Structure-Aware Graph Hypernetworks for Neural Program Synthesis

Wenhao Li[1] Yudong W. Xu[1] Elias B. Khalil[1,2,3] Scott Sanner[1,2]
[1]University of Toronto [2]Vector Institute for Artificial Intelligence [3]Scale AI Research Chair in Data-Driven Algorithms for Modern Supply Chains

ICLR

30-second takeaway

Setting. Many learning problems separate inputs into two roles: an *instance* x and a *task parameter* p that changes the task itself.

$$y = F_p(x).$$

Here x is the ordinary input, while p is a higher-level knob such as a task, environment, or program parameter.

Conventional monolithic learning

$$g_\theta(x, p) \to y$$

A single flat predictor can fit new inputs x for seen p, but often fails to generalize when p itself is **unseen**.

Our idea. Synthesize an executable *neural program* from the parameter:

$$p \xrightarrow{M_\phi} \theta_p, \qquad y = f_{\theta_p}(x).$$

A **symmetry-aware graph hypernetwork** makes this $p \mapsto \theta_p$ map much more stable and improves zero-shot generalization.

What is the learning problem?

Conventional monolithic learning

$$g_\theta(x, p) \to y$$

- Treats p like another feature

Evaluation protocol.

- **ID test:** new (x, y), seen p
- **Out-of-parameter test:** new (x, y), unseen p

Parameterized learning / neural program synthesis

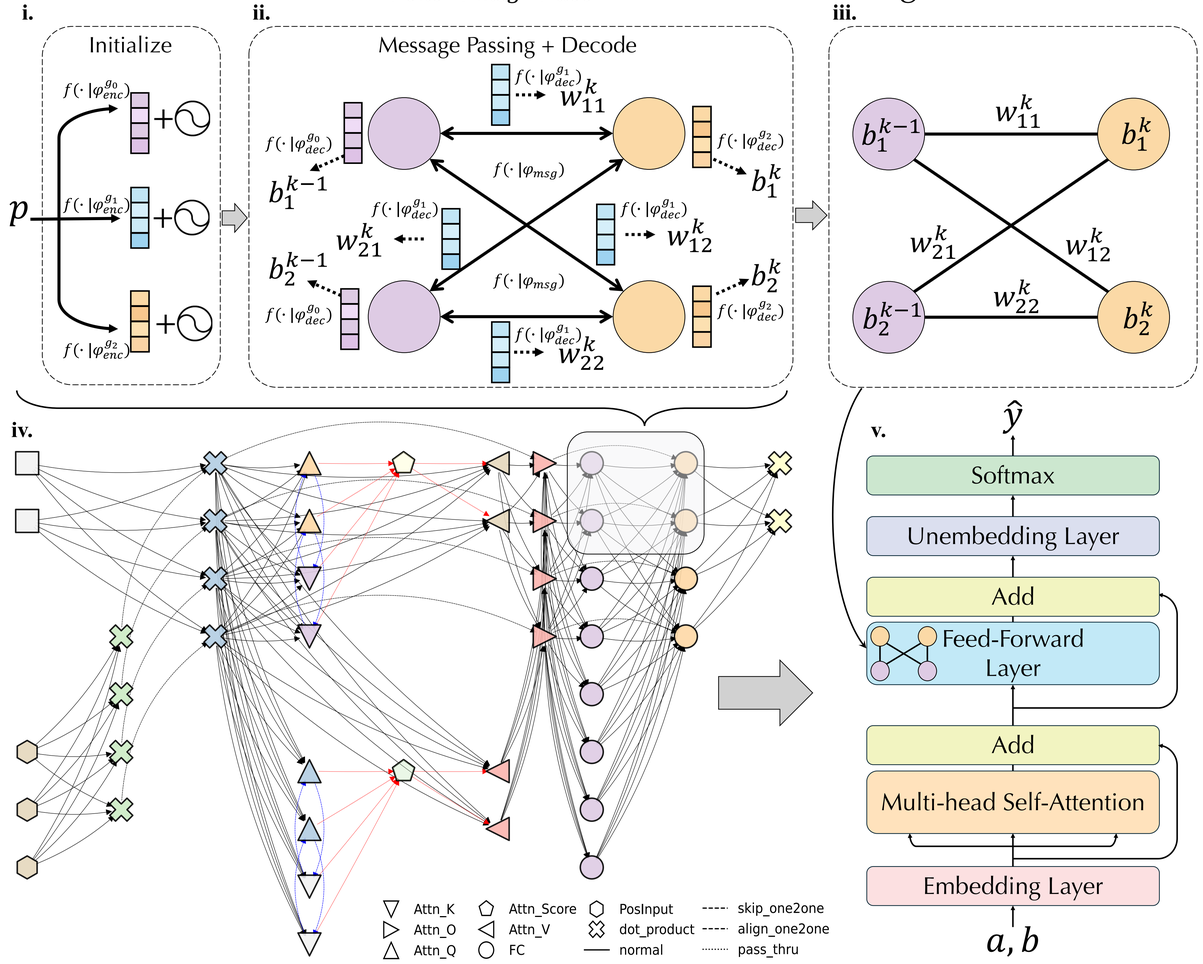
$$p \xrightarrow{M_\phi} \theta_p, \qquad y = f_{\theta_p}(x).$$

- M_ϕ is a hypernetwork, θ_p is the executable neural program

Why is learning $p \mapsto \theta_p$ hard?

- Many different weight vectors implement the same function.
- Hidden-unit / channel / attention-head **permutations** fragment supervision for classical hypernetworks.

Key idea. Represent the target network on its exact computation graph: **nodes** carry biases and **edges** carry weights. **Group-tied** encoder/decoder/message passing modules share parameters across permutation-equivalent parts.



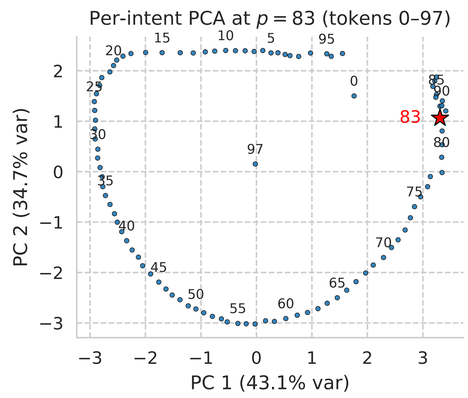
Benchmarks & Setup

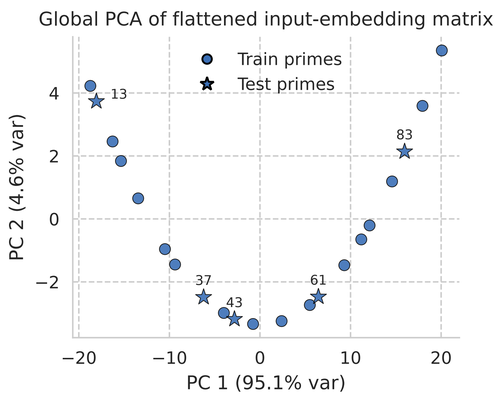
- SumFirst-N (MLP): $p = N$. Given $x = (x_1, \ldots, x_L)$, predict $\sum_{i=1}^{N} x_i$.
- AddMod-p (Transformer): p is the prime. Given a, b, predict $(a + b) \bmod p$.
- 1D-ARC (Seq2Seq): $\texttt{MoveRight}_p(\text{grid})$ shifts the bar p pixels to the right.

Results: the real separation appears on *unseen* p

Model	SumFirst		AddMod		1D-ARC	
	seen p	**unseen** p	seen p	**unseen** p	seen p	**unseen** p
Monolithic learning: $g(x, p) \to y$						
Direct MLP	1.000	0.140	0.916	0.699	–	–
Direct Transformer	–	–	0.850	0.682	0.996	0.603
Neural Program synthesis: $p \to \theta_p,\ y = f_{\theta_p}(x)$						
Meta-MLP	1.000	0.145	0.994	**0.947**	0.956	0.625
Meta-gMLP	0.996	0.344	0.923	0.883	0.942	0.610
Meta-GNN (ours)	1.000	**0.866**	0.992	0.943	0.978	**0.713**

Mechanistic evidence on AddMod-p: a smooth $p \mapsto \theta_p$ map





- For each prime p, the synthesized transformer recovers the classic *clock* algorithm.
- Across primes, the learned weights lie on a thin 1D trajectory in PCA space.
- This trajectory admits a **closed-form fit**, indicating a smooth rule $p \mapsto \theta_p$, not separate memorized solutions:

$$(\mathrm{PC}_1, \mathrm{PC}_2) = \left(23.4x,\ b + B\sum_{k=1}^{4}\binom{1/2}{k}(-1)^k x^{2k} + \sum_{j=0}^{3}\alpha_{2j+1}x^{2j+1}\right),\ x = \cos(5.7/p + 3.6),\ R^2 > 0.94.$$

chriswenhao.li@mail.utoronto.ca

https://github.com/chris91219/NeuroP