Modular Counts, Smarter Programs: Enhancing Transformer Program Generalization with Structured Biases

Arjun Menon
Princeton University
armenon@princeton.edu

David Beloglazov
Princeton University
beloglazov@princeton.edu

Abstract

Transformer Programs offer a promising framework for learning interpretable algorithmic behavior from data. However, they often struggle to generalize when input sequences grow in length or vocabulary size. We investigate whether simple, modular inductive biases enhance robustness in such settings.

We propose two plug-in architectural augmentations for the Transformer Program framework: *PrefixSumCounts*, which adds per-token cumulative frequency features, and the *SparseExpertCountingNetwork*, a learned mixture-of-experts layer that dynamically selects among interpretable counting functions. These modules are integrated into the numerical input stream without modifying the core architecture or loss. Across five compositional reasoning tasks and three generalization regimes, we evaluate all combinations of these modules and their ablations.

We find that PrefixSumCounts consistently improves generalization on tasks involving frequency, repetition, and ordering, while the expert module offers complementary gains on more complex counting tasks. Together, the modules match or exceed baseline performance in challenging settings (e.g., 98.1% vs. 50.9% on `double_hist` with large vocab and sequence length). Our findings suggest that lightweight, interpretable inductive biases can substantially enhance the generalization and modularity of Transformer Programs.

1 Introduction

Neural networks have long struggled to exhibit both high performance and interpretability on algorithmic tasks. Recent advances in modular architectures aim to reconcile these goals by learning structured computation that maps cleanly to symbolic operations. Among these efforts, *Transformer Programs* (Friedman et al., 2023) stand out as a promising framework that trains Transformer-based architectures to synthesize human-readable programs from data. This is achieved by treating layers, heads, and MLPs as modular components and encouraging them to emulate compositional, discrete functions through a differentiable relaxation.

However, despite their clean abstraction and strong performance on synthetic tasks, Transformer Programs exhibit a critical limitation: they generalize poorly when input complexity scales. When evaluated on longer sequences or larger vocabularies, accuracy frequently collapses, undermining claims of modular generalization. This limitation suggests that while the architecture supports interpretability, it lacks inductive biases that promote robustness under distribution shift.

In this work, we ask whether lightweight architectural augmentations can improve generalization in Transformer Programs without compromising their interpretability or structure. We introduce two plug-in modules:

- **PrefixSumCounts**: a differentiable, stateless operation that injects per-token cumulative frequency into the model's numerical input stream.
- **SparseExpertCountingNetwork**: a learned mixture-of-experts layer over histogram features that dynamically selects among interpretable counting functions such as total count, frequency, and uniqueness.

These additions require no changes to the core model architecture, optimization procedure, or supervision signal. Instead, they act as inductive biases, steering the model toward representations that abstract count-based reasoning—a known bottleneck for generalization in algorithmic domains.

To evaluate their effectiveness, we conduct a comprehensive empirical study across five compositional reasoning tasks from Friedman et al. (2023). For each task, we benchmark three generalization regimes: doubling sequence length, doubling vocabulary size, and doubling both. We systematically test all combinations of the proposed modules and compare them to the original baseline. Our results show that:

- PrefixSumCounts reliably improves performance across tasks involving frequency, ordering, and repetition.
- The expert module offers complementary gains on tasks with more complex count-based structure.
- Together, the modules recover or exceed baseline accuracy under distribution shift.

These findings suggest that simple, interpretable architectural biases can meaningfully enhance generalization in structured Transformer-based models. Our approach preserves the modular, analyzable nature of Transformer Programs while enhancing their robustness in compositional reasoning tasks.[1]

2 Related Work

2.1 Neural Program Induction

A long-standing goal in machine learning is to induce structured programs from data. Early work focused on recurrent architectures trained to simulate algorithms (Reed and De Freitas, 2015; Cai et al., 2017), but these models often lacked interpretability and generalization. More recent efforts have leveraged Transformer-based architectures for symbolic reasoning, with varying levels of modularity and transparency (Kim, 2021; Weiss et al., 2021). In particular, Friedman et al. (2023) introduce *Transformer Programs*, a framework that treats attention heads and MLPs as symbolic units trained to represent discrete functions. While this design improves interpretability, the model still struggles to generalize to larger input scales.

2.2 Modular and Interpretable Architectures

The interpretability community has explored modular networks that align internal components with human-interpretable concepts (Chen et al., 2019; Rudin, 2019). In NLP, architectures like Neural Module Networks (Andreas et al., 2016) and Neurosymbolic Transformers (Inala et al., 2020) decompose tasks into functional subroutines to support compositional generalization. Our approach is similarly modular but focuses on lightweight augmentations that steer learning toward interpretable counting functions—without task-specific architectural design or supervision.

2.3 Mixture-of-Experts and Sparse Computation

Sparsely activated networks have improved efficiency and performance in large-scale models (Shazeer et al., 2017), typically by routing input to expert subnetworks. These methods primarily emphasize scalability rather than interpretability. Our *SparseExpertCountingNetwork* adapts this idea to small, interpretable settings by restricting experts to a set of canonical count functions and enabling analysis of expert activation post hoc.

2.4 Inductive Bias in Algorithmic Tasks

Structured inductive biases have been shown to improve generalization in symbolic reasoning and algorithmic tasks, especially under distribution shifts (Goyal and Bengio, 2021; Hahn, 2020). For example, Cranmer et al. (2020) and Wang et al. (2021) demonstrate that incorporating symbolic structure enhances robustness in physics and classification settings. We build on this line of work by targeting counting and frequency reasoning—recurring bottlenecks in Transformer Programs—with direct architectural support.

3 Methodology

3.1 Overview of Transformer Programs

The Transformer Programs framework introduced by Friedman et al. (2023) recasts the standard Transformer architecture as a modular program synthesis system. The goal is to learn interpretable algorithmic behaviors by constraining attention heads, MLPs, and decoding procedures to align with discrete, compositional functions. The model is trained end-to-end to solve reasoning tasks, but with architectural and objective modifications that promote symbolic modularity.

Each layer is split into two parallel streams: a **categorical stream**, which represents symbolic variables using one-hot vectors over a learned vocabulary, and a **numerical stream**, which holds continuous auxiliary features such as token frequency or position. The categorical stream is propagated using *discrete Gumbel-softmax sampling* (Jang et al., 2017), allowing for differentiable approximations of hard variable selection. The nu-

[1]Code: `https://github.com/A-Menon/TransformerProgramsExtended`

merical stream is updated using standard MLPs and attention.

Attention heads act as discrete memory reads: each head selects one key and retrieves its corresponding value, producing a symbolic output that is passed to the next layer. MLPs are applied independently per token and act as learnable transformation functions on categorical variables. These MLPs operate by selecting from a predefined output vocabulary, again using Gumbel-softmax sampling to maintain modularity.

At the final layer, an output head decodes the symbolic trace into a sequence of human-readable Python-like pseudo-code via symbolic regression. This is enabled by a learned dictionary that maps internal representations to code tokens and function calls. Notably, no intermediate supervision is used; the model is trained only on task inputs and outputs, with symbolic structure encouraged by architecture design and loss annealing.

To ensure interpretability, the model enforces architectural separation between reading (attention) and transformation (MLPs), and restricts inter-stream interactions. Figure 1 (adapted from the original paper) shows the full computation graph and decoding pipeline. This modular setup allows for direct inspection of layer-wise behavior and provides a natural insertion point for our inductive bias extensions.

3.2 PrefixSumCounts

3.2.1 Motivation

Several tasks in the Transformer Programs benchmark suite rely on counting behavior—for example, predicting token histograms (`hist`), identifying the most frequent element (`most_freq`), or combining multiple histogram statistics (`double_hist`). In the original framework, such information must be inferred implicitly via self-attention and MLP interactions. However, generalization performance degrades substantially on these tasks when sequence length or vocabulary size increases, suggesting that the model does not robustly internalize count-based abstractions.

3.2.2 Method Description

We introduce *PrefixSumCounts*, a non-parametric augmentation to the model's numerical input stream. The module computes, for each token position, a cumulative count vector over all prior tokens in the sequence. Let $x = (x_1, x_2, \ldots, x_n)$ be the input sequence with vocabulary size K. We define a prefix frequency matrix $P \in R^{n \times K}$ as:

$$P_{t,k} = \sum_{i=1}^{t-1} 1[x_i = k]$$

where $P_{t,k}$ denotes how many times token k has occurred before position t. Each vector P_t is appended to the token's numerical feature vector before being passed to the model.

3.2.3 Integration into Transformer Programs

Because P is computed directly from the input sequence, this module introduces no additional learnable parameters and nor does it modify the Transformer's base architecture, training objective, or loss function. It is implemented efficiently via a cumulative sum over one-hot token encodings, vectorized across the batch. Although this increases the dimensionality of the numerical input stream by K, it preserves training stability due to its stateless nature.

Unlike absolute counts (already present in the original model), prefix-aware counts encode position-sensitive frequency information, enabling one-pass histogram estimation and frequency-based decision rules. This inductive bias provides the model with an explicit mechanism to reason about token repetition and ordering.

3.2.4 Empirical Impact

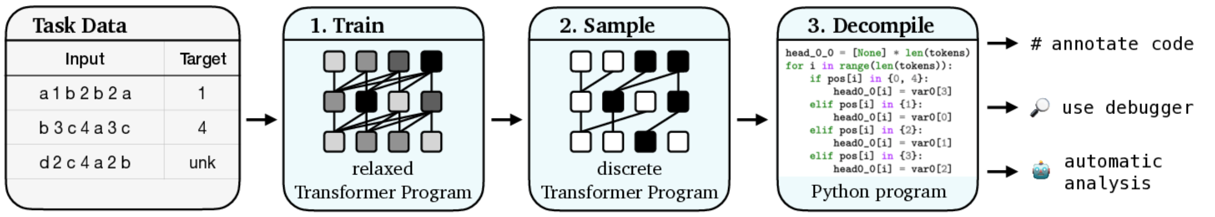
As described in **Results**, the inclusion of PrefixSumCounts improves performance on frequency-sensitive tasks in nearly all generalization settings, particularly under longer sequence lengths. It serves as a minimal but effective addition to the Transformer Programs setup, addressing a concrete failure mode without altering the core architecture.

3.3 SparseExpertCountingNetwork

3.3.1 Motivation and Context

While the PrefixSumCounts module injects cumulative frequency information, certain tasks in the Transformer Programs suite demand more complex count-based reasoning. Examples include selecting tokens by global frequency (`most_freq`) or reasoning over joint statistics (`double_hist`). These require a more expressive, task-adaptive abstraction over histogram-level features—beyond what PrefixSumCounts provides.

In preliminary experiments, we attempted to learn such abstractions using deeper MLPs on the numerical stream and by manually concatenating



Input	Target
a 1 b 2 b 2 a	1
b 3 c 4 a 3 c	4
d 2 c 4 a 2 b	unk

```
head_0_0 = [None] * len(tokens)
for i in range(len(tokens)):
    if pos[i] in {0, 4}:
        head0_0[i] = var0[3]
    elif pos[i] in {1}:
        head0_0[i] = var0[0]
    elif pos[i] in {2}:
        head0_0[i] = var0[1]
    elif pos[i] in {3}:
        head0_0[i] = var0[2]
```

Figure 1: Transformer Programs architecture. Layers are modular and symbolic: attention reads memory, MLPs transform variables, and outputs are decoded into interpretable programs.

global histogram statistics as inputs. However, these approaches either lacked sufficient task coverage or disrupted modular behavior. To address this, we introduce the *SparseExpertCountingNetwork*, a small mixture-of-experts (MoE) layer that computes and dynamically selects among a fixed set of interpretable histogram functions. Unlike typical MoE layers that prioritize scale, our goal is to provide interpretable, task-relevant count transformations under constrained capacity.

3.3.2 Architecture and Routing Mechanism

The SparseExpertCountingNetwork operates on a per-token histogram feature vector $h_t \in R^K$, which is computed by counting occurrences of each token in the full input sequence $x = (x_1, \ldots, x_n)$:

$$h_{t,k} = \sum_{i=1}^{n} 1[x_i = k]$$

This histogram is identical for all token positions in a sequence and is concatenated to the existing numerical input stream.

Let $\mathcal{E} = \{E_1, \ldots, E_m\}$ be a fixed set of m interpretable expert functions, each mapping h_t to a scalar statistic relevant for counting-based decisions. In our implementation, we use:

- $E_1(h_t)$: Total count (i.e., $\sum_k h_{t,k}$)
- $E_2(h_t)$: Token-specific count h_{t,x_t}
- $E_3(h_t)$: Binary indicator for most frequent token
- $E_4(h_t)$: Uniqueness indicator (i.e., token seen only once)
- $E_5(h_t)$: Relative frequency (i.e., $h_{t,x_t}/n$)

Each expert E_i produces a single output per token. These outputs are stacked into a vector $z_t \in R^m$, which is then routed via a learned gating network:

$$\alpha_t = \text{GumbelSoftmax}(Wh_t + b; \tau)$$

where $W \in R^{m \times K}$ and $b \in R^m$ are learnable parameters, and τ is the temperature. The final expert output is the weighted sum:

$$y_t = \sum_{i=1}^{m} \alpha_{t,i} \cdot E_i(h_t)$$

This scalar y_t is appended to the token's numerical input stream, augmenting the model with a context-aware, interpretable counting signal.

3.3.3 Design Choices and Alternatives Considered

We explored several alternative designs before settling on the current structure:

- **Token-wise histogram computation:** Initially, we attempted to build position-specific histograms h_t, but these were redundant and increased computational overhead. Global histograms proved more stable and sufficient.
- **Soft attention routing:** Using softmax instead of Gumbel-softmax resulted in vague expert allocation and degraded interpretability. The Gumbel formulation sharpened selection and encouraged module specialization.
- **Trainable experts:** While we experimented with using MLPs as expert functions, the outputs were harder to interpret and did not yield better performance than fixed counting functions.

Ultimately, the chosen design emphasizes interpretability and minimal parameter overhead. Each expert corresponds to a known statistic and the gating mechanism enables dynamic adaptation to the task without introducing architectural opacity.

3.3.4 Integration into Transformer Programs

The SparseExpertCountingNetwork is implemented as a preprocessing layer that augments each token's numerical feature vector with y_t. It requires no modifications to the core model architecture or decoding process. Because the expert set is fixed and sparse, the overhead is minimal. Empirically, we find that adding this module improves performance on structurally complex tasks that involve reasoning over token sets rather than local sequences.

As shown in **Results**, the expert module yields notable improvements on tasks like `most_freq` and `double_hist`, especially under generalization regimes with larger vocabularies. It complements PrefixSumCounts by offering a higher-level abstraction for histogram-based reasoning.

4 Experimental Setup

4.1 Tasks and Dataset Variants

We evaluate our proposed modules using the benchmark suite introduced in Friedman et al. (2023), which includes five synthetic reasoning tasks designed to assess compositional generalization and interpretability:

- `hist`: compute a histogram of token frequencies.
- `most_freq`: identify the most frequent token.
- `reverse`: reverse the input sequence.
- `sort`: sort tokens in ascending order.
- `double_hist`: compute and concatenate two histograms over disjoint input slices.

These tasks vary in complexity and span a spectrum of compositional behaviors, including frequency-based lookup, sorting, and aggregation. All datasets are procedurally generated at training time using fixed seeds, ensuring full control over distributional structure and allowing reproducible comparisons across configurations.

To evaluate robustness under distribution shift, we define three generalization regimes:

1. **Longer Inputs**: maximum sequence length is doubled from 8 to 16.
2. **Larger Vocabulary**: vocabulary size is doubled from 8 to 16.
3. **Both**: both sequence length and vocabulary size are doubled.

4.2 Model Configurations and Ablations

Each model configuration consists of a standard Transformer Programs model with one of four module combinations:

- `None`: the original baseline without inductive biases.
- `PrefixSumCounts`: appends a cumulative count vector to each token's numerical input.
- `SparseExpertCountingNetwork`: applies expert routing over histogram-derived features.
- `Both`: includes both modules simultaneously.

Tokens are represented categorically via one-hot encodings and numerically via auxiliary features. In all augmented configurations, PrefixSumCounts is concatenated to the numerical input stream before the first layer. The expert module is applied after the first MLP layer as a residual feature refinement step. These modules are designed to be architecture-agnostic, preserving the base model's symbolic decoding and layer separation.

4.3 Training Details and Hyperparameters

All models are trained using the original optimization setup from Friedman et al. (2023). Training is supervised end-to-end using Adam (Kingma and Ba, 2014) with the following hyperparameters fixed across all tasks and variants:

- Dataset size: 20,000
- Batch size: 512
- Epochs: 250
- Learning rate: 0.02
- Gumbel-softmax annealing: $\tau = 3.0 \to 0.1$ (geometric schedule)

All models are initialized with the same random seed for controlled comparisons. Input-output pairs are freshly generated at each epoch, mitigating overfitting to specific samples.

Model width (embedding dimension) and depth (number of layers) are scaled proportionally under generalization regimes to preserve capacity and gradient flow:

- `baseline`: $|V| = 8$, $N = 8$
- `longlen`: $|V| = 8$, $N = 16$
- `bigvocab`: $|V| = 16$, $N = 16$

4.4 Evaluation Metrics

Evaluation is conducted zero-shot on out-of-distribution test sets corresponding to each generalization regime. Models are not fine-tuned or adapted to the new distributions.

Performance is measured using exact-match sequence accuracy on 2,000 held-out examples per task-regime pair. For each configuration, we additionally track loss curves and (where applicable) expert gate distributions to validate specialization behavior. All experiments are run using the same infrastructure and logging scripts to ensure consistency.

5 Results

We evaluate accuracy across all tasks and generalization regimes, comparing the base Transformer Programs model against variants augmented with *PrefixSumCounts*, *SparseExpertCountingNetwork*, and both combined.

5.1 Overall Performance

Table 1 summarizes test accuracy. While all models achieve near-perfect performance in the base regime ($|V| = 8$, $N = 8$), generalization varies widely.

PrefixSumCounts yields the most consistent improvements, particularly on counting-heavy tasks. On `double_hist` with large vocab and long sequences ($|V| = 16$, $N = 16$), accuracy improves from 50.89% (base) to 98.14%. The module also improves `most_freq` (57.13% vs. 54.05%) and `sort` (81.12% vs. 79.70%) under this hardest regime, and generally avoids regressions.

SparseExpertCountingNetwork performs well on ordering-based tasks. On `sort`, it achieves the highest accuracy in two out of three regimes—up to 84.71% at $|V| = 16$, $N = 16$—and also boosts `hist` to 99.96% at the same setting. However, it underperforms on `double_hist` and `most_freq`, where fine-grained count tracking is critical.

Combined modules offer complementary improvements but do not always outperform individual modules. They achieve the best result on `double_hist` under long sequence length ($|V| = 8$, $N = 16$), improving accuracy from 64.48% (base) to 87.78%. In other regimes, however, combining modules sometimes reduces accuracy—e.g., on `double_hist` with base inputs, where performance drops from 99.88% (base) to 82.17%.

5.2 Ablation and Complementarity

Each module introduces distinct inductive structure. PrefixSumCounts provides strong gains on frequency-based tasks and demonstrates resilience across regimes. SparseExpertCountingNetwork improves performance selectively on tasks with implicit ordering and sorting structure.

Interestingly, combined modules do not always produce additive gains. On `sort`, for example, PrefixSumCounts and Expert models achieve 81.12% and 84.71% respectively at $|V| = 16$, $N = 16$, while the combined model drops to 79.66%. In contrast, for `double_hist` at $|V| = 8$, $N = 16$, the combination boosts accuracy by over +23 points relative to either module alone.

These results suggest the modules address distinct weaknesses in the base model but may interfere when both are active unless carefully tuned.

5.3 Visualization

Figures 2–4 compare model performance by task and regime for each augmentation. Figure 5 summarizes accuracy shifts across all 15 task-regime pairs. In 13 of 15 settings, the final model (best performing between PrefixSumCounts, SparseExpertCountingNetwork, or Combined) matches or exceeds base accuracy; the remaining 2 regress slightly but still retain the same approximate performance level.

6 Analysis and Discussion

6.1 When and Why Each Module Helps

The two proposed modules demonstrate distinct patterns of utility. *PrefixSumCounts* tends to benefit tasks involving cumulative recurrence, indicating that explicit cumulative count features alleviate the difficulty of learning such abstractions purely through self-attention. Its resilience across settings indicates that prefix-aware recurrence information is broadly useful, especially when token redundancy increases (e.g., with larger vocabularies).

In contrast, the *SparseExpertCountingNetwork* appears particularly well-suited for tasks with localized comparison logic. This suggests that its mixture-of-experts routing is well-suited for structural patterns where global token frequency is less relevant than relational structure among tokens.

6.2 Complementarity and Interference

Despite their individually useful properties, the two modules do not always combine synergistically. In

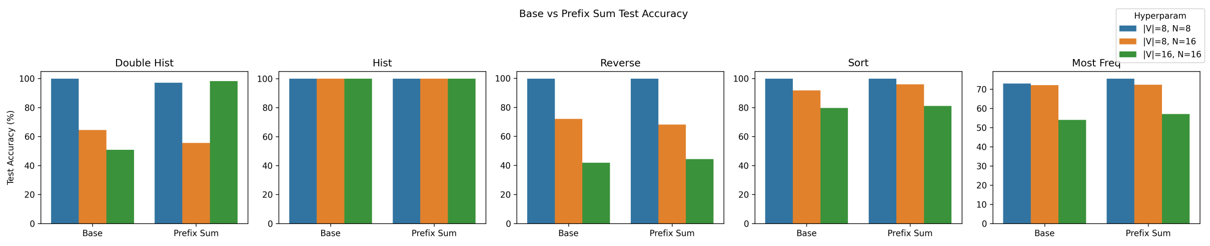


Figure 2: Base vs. PrefixSumCounts. Strong gains on frequency- and histogram-based tasks.

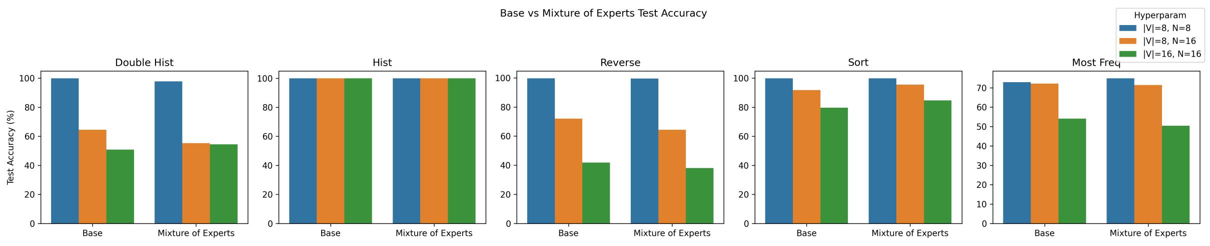


Figure 3: Base vs. Expert module. Improves sorting/generalization performance selectively.

some tasks (e.g., `double_hist` with long input), the combined model significantly outperforms either component alone, indicating successful integration of fine-grained and coarse-grained counting mechanisms.

However, in other regimes—such as `sort` at $|V| = 16$, $N = 16$—combined performance lags behind the better of the two single-module variants. This suggests that feature interference or overfitting may arise when both modules are active, especially under fixed capacity constraints. Future work could explore dynamic gating or conditional computation to mitigate such interference.

6.3 Expert Routing Behavior

To better understand the internal behavior of the expert module, we inspected the Gumbel-softmax gating outputs across evaluation runs. We observed that the learned router does not collapse to a single expert; rather, it consistently distributes probability mass across different experts depending on input characteristics.

For instance, in `hist`, routing often favors experts corresponding to total-count or category-specific count statistics. In `sort`, more emphasis is placed on experts resembling relative comparison functions. These patterns suggest that the expert network successfully captures useful subroutines, aligning with the modular programming interpretation that underpins Transformer Programs.

6.4 Interpretability and Structural Modularity

Both modules preserve the core interpretability ethos of Transformer Programs. *PrefixSumCounts* introduces no learnable parameters and provides semantically grounded numerical inputs. The expert module, while learned, restricts its basis functions to interpretable count abstractions, and the routing mechanism is inspectable at test time.

This design-level modularity enables post hoc analysis of model behavior and failure cases—unlike fully entangled neural architectures. Moreover, the separability of modules allows them to be reused or removed with minimal impact on the rest of the system, aligning with the design goals of composable, symbolic reasoning models.

6.5 Limitations and Future Directions

While our improvements address key failure modes in Transformer Programs, several limitations remain. The combined module occasionally regresses due to uncoordinated interaction effects. Gating behavior, while interpretable, may be unstable under minor distribution shifts. Additionally, our modules were designed for synthetic tasks; their generality on real-world algorithmic or program synthesis problems remains an open question.

Future work could explore adaptive capacity allocation between modules, soft parameter sharing, or regularization techniques to encourage complementary specialization. Extending this framework to hierarchical or multi-step reasoning tasks may also

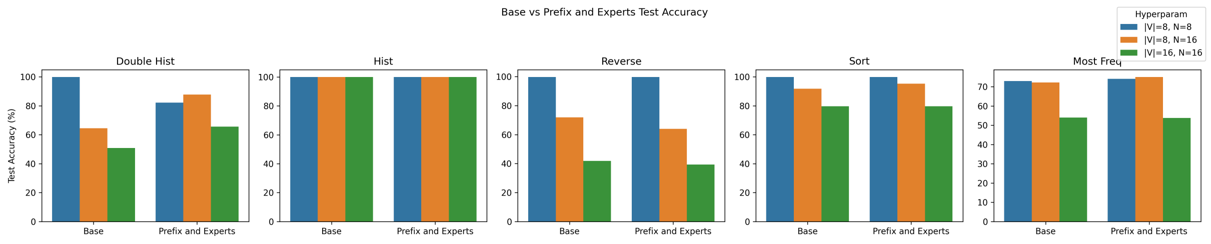


Figure 4: Base vs. Combined modules. Helpful on specific tasks, but not universally dominant.

Task	Hyperparam	Base	PrefixSumCounts	Experts	Prefix + Experts	Improved
`double_hist`	$\|V\| = 8, N = 8$	**99.88**	97.02	97.76	82.17	97.76
`double_hist`	$\|V\| = 8, N = 16$	64.48	55.56	55.23	**87.78**	**87.78**
`double_hist`	$\|V\| = 16, N = 16$	50.89	**98.14**	54.41	65.64	**98.14**
`hist`	$\|V\| = 8, N = 8$	**99.95**	**99.95**	**99.95**	**99.95**	**99.95**
`hist`	$\|V\| = 8, N = 16$	99.92	99.93	99.94	**100.00**	**100.00**
`hist`	$\|V\| = 16, N = 16$	99.94	**100.00**	99.96	**100.00**	**100.00**
`reverse`	$\|V\| = 8, N = 8$	99.71	99.63	99.56	**99.80**	**99.80**
`reverse`	$\|V\| = 8, N = 16$	**72.00**	68.19	64.34	64.01	68.19
`reverse`	$\|V\| = 16, N = 16$	41.83	**44.28**	38.03	39.39	**44.28**
`sort`	$\|V\| = 8, N = 8$	99.90	99.84	**99.93**	99.89	**99.93**
`sort`	$\|V\| = 8, N = 16$	91.82	**95.94**	95.65	95.31	**95.94**
`sort`	$\|V\| = 16, N = 16$	79.70	81.12	**84.71**	79.66	**84.71**
`most_freq`	$\|V\| = 8, N = 8$	72.95	**75.54**	74.90	74.11	**75.54**
`most_freq`	$\|V\| = 8, N = 16$	72.20	72.36	71.44	**75.05**	**75.05**
`most_freq`	$\|V\| = 16, N = 16$	54.05	**57.13**	50.49	53.79	**57.13**

Table 1: Test accuracy (%) across tasks and generalization regimes. Best results per row are in bold.

reveal deeper synergies between discrete program structure and continuous neural representation.

7 Conclusion

Transformer Programs represent a compelling framework for learning symbolic, interpretable algorithms from data, but their generalization under distribution shift remains limited. In this work, we introduced two modular augmentations—*PrefixSumCounts* and the *SparseExpertCountingNetwork*—that introduce inductive biases tailored to counting and histogram abstraction, two known bottlenecks in algorithmic reasoning tasks.

Both modules integrate seamlessly into the existing Transformer Programs architecture without altering its core structure or supervision signal. Across five synthetic reasoning tasks and multiple generalization regimes, we demonstrated that these augmentations significantly improve accuracy under sequence length and vocabulary expansion, particularly on frequency-sensitive and sorting-oriented problems.

Our results highlight the utility of lightweight, interpretable architectural biases in bridging the gap between symbolic reasoning and neural generalization. Moreover, our ablation studies reveal complementary strengths and potential interference between modules, motivating future work in adaptive integration and capacity allocation.

Ultimately, this work supports the broader thesis that simple, structured modifications can substantially enhance the robustness and modularity of program-inducing neural networks. We hope these contributions inform future designs of compositional architectures for interpretable and generalizable algorithmic learning.

8 Acknowledgements

We would like to thank Professor Danqi Chen, Professor Tri Dao, and Professor Vikram Ramaswamy for the learning opportunities they offered us during and prior to this project. We also extend our gratitude to Adithya Bhaskar, David Braun, Catherine Cheng, Simon Park, Colin Wang, and Tyler Zhu for their support with our work.

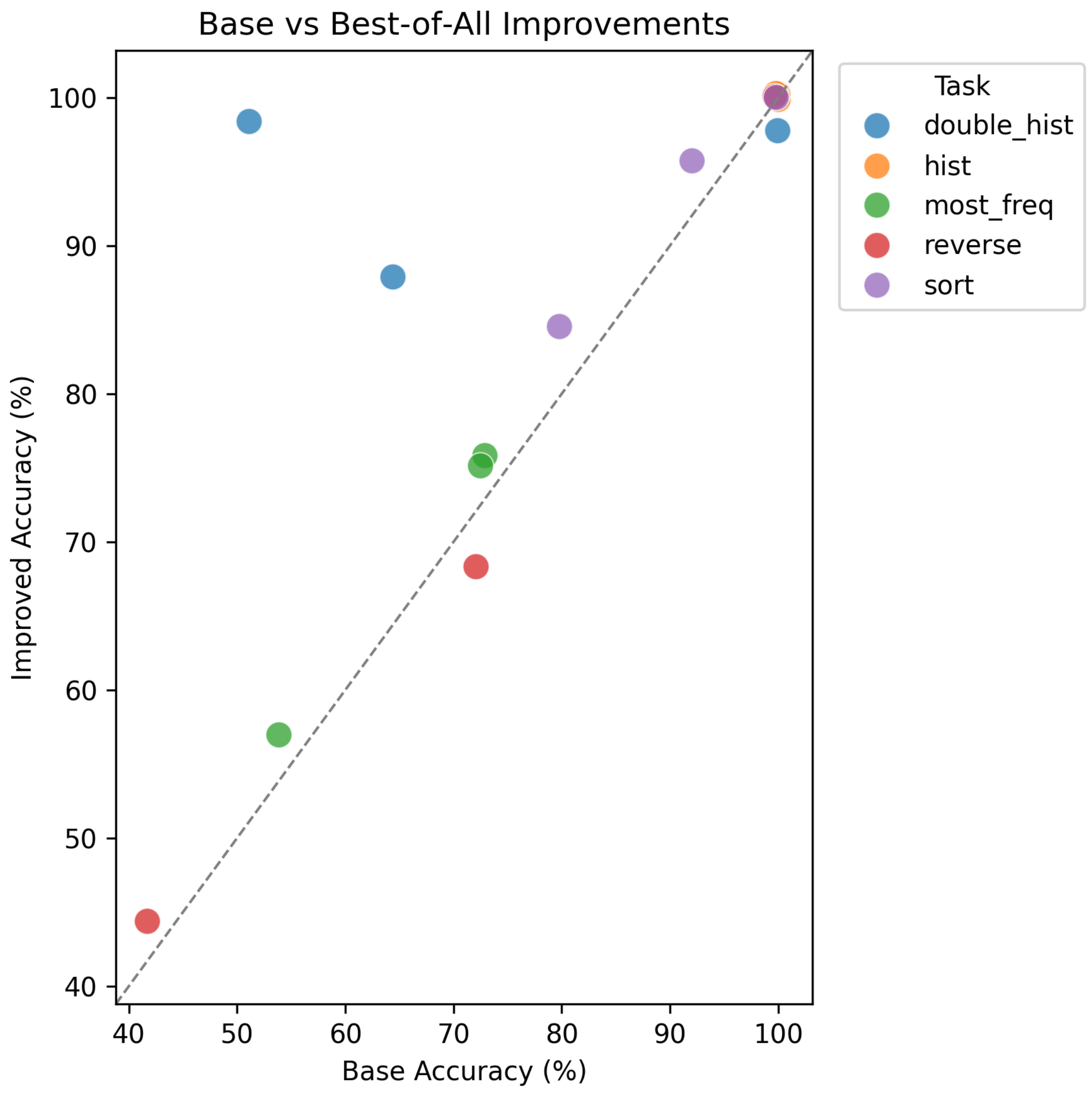


Figure 5: Scatterplot of Base vs. Improved accuracy across all settings. Majority lie above $y = x$.

References

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

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. 2021. An interpretability illusion for bert. *arXiv preprint arXiv:2104.07143*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901.

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

Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K. Su. 2019. This looks like that: Deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and

Christopher D. Manning. 2019. What does bert look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP*, pages 276–286.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering symbolic models from deep learning with inductive biases. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 17429–17442.

Andrew Cropper and Sebastijan Dumančić. 2022. Inductive logic programming at 30: A new introduction. *Journal of Artificial Intelligence Research*, 74:765–850.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer ElShowk, Nicholas Joseph, Nova DasSarma, Ben Mann, et al. 2022. Softmax linear units. *Transformer Circuits Thread*.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Thomas Conerly, et al. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Dan Friedman, Alexander Wettig, and Danqi Chen. 2023. Learning transformer programs. In *Advances in Neural Information Processing Systems 36*.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. 2023. Finding alignments between interpretable causal variables and distributed neural representations. *arXiv preprint arXiv:2303.02536*.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*.

Angeliki Giannou, Shashank Rajput, Jiyong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. 2023. Looped transformers as programmable computers. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Anirudh Goyal and Yoshua Bengio. 2021. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266).

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810.

Jeevana Priya Inala, Yichen Yang, James Paulos, Yewen Pu, Osbert Bastani, Vijay Kumar, Martin Rinard, and Armando Solar-Lezama. 2020. Neurosymbolic transformers for multi-agent communication. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 13597–13608.

Henrik Jacobsson. 2005. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not explanation. In *Proceedings of NAACL-HLT*, pages 3543–3556.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 26302–26317.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. 2021. Datasets: A community library for natural language processing. In *Proceedings of EMNLP: System Demonstrations*, pages 175–184.

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. 2023. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. 2023. The hydra effect: Emergent self-repair in language model computations. *arXiv preprint arXiv:2307.15771*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 17359–17372.

William Merrill and Ashish Sabharwal. 2022. Transformers implement first-order logic with majority quantifiers. *arXiv preprint arXiv:2210.02671*.

William Merrill, Ashish Sabharwal, and Noah A. Smith. 2022. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856.

Stephen Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679.

Hiroki Nakayama. 2018. Seqeval: A framework for sequence labeling evaluation. `https://github.com/chakki-works/seqeval`. Accessed: 2025-05-11.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*.

Nostalgebraist. 2020. Interpreting GPT: The logit lens. `https://www.alignmentforum.org/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`. Accessed: 2025-05-11.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 271–278.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 115–124.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32.

Ali Payani and Faramarz Fekri. 2019. Learning algorithms via neural logic networks. *arXiv preprint arXiv:1904.01554*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. 2022. Deep differentiable logic gate networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations (ICLR)*.

Scott Reed and Nando De Freitas. 2015. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*.

Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4593–4601.

Erik Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of HLT-NAACL*, pages 142–147.

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

Ellen M. Voorhees and Dawn M. Tice. 2000. Building a question answering test collection. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 200–207.

Fulton Wang and Cynthia Rudin. 2015. Falling rule lists. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1013–1022.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: A circuit for indirect object identification in gpt-2 small. In *International Conference on Learning Representations (ICLR)*.

Qinglong Wang, Kaixuan Zhang, Alexander G. Ororbia, Xinyu Xing, Xue Liu, and C. Lee Giles. 2018. An empirical evaluation of rule extraction from recurrent neural networks. *Neural Computation*, 30(9):2568–2591.

Zhuo Wang, Wei Zhang, Ning Liu, and Jianyong Wang. 2021. Scalable rule-based representation learning for interpretable classification. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5247–5256.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like transformers. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 11080–11090.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 3770–3785.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2021. Sparse attention with linear units. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6507–6520.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 649–657.