% vs. 26.5% mean cell accuracy); at $n = 1000$, scratch clearly wins (69.2% vs. 39.5%, finetune nearly flat across seeds). If the frozen body were a portable, alphabet-agnostic solving procedure, reusing it should have been easier than relearning everything from scratch; instead it is entangled with the specific alphabets it was trained on, not a clean, reusable abstraction (full numbers in Appendix A.3.1).

Mixed-alphabet compositional test. Using the same $K = 6$ checkpoint, we recombine the 6 known alphabets instead of testing yet another one: each of the 9 roles independently selects one of the 6 known alphabets to draw its token from, so the same alphabet can, and typically does, supply more than one role. We call one such role-to-alphabet assignment a *trial*, evaluated once over the entire held-out test set. Throughout the paper we reuse the same 12 fixed trials: *control* (all 9 roles from alphabet A), *maximally mixed* (roles cycle through all 6 alphabets in order), and 10 independently drawn *random* trials, summarized by range and mean (full definitions in Appendix A.2). The resulting combination was never presented as one coherent alphabet during training, even though every individual symbol is well learned.

Trial	Cell accuracy
control (all roles from alphabet A)	99.49%
maximally mixed (9 roles, 6 alphabets, cycled)	44.5%
10 random mixes	23.2%–70.8% (mean 45.4%, s.d. 12.4%)

Table 2: Recombining known alphabets across the 12 fixed trials (Appendix A.2): well above the roughly 11% chance level, well below the 99% ceiling, with accuracy tracking how coherent the specific recombination is.

Table 2 shows accuracy far above the roughly 11% chance level but far below the 99% ceiling, tracking how the trial's 9 role-to-alphabet assignments are distributed: a trial where most roles draw from just one or two of the 6 alphabets scores high, while one spread evenly across all 6 scores low. The model has partial, graded compositional ability instead of a clean, disentangled binding of each symbol to its meaning.

4.3 Does removing the output-vocabulary bottleneck help?

We repeat the experiment with the same hyperparameters ($K = 6$ alphabets, 60,000 puzzles per alphabet, 20 epochs; Table 7), with the pointer head (Section 3) instead of the classifier head, same held-out E, for direct comparability.

Table 3's in-distribution row matches the classifier head's equivalent in-distribution check on this same $K = 6$ setup (99.57% cell accuracy, Section 4.2) almost exactly: swapping in the pointer head costs essentially nothing on the easy, in-distribution case. On the zero-shot row, zero percent board-level correctness is the more decisive number: not one of 10,000 held-out boards was fully solved. The 14.47% cell accuracy is a small but statistically real signal above the roughly 11.1% chance floor, unlike the classifier's flat 11.8% (Section 4.2): the architecture change measurably changes what the model does with E's tokens, even though it never once produces a fully valid board.

Condition	Cell accuracy	Exact match rate
In-distribution, A/B/C/D/F/G (each)	99.3%	92.5%–93.0%
Zero-shot, held-out E	14.47%	0%

Table 3: Same $K = 6$ pointer-head checkpoint: in-distribution accuracy on each of the 6 known alphabets (without recombination, tested separately in Table 4), versus zero-shot on held-out alphabet E (no fine-tuning, no E-puzzles seen during training).

Trial	Classifier	Pointer	Δ
control (A)	99.5%	99.4%	$\sim$0
maximally mixed cycle	44.5%	34.9%	-9.6pp
floor across 10 random trials	23.2%	34.4%	–
mean of 10 random trials	45.4% (s.d. 12.4)	52.8% (s.d. 14.3)	$+7.4$pp

Table 4: Same mixed-alphabet protocol as Table 2, same seeds, classifier versus pointer head. Full 12-trial breakdown in Appendix A.3.2.

Table 4 shows a real average improvement, helping most on the classifier's worst trials (floor rises from 23.2% to 34.4%), though unevenly: 5 of 11 non-control trials get worse (Appendix A.3.2), including the most systematic one (maximally mixed cycle). The architecture change helps more with the easier question (recombining known symbols) than the harder one (a truly novel symbol), in both cases by a modest instead of decisive margin. Why the pointer head shows any positive zero-shot signal at all, where the classifier shows none, is likely a property of the vector-equality comparison its architecture relies on; Section 5 discusses this candidate explanation and its limits.

4.4 Does more training diversity close the gap?

We test whether Section 4.3's small zero-shot improvement depends on training diversity: with no architecture change, we apply Section 4.2's per-role random alphabet selection during training itself, instead of 6 fixed alphabets, and grow the alphabet pool from 6 to the full 25 non-E alphabets, so every training example draws a random set of 9 symbols from that larger collection.

Training reaches 98.41% cell accuracy on the held-out (also randomly mixed) validation set by epoch 19, over 20 epochs of $\sim$178,560 fully-represented puzzles (no repetition needed, since every epoch's pass draws a new random assignment for each puzzle).

Condition	Cell accuracy	Exact match rate
In-distribution, all six alphabets	98.2%	76.9%–78.0%
Zero-shot E, fixed-alphabet training (4.3)	14.47%	0%
Zero-shot E, random-mix training (this section)	15.85%	0%

Table 5: In-distribution accuracy, testing each alphabet as a pure coherent block as in Table 3, and zero-shot transfer to held-out alphabet E, comparing this random-mix-trained model against Section 4.3's fixed-alphabet pointer.

The in-distribution row is notably lower than Section 4.3's fixed-alphabet pointer (99.3% cell, 92.5–93.0% exact match), since this model almost never saw a pure single-alphabet puzzle during training, making a coherent alphabet itself a rare, close-to-out-of-distribution case for it. On zero-shot transfer to E, cell accuracy bumps up slightly but exact match rate stays at exactly 0%: training diversity, at least at this scale, does not meaningfully close the zero-shot gap.

Table 6 shows the trial-to-trial variance seen in every earlier condition (classifier range 23–71%, Table 2; fixed-alphabet pointer range 34–76%, Table 4 and Appendix A.3.2) has essentially vanished: all 12 trials now score within a 0.3 percentage-point band. Random-mix training exposes the model to an enormous number of recombinations of the same known letters, so no new combination is truly novel to it by the end of training; alphabet E, never in that pool, gets no such benefit, its embedding remaining exactly as untrained as ever. The same table shows this regime trades some peak performance for a large gain in consistency: narrow, fixed-alphabet training sharpens performance on familiar cases (coherent-alphabet exact match near 93%) but leaves the model fragile outside them

	Fixed-alphabet training (4.3)	Random-mix training (this section)
Cell acc., control (A)	99.4%	98.2%
Cell acc., maximally mixed cycle	34.9%	98.3%
Cell acc., range, 10 random trials	34.4%–76.3%	98.0%–98.3%
Cell acc., mean, 10 random trials	52.8% (s.d. 14.3)	98.2% (s.d. 0.1)
Cell acc., range, all 12 trials	34.4%–99.4%	98.0%–98.3%
Exact match, range, all 12 trials	0%–92.6%	76.3%–79.5%

Table 6: The same 12 fixed trials as Tables 2–4 (Appendix A.2), for two training regimes of the pointer head. *Range, all 12 trials*: lowest and highest value across all 12 trials.

(0–93% across mixed trials), while the constantly varying distribution costs some peak performance (77–79%) for far more uniform behavior throughout (76–80%).

4.5 Disentangling pool size from mixing exposure

Section 4.4 changed two variables relative to Section 4.3 at once: the alphabet pool grew from 6 to 25 letters, and training switched from never mixing letters within a puzzle to always mixing them. Holding pool size fixed at Section 4.3's 6 letters while giving it Section 4.4's always-mixed training format reproduces the same tight, uniform band of mixed-alphabet accuracy seen with 25 letters, in fact slightly better (98.5–98.7% versus 98.0–98.3% cell accuracy; Appendix A.4). Going from 6 to 25 letters bought essentially nothing extra for this sub-problem: the variable that matters is whether training ever exposes the model to within-puzzle mixing at all, not how large the alphabet pool is (full tables and per-trial figure in Appendix A.4).

On zero-shot transfer to held-out E, this condition reaches 11.46% cell accuracy and 0% exact match rate: the small positive signal seen in Sections 4.3 (14.47%) and 4.4 (15.85%) is essentially absent here, close to the roughly 11.1% chance floor. Exact match rate stays at exactly 0% across all three conditions, so the headline conclusion is unchanged either way, but this leaves open whether the weak signal in the other two conditions depends on genuine pool diversity or is simply run-to-run noise across single training runs (Section 6).

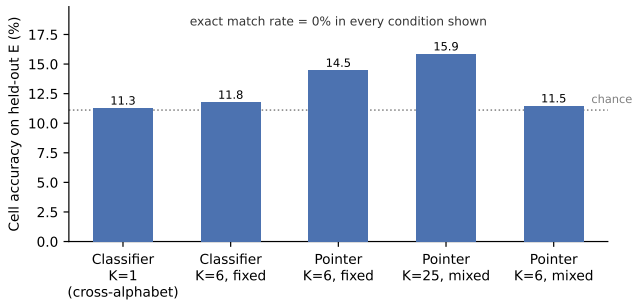


Figure 2: Cell accuracy on held-out alphabet E, zero-shot, across every condition tested (Sections 4.1–4.5). No condition rises meaningfully above the $1/9$ chance floor, and exact match rate is exactly 0% throughout: none of pretraining diversity, an alphabet-agnostic output mechanism, or exposure to within-puzzle mixing produces a single fully consistent completion on a symbol the model never trained on.

5 Discussion

A single failure mode appears from every angle we tested, resolving into two sub-problems with different answers. **Recombining known symbols is solved by training diversity, specifically by mixing exposure instead of pool size.** Sections 4.2 and 4.3 showed graded, partial degradation on recombined-alphabet puzzles under training that never mixed alphabets within a puzzle; Section 4.4 showed that degradation essentially vanishes once training always mixes a large pool of letters; Section 4.5 located why, reproducing the same near-perfect uniformity with only 6 letters as long

as training always mixes them. This part of the entanglement was, in the end, a matter of what the training distribution exposed the model to, and a cheap property to supply.

Transfer to a symbol never seen at all is not solved by anything we tried. Neither the architectural change (Section 4.3) nor either training-diversity condition (Sections 4.4–4.5) moved held-out alphabet E's exact match rate off exactly 0%. Cell accuracy's small residual signal is less consistent than it first appeared, rising from 11.8% (Section 4.2) to 14.47% (4.3) to 15.85% (4.4) but falling back to 11.46%, close to chance, in Section 4.5; whether this reflects a real dependence on pool diversity or run-to-run variation cannot be settled with one seed per condition. Either way, nothing tried gives the model a way to discover what an unseen symbol means well enough to produce a single fully valid board.

The result underlying both halves is Section 4.1's transplant control: the model's procedure reuses across equivalent instances exactly, once told the symbol correspondence. Not the computation but the recognition step remains bound to specific training instances, consistent with the broader shortcut-learning literature (Geirhos et al., 2020); every intervention here either supplies recognition capacity directly (the transplant control) or tries to induce it indirectly, and none of the indirect routes closed the harder half of the gap. One candidate account for the small residual signal in Sections 4.3–4.4, absent in Section 4.5: a "do not repeat this symbol nearby" detector needs only compare two representations for equality, not know their content, so it could transfer to never-trained embeddings by accident (untested, left to future interpretability work). Since no data-side intervention touched the harder half of the gap, the most promising next step is architectural: a mechanism constraining computation to operate on similarity between representations by construction, instead of leaving that as one option among others. Abstractors (Altabaa et al., 2024) implement exactly this for Transformers, and are a natural candidate for closing what training-side changes could not.

6 Limitations

Every model here is a small transformer trained from scratch under fully controlled data; we make no claim about large pretrained models, whose training history spans far more surface variation than any single relabeling could add. This is deliberate: a large pretrained model could appear to recognize a relabeled puzzle's equivalence through mechanisms our setting excludes, e.g. chain-of-thought translation (Wei et al., 2022), tool use, or memorized pretraining exposure to similar puzzles, none of which would show its own representations became alphabet-invariant (Yang et al., 2025).

We deliberately study one domain, Sudoku, instead of several, and one equivalence transform, relabeling the symbol alphabet, out of several a puzzle could plausibly support (e.g. permuting rows and columns). The pointer head (Section 4.3) has a structural blind spot: it can only predict a token already present among a puzzle's own fixed cells, and 0.80% of puzzles from our generator have one that does not; we filtered this out of training and left it in for evaluation. Most training conditions in Section 4 are a single run (the few-shot sweep, Section 4.2, is the exception, at 3 seeds); the residual zero-shot signal (Section 5) moved non-monotonically, possibly run-to-run noise, though exact match rate stayed at exactly 0% regardless.

7 Conclusion

We provided a checkable operationalization of a question usually left informal, whether a model's competence transfers between instances of the same problem differing only in surface symbols, with a control separating a true failure to transfer reasoning from the simpler fact that some parameters were never trained. Applied to Sudoku, naive transfer fails completely, while the transplant control recovers full performance exactly: the model's procedure is reusable, but nothing in ordinary training lets it recognize, on its own, that a new instance calls for it. Recombining known symbols turned out to be a matter of training diversity and is now essentially solved; transferring to a symbol never encountered at all was untouched by every intervention we tried. That split result points clearly to what comes next: an architecture constraining its computation to operate on relations between representations, instead of merely allowing it to.

AI USE STATEMENT

AI tools (Claude Code) were used throughout the research and writing process as collaborative assistants, in an iterative feedback loop with the authors instead of autonomous generation of the research: the main research ideas and conceptual motivation originated from the authors, who continuously reviewed, discussed, and directed every subsequent step. Among tasks with required disclosure, we used generative AI to help develop and refine conceptual framing (operationalizing alphabet equivalence and the weight-transplant control), propose and refine hypotheses (e.g. the vector-equality account in Section 5), design and give feedback on research methodology and experiments (including the mixed-alphabet trial protocol and the disentangling ablation, Section 4.5), implement methods (the model architectures, training, and evaluation code), generate the synthetic Sudoku dataset, and interpret results. We did not use generative AI tools to formulate mathematical claims, provide ingredients for or write proofs, assist with translation, or support qualitative or thematic data analysis; these are not applicable to this work. Among tasks with recommended disclosure, we used generative AI to draft and edit sections of this paper, search for and format relevant literature, create the paper's figures, and suggest this appendix's structure. All AI-assisted code, experimental procedures, results, and text were reviewed and verified by the authors, who determined the final content, methodology, and interpretation; we take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Code to reproduce every experiment in this paper, including data generation, model architectures, training, and evaluation, is available at an anonymized repository: `https://anonymous.4open.science/r/equivalent-problems-transformers-98D3/`. Section 3 defines the datasets, architectures, and metrics used throughout; Appendix A.1 lists every hyperparameter, including the exceptions to the defaults; and Appendix A.2 spells out the fixed mixed-alphabet trial protocol reused across Section 4.

REFERENCES

Awni Altabaa, Taylor Webb, Jonathan Cohen, and John Lafferty. Abstractors and relational cross-attention: An inductive bias for explicit relational reasoning in Transformers. In *International Conference on Learning Representations*, 2024.

Oscar Chang, Lampros Flokas, Hod Lipson, and Michael Spranger. Assessing SATNet's ability to solve the symbol grounding problem. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 619–634, 2021.

Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Learning iterative reasoning through energy minimization. In *International Conference on Machine Learning*, pp. 5570–5582, 2022.

Jiahai Feng and Jacob Steinhardt. How do language models bind entities in context? In *International Conference on Learning Representations*, 2024.

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

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

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.

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 (EMNLP)*, pp. 9087–9105, 2020.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pp. 4487–4499, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Adam Santoro, David Raposo, David G. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, 2017.

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 and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pp. 2692–2700, 2015.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554, 2019.

Taylor W. Webb, Ishan Sinha, and Jonathan D. Cohen. Emergent symbols through binding in external memory. In *International Conference on Learning Representations*, 2021.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Yukang Yang, Declan Campbell, Kaixuan Huang, Mengdi Wang, Jonathan Cohen, and Taylor Webb. Emergent symbolic mechanisms support abstract reasoning in large language models. In *International Conference on Machine Learning*, 2025.

A Appendix

This appendix follows the order of the paper. Section A.1 lists the hyperparameters used. Section A.2 spells out the fixed mixed-alphabet trial protocol that Sections 4.2–4.4 reuse and refer back to. Section A.3 gives the full numeric detail behind two results the main text summarizes. Section A.4 reports the disentangling experiment (Section 4.5) in full, including its zero-shot result. Section A.5 accounts for the compute used.

Data	
Puzzles generated	200,000, once, canonical alphabet
Givens per puzzle	40
Split	180k / 10k / 10k train/val/test, fixed indices
Model (both heads)	
Encoder	8-layer pre-norm Transformer, $d_{\text{model}} = 384$, 8 heads
Feedforward dimension	1536
Dropout	0.1
Positional embeddings	additive row/column/box embeddings
Unrolled iterations	1 (architecture supports more; unused here)
Pointer query projection	$\text{Linear}(d_{\text{model}}, d_{\text{model}})$
Training	
Optimizer	AdamW, lr 3×10^{-4}, weight decay 0.01
Batch size	256
Epochs, single/union alphabet (4.1)	30
Epochs, K-alphabet pretraining (4.2–4.5)	20
Epochs, few-shot finetune/scratch sweep (4.2)	150
Seeds, few-shot finetune/scratch sweep	3 (0, 1, 2) per $(n, \text{condition})$
Seeds, all other conditions	1

Table 7: Hyperparameters used across all experiments. Section labels indicate where each setting is first used.

A.1 HYPERPARAMETERS

Table 7 lists every hyperparameter used across the experiments in this paper, grouped by data generation, model architecture, and training. One condition deviates from these defaults; the paragraph after the table explains it.

The few-shot *finetune* condition (Section 4.2) uses weight decay 0, not 0.01. This condition adapts a model pretrained on $K = 6$ alphabets to held-out alphabet E by freezing everything except E's own embedding and output rows: the shared encoder and positional embeddings are fully frozen, and the token-embedding and classifier-head rows belonging to alphabets other than the target one are kept fixed by masking their gradient as soon as it is computed, instead of zeroing it afterward. AdamW's weight decay shrinks every parameter on every step regardless of its gradient, so a masked-gradient row would still drift under the table's default nonzero weight decay; hence weight decay 0 for this condition only.

A.2 MIXED-ALPHABET TRIAL PROTOCOL

Sections 4.2, 4.3, and 4.4 all evaluate the same fixed set of 12 alphabet recombinations, reporting only their range and mean in the main text to keep the tables short. A *trial* is a single, fixed choice of which alphabet supplies each of the 9 roles, evaluated once over the entire held-out test set to produce one accuracy number, not a repeated statistical sample. Table 8 gives the exact role-to-alphabet assignment for the two deterministic trials; the 10 *random* trials are independently drawn (seed 0, in the order the evaluation script generates them) rather than hand-picked, so their range and mean summarize genuine sampling variability, not cherry-picking.

	Role								
Trial	1	2	3	4	5	6	7	8	9
control	A	A	A	A	A	A	A	A	A
maximally mixed	A	B	C	D	F	G	A	B	C

Table 8: Exact role-to-alphabet assignment for the two deterministic trials reused throughout Section 4. The 10 random trials are independently drawn, not tabulated here; wherever a table reports "10 random trials," it means their range and mean.

A.3 Supplementary results for Sections 4.2 and 4.3

In Section 4.2 we stated that, when adapting a model pretrained on $K = 6$ alphabets to held-out alphabet E, finetuning its frozen body and training an unfrozen model from scratch are indistinguishable for $n = 100$, while for $n = 1000$, the model trained from scratch clearly outperforms the other. We provide precise numbers in Section A.3.1. In Section 4.3 we stated that the pointer head gives a real but uneven improvement over the classifier head on recombined alphabets. We provide precise numbers in Section A.3.2.

A.3.1 Few-shot adaptation to held-out alphabet E

Condition	$n = 100$	$n = 1000$
zero-shot (no fine-tuning)	11.8%	–
finetune (frozen body)	26.8%	39.5%
scratch (unfrozen)	26.5%	69.2%

Table 9: Mean cell accuracy on E's held-out test split, for the $K = 6$ pretrained classifier of Section 4.2, given $n \in \{100, 1000\}$ labeled E-puzzles (3 seeds each). At $n = 1000$ the finetune condition is nearly flat across all 3 seeds.

At $n = 100$, finetune and scratch land within half a percentage point of each other, both barely above the 11.8% zero-shot floor: 100 labeled examples is not enough for either condition to make much headway, frozen body or not. The gap only opens up at $n = 1000$, where scratch pulls decisively ahead (69.2% versus 39.5%); the finetune condition's low seed-to-seed spread there suggests a stable ceiling on what the frozen pretrained body can support, instead of a noisy artifact of any one run.

A.3.2 Full mixed-alphabet per-trial breakdown

Table 10 expands Table 4 into the full per-trial breakdown, reporting classifier and pointer-head accuracy separately for every trial of the mixed-alphabet protocol (Appendix A.2).

Trial	Classifier	Pointer	Δ
control (A)	99.5%	99.4%	$\sim$0
maximally mixed cycle	44.5%	34.9%	-9.6pp
random 0	70.8%	59.5%	-11.3pp
random 1	53.1%	75.8%	$+22.7$pp
random 2	42.5%	46.5%	$+4.0$pp
random 3	38.7%	45.3%	$+6.5$pp
random 4	23.2%	43.0%	$+19.8$pp
random 5	39.2%	76.3%	$+37.1$pp
random 6	45.6%	42.7%	-2.9pp
random 7	54.0%	45.9%	-8.0pp
random 8	41.1%	34.4%	-6.7pp
random 9	46.3%	58.7%	$+12.4$pp
mean of 10 random trials	45.4% (s.d. 12.4)	52.8% (s.d. 14.3)	$+7.4$pp

Table 10: Full per-trial breakdown behind Table 4, same mixed-alphabet protocol as Table 2 (Appendix A.2), same seeds, classifier versus pointer head. "A" denotes the control trial, in which every one of the 9 roles is assigned the same alphabet A (Table 8), i.e. the unmixed case.

The value of Δ is highly variable, from -11.3pp (random 0, the pointer head's worst trial relative to the classifier) to $+37.1$pp (random 5, its best), a spread almost as wide as the trials' raw accuracy range. On any single recombination, then, which architecture is more performant is close to a coin flip; it is only in aggregate, across many trials, that the pointer head's advantage on this sub-problem becomes visible (mean $+7.4$pp). The practical consequence is that a single held-out recombination is not a reliable benchmark for comparing the two architectures: which one wins on any given trial can flip with a different draw of the alphabet-to-role assignment, so only a many-trial average supports a comparison claim. This is consistent with the main text's reading (Section 4.3): the architecture change is a real but modest help, not a decisive fix for any one puzzle.

A.4 DISENTANGLING POOL SIZE FROM MIXING EXPOSURE

In Section 4.3 we reported experiments training the pointer head on 6 fixed alphabets, on pure (unmixed) puzzles, and found trial-to-trial cell accuracy on the mixed-alphabet test ranging 34.4%–76.3%. In Section 4.4 we instead trained on a pool of 25 alphabets, on mixed puzzles, and found this variance collapse to a tight 98.0%–98.3% band. Those two conditions differ in two variables at once - pool size ($K = 6$ versus $K = 25$ alphabets) and mixing exposure (pure versus mixed) - so Section 4.5 asks which one actually drove the collapse. This condition isolates the second variable alone, holding the pool fixed at Section 4.3's original 6 letters while giving it Section 4.4's always-mixed training format; the tables below report the full comparison behind that section's answer.

Cell accuracy	K=6, never mixed	K=25, always mixed	K=6, always mixed
Range, 10 random trials	34.4%–76.3%	98.0%–98.3%	98.5%–98.7%

Table 11: Mixed-alphabet compositional test under three training conditions.

Exact match rate	K=6, never mixed	K=25, always mixed	K=6, always mixed
Range, 12 trials	0%–92.6%	76.3%–79.5%	81.0%–84.3%

Table 12: Same 12 trials and three training conditions as Table 11, board-level metric instead of cell accuracy.

The answer is mixing exposure and not pool size. Table 11 shows that the $K = 6$, always-mixed condition reproduces - and, on cell accuracy, marginally beats - the tight band achieved with a pool of 25 alphabets (98.5–98.7% versus 98.0–98.3%), and Table 12 shows the same pattern on the stricter board-level metric (81–84% versus 76–80% exact match). The $K = 6$ condition's slight edge is consistent with the same learning capacity being spread across fewer distinct alphabets. Growing the pool fivefold, from $K = 6$ to $K = 25$ alphabets, therefore bought essentially nothing for this sub-problem on its own; what matters is whether training ever exposes the model to within-puzzle mixing at all.

Figure 3 places all four conditions side by side: wide, trial-dependent spread under never-mixed training (the two leftmost columns), collapsing to a tight cluster near the ceiling the moment training always mixes alphabets within a puzzle (the two rightmost columns), regardless of whether the pool behind that mixing has $K = 6$ or $K = 25$ alphabets.

The main text omits this condition's zero-shot transfer to held-out alphabet E as not necessary to the main argument; we report it here for completeness. It reaches 11.46% cell accuracy and 0% exact match rate, close to the roughly 11.1% chance floor and the weakest of the three zero-shot cell-accuracy numbers reported in this paper (versus 14.47% for Section 4.3's fixed-alphabet pointer and 15.85% for Section 4.4's 25-letter random mix). Exact match rate is 0% in all three conditions regardless, so the paper's headline conclusion, that transfer to a truly novel symbol is unsolved, is unaffected either way; whether this specific ordering of cell-accuracy numbers reflects a real dependence on alphabet-pool diversity or simply run-to-run noise cannot be settled with a single seed per condition (Section 6).

A.5 COMPUTE

All experiments ran on Leonardo (CINECA EuroHPC), a single NVIDIA A100 GPU per job. For Section 4.1's single- and union-alphabet training, we have measured wall-clock time directly: 52 minutes each for the two single-alphabet runs and 1 hour 43 minutes for the union run (≈ 3.5 GPU-hours together). For every other job, Table 13 reports the wall-clock budget requested at submission time instead of measured usage; where we do have measured numbers (Section 4.1), actual usage came in well under the requested budget, so the totals below should be read as an upper bound on total compute, not an exact accounting.

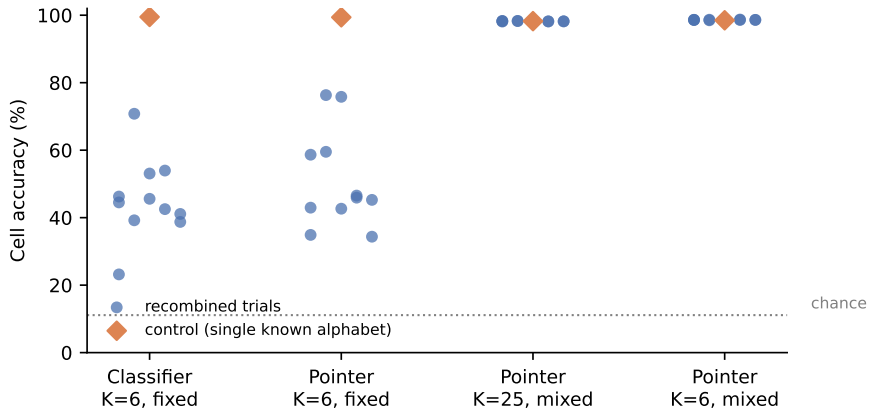


Figure 3: Per-trial cell accuracy on the mixed-alphabet compositional test (Tables 2, 3, 6, 11), across the four conditions that were evaluated on it. Each point is one of the 11 recombination trials (the maximally mixed cycle plus 10 random draws); diamonds mark the coherent single-alphabet control. Training that never mixes alphabets within a puzzle (the two leftmost conditions) leaves accuracy highly sensitive to which specific recombination is tested; training that always mixes them (the two rightmost conditions) collapses that variance almost entirely, regardless of how many letters the training pool contains.

Job	Requested wall-clock
Data generation (one-time)	1h00m
Single-alphabet training $\times 2$, union training (Section 4.1)	2h00m + 2h00m + 3h00m
Cross-alphabet evaluation	0h30m
$K{=}6$ classifier pretraining (Section 4.2)	2h00m
Few-shot finetune/scratch sweep	1h30m
Mixed-alphabet evaluation	0h20m
$K{=}6$ pointer pretraining (Section 4.3)	2h00m
Pointer zero-shot and mixed-alphabet evaluation	0h30m + 0h20m
$K{=}25$ random-mix pointer pretraining (Section 4.4)	1h30m
Random-mix zero-shot and mixed-alphabet evaluation	0h30m + 0h20m
$K{=}6$ always-mixed pointer pretraining (Section 4.5)	1h30m
$K{=}6$ always-mixed zero-shot and mixed-alphabet evaluation	0h30m + 0h20m
Total requested (upper bound)	$\approx$ 20 GPU-hours

Table 13: Requested wall-clock per SLURM job, one A100 GPU each. Measured usage where available (Section 4.1) came in roughly half of the requested budget.