STRUCTURE-AWARE GRAPH HYPERNETWORKS FOR NEURAL PROGRAM SYNTHESIS

Wenhao Li[1] **Yudong Xu**[1] **Elias B. Khalil**[1] **Scott Sanner**[1,2]
[1]Department of Mechanical & Industrial Engineering, University of Toronto
[2]Vector Institute for Artificial Intelligence
chriswenhao.li@mail.utoronto.ca, wil.xu@mail.utoronto.ca,
elias.khalil@utoronto.ca, ssanner@mie.utoronto.ca

ABSTRACT

We study the neural program synthesis of *parameterized* function families through the lens of meta-learning with hypernetworks. Given a user intent U, a meta-learner M_ϕ produces a full weight set $\hat{\theta} = M_\phi(U)$ for a target neural network with fixed architecture S, and the instantiated network $m_{S,\hat{\theta}}(X) \to Y$ realizes the behavior intended for U. *Classical* hypernetworks typically *ignore the target network's structure* and emit a flat list of weights; as a consequence, they fail to account for *neuron-permutation symmetry*—many distinct parameterizations of S implement the same function—so equivalent solutions are treated as different targets, fragmenting supervision and hurting out-of-distribution generalization. To address this, we propose *Meta-GNN*, a hypernetwork that constructs a *neural graph* from the target architecture S and applies **structure-aware** message passing with parameter-tied encoders and decoders. This design reduces the search space during learning by collapsing equivalent classes of target networks, without loss of expressivity. Empirically, across modular arithmetic (ADDMOD-p), array operations (SUMFIRST-n), and inverse-rule tasks from 1D-ARC, *Meta-GNN* substantially improves learning and **out-of-distribution generalization** compared to classic hypernetworks and direct $(U, X) \to Y$ baselines. Mechanistic analyses reveal *what is learned*: on ADDMOD-p the synthesized Transformers recover the canonical clock representation and admit a compact closed-form map $U \mapsto \theta$. These results demonstrate that structure-aware Meta-GNNs enable reliable generalization to *unseen program parameterizations*, providing a critical advance for the nascent field of neural program synthesis.

1 INTRODUCTION

Program synthesis is traditionally framed as a combinatorial search over *symbolic* artifacts such as abstract syntax trees or domain-specific languages (DSLs) (Solar-Lezama, 2008; Alur et al., 2013). A user intent—given by I/O pairs, contracts, or natural language—must be realized by a token sequence that is both syntactically valid and semantically correct. This setting exposes two long-standing bottlenecks. **(i) Search:** navigating the discrete program space requires expensive enumeration, as even small edits can simultaneously affect syntax and semantics. **(ii) Verification:** correctness is a hard True/False property, offering no graded signal about how far a candidate program is from meeting the specification. These factors make scaling beyond small DSLs or short snippets extremely challenging.

We propose a continuous relaxation of program synthesis. A neural network with structure S and weights θ can be viewed as an executable program, so we define a *neural program* (**NeuroP**) as the pair (S, θ) (Goodfellow et al., 2016). To make synthesis tractable, we fix a template architecture $S^\star$ and learn to generate its parameters θ directly from a user intent U. In this setting, syntax is guaranteed by construction, while semantics—captured by $\mathrm{exec}(S^\star, \theta, X)$—are differentiable whenever a smooth loss can be defined. Neural programs are best understood as parameterized forward-pass equations that approximate the desired behavior (e.g., a classifier with high but not perfect accuracy). This perspective replaces the two classical bottlenecks: instead of navigating a discrete search space,

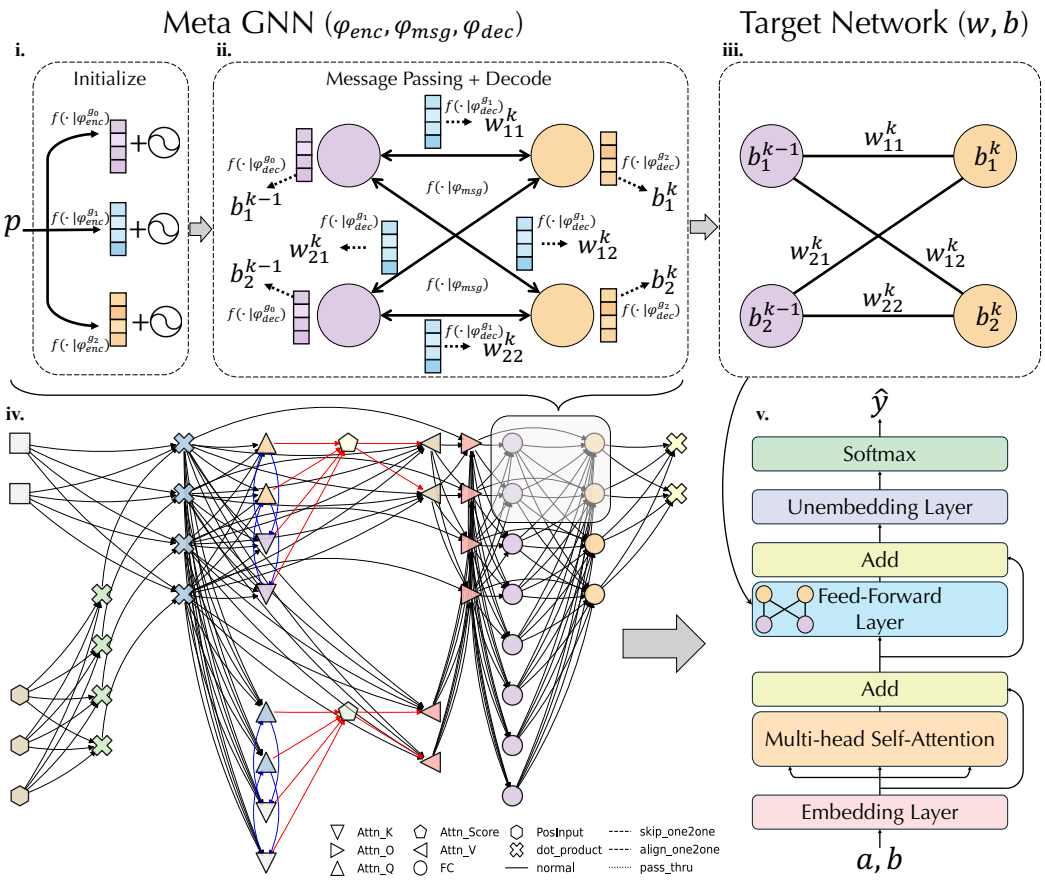


Figure 1: **Synthesizing a NeuroP.** We use ADDMOD–p to illustrate the pipeline. Given $p \geq 1$, the goal is to generate the weights of a one-layer Transformer (the *Target Network*) that computes $y = (a + b) \bmod p$ for any integers a, b. A Meta-GNN is trained on pairs $\{((p, a, b), y)\}$. **(i)** At each step the meta-learner receives only the user intent $U = \{p\}$. **(v)** The Transformer to be produced is first converted into a *neural graph* **(iv)** with nodes (biases) and edges (weights). As the meta-network, the only learnable parameters φ live in the encoders f_{enc}, message function f_{msg}, and decoders f_{dec}, all *tied within permutation-equivalent groups*. **(ii)** The Meta-GNN runs k rounds of message passing parameterized by f_{msg} with permutation-invariant aggregation, producing node/edge embeddings; **(iii)** group-tied decoders f_{dec} map these embeddings to scalar parameters that instantiate target network weights w_{ij}^k and biases b_i^k. The instantiated Transformer **(v)** then maps (a, b) to $\hat{y}$, and gradients of $\mathcal{L}(\hat{y}, y)$ backpropagate through **(v)**→**(iii)**→**(ii)**→**(i)** to update φ. We inject positional encodings only at initialization **(i)** to break symmetry; these codes are *not* used inside f_{enc}, f_{msg} or f_{dec}, so learning remains permutation-invariant. This preserves target network expressivity and reduces the effective search space, which empirically improves learning.

we optimize in continuous weight space; and instead of a hard True/False specification, we rely on graded losses that quantify semantic error. The trade-off is that we give up exact correctness guarantees, but in return enable gradient-based optimization and rich learning signals.

Within this framework, two central questions arise: (1) can complete neural programs be synthesized in one pass from intent alone, and (2) can the synthesizer learn a reliable mapping $U \to \theta$ that extrapolates to intents never seen during training? The latter is especially significant, as it amounts to true out-of-distribution generalization (OOD), a long-standing challenge in both program synthesis and machine learning. Under mild realizability and continuity assumptions, existence and stability results for intent-to-weights synthesis follow from standard extensions of universal approximation and weight-perturbation bounds; we provide formal statements in App. A. Given these existence results, it may seem that classical hypernetworks (Ha et al., 2016) may be able to address

target network program synthesis; unfortunately, such hypernetworks ignore the target network's structure and *neuron-permutation symmetry* (Navon et al., 2023; Zhou et al., 2024), where many weight assignments implement the same function, fragmenting supervision and hurting OOD generalization. To overcome this, we introduce a structure-aware Meta-GNN that decodes to the target network's computation graph (weights as edges, biases as nodes), uses DeepSet-style (Zaheer et al., 2017) parameter tying for the encoders and decoders within permutable neuron groups, and performs structure-aware message passing aligned with the graph's structure, thereby reducing the effective search space. We refer to Fig. 1 for an overview of the methodology and §4.3 for additional details.

Empirically, we evaluate on three classic modeling setups with standard loss functions—an MLP trained with MSE for prefix summation (SUMFIRST-n), a Transformer trained with cross-entropy for modular addition (ADDMOD-p), and a seq2seq Transformer for mixtures of program families in 1D-ARC (Xu et al., 2024). This design highlights that our pipeline is architecture- and loss-agnostic, while the 1D-ARC setting in particular provides a more realistic testbed with diverse tasks. Across all tasks, meta-learners in weight space substantially outperform direct $(U, X) \to Y$ baselines, which mimic the conventional supervised learning setup. Among our models, the structure-aware Meta-GNN achieves the strongest and most reliable generalization to unseen intents. On ADDMOD-p, it not only recovers the modular "clock" algorithm but also allows us to empirically extract the learned mapping $U \to \theta$, providing direct evidence that the meta-learner captures the underlying algorithm rather than memorizing instances.

In summary, our contributions are threefold. First, we establish **NeuroP** as a program modality that trades exact symbolic semantics for continuous, differentiable representations, enabling gradient-based optimization in place of enumeration-based search. Second, we propose a structure-aware graph hypernetwork (Meta-GNN) that represents the target architecture as a neural graph, ties encoders/decoders across permutation-equivalent neurons, and generates the *entire* weight set in a single forward pass. We further provide empirical evidence that intent-to-weight synthesis is feasible, and structure-aware meta-learners achieve strong OOD generalization across diverse program families. Third, we release complete code and datasets to support reproducibility and future research (see App. F).

2 RELATED WORK

Symbolic and classical program synthesis. Traditional synthesis operates over *discrete* artefacts such as abstract syntax trees or DSLs, using constraint solving to find programs that satisfy user intent. Pioneering systems like SKETCH (Solar-Lezama, 2013) and SyGuS (Alur et al., 2013) frame the task as search within a template space, yielding strong correctness guarantees but facing combinatorial blow-ups as the template grows. Subsequent work improves scalability through smarter search: CEGIS (Gulwani et al., 2017) iteratively alternates between candidate generation and counterexample finding; cvc4sy (Reynolds et al., 2019) combines SMT enumeration with divide-and-conquer; and hybrid approaches integrate learning-based heuristics or symmetry reduction to prune the search (Huang et al., 2020; Abate et al., 2023; Kim et al., 2021). Yet all remain tethered to discrete syntax and handcrafted templates. By contrast, we recast synthesis as continuous optimization in weight space: a fixed neural template guarantees syntactic validity, while differentiability enables efficient gradient-driven search without explicit symbolic enumeration.

Neuro-symbolic approaches. Neuro-symbolic methods blend discrete program representations with differentiable neural modules, aiming to merge symbolic structure with the adaptability of learning-based models. Techniques such as Neuro-Symbolic Program Synthesis (NSPS) (Parisotto et al., 2017) and Learning to Infer Program Sketches (Nye et al., 2019) augment partial symbolic sketches with neural modules, typically relying on handcrafted DSLs or differentiable logic. Recent systems like NEAR (Shah et al., 2020), NeuroSketcher (Emireddy, 2023), and SYNAPSE (Modak et al., 2025) extend this idea by treating sketches as partially differentiable computation graphs. Implicit-representation work such as NeuroSAT (Selsam et al., 2019) and dPads (Cui & Zhu, 2021) also uses continuous spaces, but ultimately decodes back to discrete code. In contrast, our method dispenses with DSLs and intermediate symbolic sketches altogether and treats the *entire weight vector* of a fixed architecture as the synthesized program, so the hypernetwork directly solves program synthesis in weight space rather than decoding back into a discrete language.

Hypernetworks and meta-learning. Meta-learning and hypernetwork approaches attempt to learn mappings from task conditions to full model parameters. HyperNetworks (Ha et al., 2016) pioneered the idea of using one network to generate weights for another. Similarly, MAML (Finn et al., 2017) and its variants (Rusu et al., 2019) enable fast adaptation to new tasks using shared priors. Beyond the original HyperNetworks paper, many later works explore task-conditioned hypernetworks and meta-learning for continual learning and scientific ML (von Oswald et al., 2020; Zhao et al., 2020; Beck et al., 2023; Cho et al., 2023). However, most hypernetwork or meta-learning settings still rely on some form of test-time supervision, such as gradient adaptation on labeled examples from the new task. In contrast, our synthesis framework requires *complete zero-shot generation*: at deployment the meta-learner receives only the user intent U, with no access to test input–output pairs (X, Y), and must emit correct and executable weights $\hat{\theta}$.

Weights as a modality and permutation symmetries. A growing body of work treats *weights themselves* as a first-class learning signal (Schürholt et al., 2021). Ainsworth et al. (2023) demonstrate that aligned neuron permutations can "re-basin" independently trained models, while Navon et al. (2023) introduce DWSNets—weight-space networks that enforce neuron-permutation equivariance for MLPs. Subsequent studies extend this principle: Zhou et al. (2024) apply permutation-equivariant functionals to CNNs, and Lim et al. (2024); Kofinas et al. (2024) represent diverse architectures as *neural graphs*, using GNN updates that naturally respect permutation symmetry, albeit only for MLPs and CNNs. In contrast to methods that *ingest trained networks* for analysis or editing, our approach is *generative*: a meta-learner maps user intent to a complete set of weights for a fixed template, producing an executable network. Moreover, we extend the neural-graph formulation and symmetry-aware message passing to *full Transformers*—not only MLPs or CNNs—covering multi-head attention, residual pathways, and layer normalization.

3 Problem Setup

Programs as (S, θ). We model any executable program as a pair (S, θ) of a fixed structure S (architecture or grammar) and parameters θ. A valid program must satisfy

$$\textbf{syntax: } \mathrm{valid\_syntax}(S, \theta) = \text{true}, \qquad \textbf{semantics: } \forall (X, Y) \in \mathcal{D}_U : \ \mathrm{exec}(S, \theta, X) = Y.$$

Symbolic programs: Verification. With slot budget T, parameters are token sequences $\theta \in \Theta_{\mathrm{sym}}$ over a finite vocabulary. The execution map $\mathrm{exec}(S, \theta, \cdot)$ is evaluated via a compiler or interpreter and returns discrete pass/fail outcomes. This signal is non-graded and non-local: it reports *what* failed, not *how* to adjust θ. **Search.** Because tokens are discrete and syntax entangles with semantics, small edits can alter either or both; the objective landscape is piecewise constant.

Neural programs: Verification. Let T be the number of scalar parameters in S; neural parameters are $\theta \in \Theta_{\mathrm{neu}} = \mathbb{R}^T$. The execution map is the forward pass $m_{S,\theta}$. Let U denote the intent and $\mathcal{D}_U$ the corresponding data distribution over (X, Y). The semantic risk is

$$\mathcal{L}_{\mathrm{sem}}(S, \theta) \ = \ \mathbb{E}_{(X,Y)\sim\mathcal{D}_U}\big[\mathcal{L}\big(m_{S,\theta}(X), Y\big)\big]$$

is differentiable in θ, providing dense, graded signals that convey both direction and magnitude of improvement. **Optimization.** Fixing S yields $\mathrm{valid\_syntax}(S, \theta)$ for all $\theta \in \mathbb{R}^T$, removing syntax as a constraint during learning and disentangling it from semantics. Cast synthesis as continuous weight-space optimization with informative gradients, rather than discrete token search.

Trade-offs and scalability. We explicitly acknowledge two trade-offs.

Approximate semantics. By moving from discrete code to real-valued weights, correctness becomes approximate (e.g., a high-accuracy classifier rather than a Boolean proof). In return, we gain true scalability: gradients provide dense supervision, training is efficient, and we avoid DSL engineering altogether—fixing S defines the space, and synthesis reduces to learning θ from intent.

Human interpretability. Neural programs are not line-by-line human-readable like symbolic code. However, they are still deterministic, closed-form computations, and a substantial toolbox exists to analyze them. Attribution and feature-based methods (e.g., LIME, SHAP, Grad-CAM), concept-based analyses (TCAV), and mechanistic studies of circuits and weight-space structure provide multiple lenses for inspecting $m_{S,\theta}$ and relating parameters to behavior (Ribeiro et al., 2016; Lundberg & Lee, 2017; Selvaraju et al., 2017; Kim et al., 2018; Olah et al., 2020; Nanda et al., 2023).

4 LEARNING TO SYNTHESIZE NEURAL PROGRAMS

Meta Learning Setting. Let $\mathcal{U} \subset \mathbb{R}^{d_u}$ be the encoded intent space and $\Theta(S) \subset \mathbb{R}^T$ the weight space for S. A meta-learner (hypernetwork) $M_\phi : \mathcal{U} \to \Theta(S)$ maps $u \mapsto \hat{\theta} = M_\phi(u)$, yielding the composed system $F(u, x) = m_{S, M_\phi(u)}(x)$.

Assumption 1 (realizable template)**.** *A task family is realizable by a fixed, over-parameterized network template $S^\star$ iff*

$$\exists\, \theta \in \mathbb{R}^T \;\; \textit{s.t.} \;\; \operatorname{exec}(S^\star, \theta, X) = Y \quad \forall\, (X, Y) \sim \mathcal{D}_U.$$

With $S^\star$ fixed, synthesis reduces to minimizing the *expected semantic loss* over weights,

$$\min_{\theta \in \mathbb{R}^T} \mathcal{L}_{\text{sem}}(\theta), \qquad \mathcal{L}_{\text{sem}}(\theta) = \mathbb{E}_{(X,Y)\sim\mathcal{D}_U}\Big[\mathcal{L}\big(\operatorname{exec}(S^\star, \theta, X),\, Y\big)\Big],$$

where $\mathcal{L}$ is a differentiable loss (e.g., mean squared error) to be minimized by gradient descent.

We further show that, under common conditions on the task family, the *learnability of an approximation* for intent$\to\ \theta$ follows from standard universal-approximation and weight–perturbation results; we summarize these guarantees and give formal statements in App. A. While of interest to some readers, our emphasis here is practical—the design of a symmetry-aware hypernetwork (Meta-GNN) and its evaluation on increasingly challenging synthesis suites—which we develop next.

4.1 AMBIGUITY IN WEIGHT SPACE AND DESIGN STRATEGIES

Even with a fixed architecture $S^\star$, many distinct parameterizations implement the same function. Two ubiquitous sources of ambiguity are:

- **Multiple minima.** Gradient-based training can converge to different θ that attain (near-)zero semantic loss; denote the set of such solutions by $\mathcal{O}$.
- **Neuron permutations.** Layerwise permutations of hidden units map any valid θ to another $\pi\theta$ with identical network function; let Π be the permutation group.

We adopt two complementary design principles:

End-to-end training on semantic loss. Because the execution map is differentiable in weights, backpropagating the task loss through $M_\phi(u) \mapsto \hat{\theta}$ leverages continuity to learn a stable intent$\to$weights mapping without any post-hoc alignment.

Symmetry-aware inductive bias. Respecting permutation symmetries *collapses* equivalent parameterizations and reduces the effective search space. Concretely, tying parameters across permutation-equivalent groups and using structure-aware updates substantially reduces the redundancy in supervision.

4.2 PERMUTATION EQUIVARIANCE VIA NEURAL GRAPHS

We operationalize symmetry awareness—indeed, broader structure awareness—by casting S as a *neural graph* and deriving an update rule equivariant to its automorphisms.

Equivariance and invariance. Following Section 3 of Navon et al. (2023), we recall the basic notation and definitions. Let G be a group and let V, W be vector spaces. The *general linear group* $\mathrm{GL}(V)$ denotes all invertible linear maps $V \to V$. A *representation* of G on V is a homomorphism $\rho_1 : G \to \mathrm{GL}(V)$, where $\mathrm{GL}(V)$ denotes the group of invertible linear maps $V \to V$ (i.e., invertible matrices in coordinates). Analogously, $\rho_2 : G \to \mathrm{GL}(W)$ describes the action of G on W. A map $L : V \to W$ is said to be *G-equivariant* if

$$L\big(\rho_1(g)\, v\big) \;=\; \rho_2(g)\, L(v) \qquad \forall g \in G,\, v \in V. \tag{1}$$

Intuitively, applying a group symmetry before or after L yields the same result. A special case is *G-invariance*, where $\rho_2(g)$ is the identity for all g (e.g., $W = \mathbb{R}$), so $L(\rho_1(g)v) = L(v)$.

Neural graphs and automorphisms. To unify these cases we represent the fixed template $S^\star$ as a directed multigraph $G^\star = (V, E)$, with nodes for biases and edges for weights. A *neural-graph automorphism* $\phi : V \to V$ is a node relabeling that (i) preserves adjacency, (ii) respects node types (input/hidden/output/bias), and (iii) preserves weight-sharing constraints. Each automorphism induces a consistent permutation of parameters via $\phi_e(i, j) = (\phi(i), \phi(j))$, generalizing hidden-unit permutations in MLPs and channel permutations in CNNs. Crucially, automorphisms leave the instantiated network function unchanged, and message-passing GNNs defined on $G^\star$ are provably equivariant to these induced parameter permutations (Lim et al., 2024).

4.3 Structure-Aware Weight Synthesis

Motivated by the structure-aware view developed above—neural graph formulation and symmetry considerations—we present three meta-architectures with increasingly strong structure awareness:

Notation. Let U denote the *encoded* user intent embedding. All maps $f(\cdot)$ are MLPs.

Meta-MLP (structure-agnostic). Let the target network be $(S^\star, \theta)$, and let T be the total number of scalar parameters in θ. A single meta-MLP parameterized by φ predicts the weights, given intent embedding U, as a flat vector:

$$\hat{\theta} \;=\; f_{\mathrm{MLP}}(U; \varphi) \in \mathbb{R}^T.$$

Meta-gMLP (group-aware). Index parameters by slots $t \in \{1, \dots, T\}$ with group map $g(t) \in \mathcal{G}$ and a fixed *per-weight* positional encoding $\mathbf{p}_t$. Use *group-tied* encoder parameters $\{\varphi_g\}_{g\in\mathcal{G}}$ and a global mixer with parameters φ_{mixer}:

$$\mathbf{h}_t \;=\; f\big([\,U \,\|\, \mathbf{p}_t\,];\, \varphi_{g(t)}\big), \qquad \hat{\theta} \;=\; f\left([\mathbf{h}_t]_{t=1}^{T};\, \varphi_{\mathrm{mixer}}\right),$$

Meta-GNN (group- and structure-aware). Represent the template $S^\star$ as a parameter graph $G^\star = (V, E)$ with neighborhood $\mathcal{N}(i) \coloneqq \{\, j \in V :\ (i, j) \in E \text{ or } (j, i) \in E \,\}$. Nodes (bias slots) and edges (weight slots) have *per-slot* positional encodings $\mathbf{p}_{V(i)}$, $\mathbf{p}_{E(i,j)}$ and belong to groups $g_{V(i)}$, $g_{E(i,j)}$ as in Meta-gMLP. Let $\mathbf{h}_V^t(i), \mathbf{h}_E^t(i, j) \in \mathbb{R}^d$ be embeddings after t steps. Use *group-tied* encoder parameters $\{\varphi_{\mathrm{enc}}^g\}_{g\in\mathcal{G}}$ and decoder parameters $\{\varphi_{\mathrm{dec}}^g\}_{g\in\mathcal{G}}$, together with global parameters for messaging and updates, denoted φ_{msg}, φ_{upd}^V, and φ_{upd}^E. See Fig. 1 for a visual overview; the update modules f_{upd} are omitted there for space.

Initialization (shared within groups).

$$\mathbf{h}_V^0(i) = f\big([\,U \,\|\, \mathbf{p}_{V(i)}\,];\, \varphi_{\mathrm{enc}}^{g_{V(i)}}\big), \quad \mathbf{h}_E^0(i, j) = f\big([\,U \,\|\, \mathbf{p}_{E(i,j)}\,];\, \varphi_{\mathrm{enc}}^{g_{E(i,j)}}\big).$$

Message passing (permutation-invariant aggregation). For $t = 1, \dots, k$,

$$\begin{aligned}
\mathbf{m}_i^t &= \mathrm{AGG}_{j\in\mathcal{N}(i)}\, f\big(\mathbf{h}_V^{t-1}(i), \mathbf{h}_V^{t-1}(j), \mathbf{h}_E^{t-1}(i, j); \varphi_{\mathrm{msg}}\big),\\
\mathbf{h}_V^t(i) &= f\big(\mathbf{h}_V^{t-1}(i), \mathbf{m}_i^t; \varphi_{\mathrm{upd}}^V\big),\\
\mathbf{h}_E^t(i, j) &= f\big(\mathbf{h}_V^t(i), \mathbf{h}_V^t(j), \mathbf{h}_E^{t-1}(i, j); \varphi_{\mathrm{upd}}^E\big).
\end{aligned}$$

Decoding (shared within groups).

$$\hat{b}_i = f\big(\mathbf{h}_V^k(i); \varphi_{\mathrm{dec}}^{g_{V(i)}}\big), \qquad \hat{w}_{ij} = f\big(\mathbf{h}_E^k(i, j); \varphi_{\mathrm{dec}}^{g_{E(i,j)}}\big).$$

Structure awareness in learning. Without positional encodings, Meta-GNN is permutation–equivariant under automorphisms of the neural graph. Let $\mathrm{Aut}(G^\star)$ be the automorphism group of $G^\star$ and $\rho_\Theta(\phi)$ the induced permutation of the flattened parameter vector. If (i) all $\varphi_{\mathrm{enc}}, \varphi_{\mathrm{msg}}, \varphi_{\mathrm{upd}}, \varphi_{\mathrm{dec}}$ are shared within groups and (ii) AGG is order-invariant, then for any $\phi \in \mathrm{Aut}(G^\star)$,

$$\widehat{\theta}(U; G^\star) \;\mapsto\; \rho_\Theta(\phi)\, \widehat{\theta}(U; G^\star) \;=\; \widehat{\theta}(U; \phi \cdot G^\star).$$

Unlike prior work that *encodes trained networks* (Navon et al., 2023; Lim et al., 2024; Zhou et al., 2024), we are *generative*: the meta-learner must emit a concrete $\hat{\theta}$ that is *written into fixed tensor slots* of $S^\star$. In this setting, strict permutation–equivariance is incompatible with deployment: if hidden units remain exchangeable, there is no permutation-canonical way to assign scalars to slots, just as training a vanilla MLP requires asymmetric initialization to avoid neuron exchangeability. Following the Transformer's design (Vaswani, 2017), we break symmetry *only* at initialization by

adding positional encodings to the *initial* intent embedding (cf. Fig. 1, upper-left). The Meta-GNN's encoding, message-passing, and decoding modules *share parameters within groups* and are otherwise permutation-invariant. Thus, the practical gain over a classical (flat) hypernetwork (Meta-MLP) is **structure awareness**: (i) permutation-aware learning via invariant aggregation and group tying, and (ii) *locality* through message passing over graph neighborhoods. Together, these choices explicitly exploit the computation graph, *preserve* target-network expressivity (Navon et al., 2023; Lim et al., 2024), and collapse symmetry-induced redundancy—thereby shrinking the effective search space that a flat Meta-MLP cannot model.

5 EXPERIMENTS

5.1 BENCHMARKS

We evaluate three settings (Table 1) chosen to pair *classical target architectures and losses*—MLP+MSE, Transformer+CE, seq2seq+CE—with progressively richer intent structure. SUMFIRST–n and ADDMOD–p each form a *single* program family with a single varying parameter (the full spaces are `sumfirst_n`, $n \in [1, 10]$, and `addmod_p`, primes $p < 100$), yielding clean, low-entropy program domains. By contrast, the 1D–ARC subset is *multi-family*: intents are (family, parameter) pairs spanning transformations such as `Flip`, `Hollow`, `Fill`, `RecolorBar`$(1-9)$, and `MoveRight`$(1-9)$—operations easy to specify yet non-trivial to implement robustly. Representative families are illustrated in Fig. 2.

Figure 2: 1D–ARC subset (illustration). Example families (e.g., *RecolorBar*, *MoveRight*, *Flip*) with intents specified as (family, parameter) pairs referenced in Table 1.

Field	**SUMFIRST-n**	**ADDMOD-p**	**1D-ARC (mixture)**
Program space	$y = \sum_{k=1}^{n} L_k$	$y = (a + b) \bmod p$	Task-specific 1D transformations
Input / output	List $L \in \mathbb{Z}^{10} \mapsto y$	Pair $(a, b) \mapsto y,\ a, b < p$	1D grid of fixed length $= 70$
Target network m_θ	1-hidden layer ReLU MLP	1-layer Transformer	Seq2seq Transformer
# Weights	193	6 048	17 568
Loss function	Mean squared error (MSE)	Cross-entropy (classification)	Cross-entropy (Seq2Seq)
User intent	Prefix length $n \in \{1, \ldots, 10\}$	Prime modulus $p < 100$	(family, parameter)

Table 1: We deliberately select three representative combinations of architecture and loss to demonstrate that our pipeline is agnostic to both model architecture and loss. For 1D-ARC, the user intent is specified by a *program family* (e.g., `RecolorBar`) together with a *parameter* (e.g., 2).

Positioning w.r.t. symbolic DSL benchmarks: our setup is *DSL-free*, mapping intents directly to neural weights, so results are not directly comparable to systems whose performance hinges on the chosen DSL. Unlike large combinatorial DSL spaces, we use compact program spaces—one family with one (or few) intent variables (n, p, or (family, parameter))—to target OOD *over intent* and permit mechanistic analysis. Still, SUMFIRST–n and ADDMOD–p are spirit-close to canonical SyGuS tasks, and 1D–ARC extends to more realistic sequence transforms.

Data splits. For each benchmark, intents (e.g., n or p) are partitioned into *train* vs. *held-out*. Within each intent we form four partitions of I/O pairs: **train** (used to update the meta-learner), **test** (new (X, Y) for the *same* train intents, measuring input-level generalization at fixed intent), **validation**

Model	SUMFIRST-n			ADDMOD-p			1D-ARC (MIXTURE)		
	Train	Test	OOD Test	Train	Test	OOD Test	Train	Test	OOD Test
direct MLP	1.000	1.000	0.140	0.928	0.916	0.699	—	—	—
direct Transformer	—	—	—	0.849	0.850	0.682	0.994	0.996	0.603
Meta-MLP	1.000	1.000	0.145	0.996	0.994	**0.947**	0.953	0.956	<u>0.625</u>
Meta-gMLP	0.998	0.996	<u>0.344</u>	0.935	0.923	0.883	0.932	0.942	0.610
Meta-GNN	1.000	1.000	**0.866**	0.995	0.992	<u>0.943</u>	0.976	0.978	**0.713**

Table 2: Best-seed accuracies. **Bold** indicates the highest OOD Test accuracy; <u>underline</u> the second highest. A direct Transformer baseline is omitted on SUMFIRST-n (raw-integer inputs are ill-suited to sequence models). A direct MLP baseline is omitted on 1D–ARC (seq2seq loss incompatible with naive MLPs). For the full seed distributions and variance analyses see App. D.

(unseen intents used *only* for model selection), and **OOD test** (unseen intents held out for final reporting, measuring zero-shot generalization across intents).

In 1D–ARC (Fig. 2), some families are *not parametric* (`Fill`, `Hollow`, `Flip`) while others are *parametric* (`RecolorBar`, `MoveRight`). For the latter, training covers parameters $\{1-6, 8, 9\}$ and the held-out parameter 7 is used exclusively for OOD test.

Primary metric: *percentage of solved instances*. For SUMFIRST–n we round to the nearest integer and check equality; for ADDMOD–p and 1D–ARC we use exact match. We have trained each model with ten random seeds to assess variability to weight initialization as well as a means of increasing the odds of converging to weights that perform well on **OOD test** instances; both of these uses of the random seeds are common (Bethard, 2022).

Baselines and capacity matching. *Direct prediction* trains $(U, X) \mapsto Y$ on **train** only (no meta component). *Meta-MLP* is the *classical hypernetwork* baseline: a MLP that maps $U \mapsto \hat{\theta}$ as a flat *list* of parameters. *Meta-gMLP* (group-aware) and *Meta-GNN* (neural-graph, equivariant) add progressively stronger structural bias while sharing the same target $m_{S,\theta}$. All models are capacity-matched to comparable trainable parameter counts (details in App. §C).

5.2 RESULTS

Is neural synthesis even possible? First, we note that all meta models generate *semantically correct programs* for in-distribution test set in a single forward pass across benchmarks, including a $\sim$17k-parameter Seq2Seq Transformer for 1D-ARC. Table 2 shows that **test** accuracies are close to 100% across models (rows) and benchmarks ("Test" columns). The real challenge, however, is in out-of-distribution generalization, as we will discuss next.

Benefit over direct prediction. Across all three benchmarks, meta-learners substantially outperform direct models on **OOD test** (unseen intents) under capacity matching. This supports our claim that *predicting programs (weights)* yields more reliable intent generalization than *predicting outputs*. In ADDMOD–p, we further show that the meta-learner recovers a *closed-form* $U \to \theta$ mapping (Sec. 5.3), rather than memorizing instances.

Role of structural bias. Meta-GNN consistently outperforms other hypernetwork variants. By viewing the target as a neural graph and using structure-aware message passing, Meta-GNN *reduces* weight-space redundancy so that supervision is less fragmented. This structural bias yields more consistent learning signals and, empirically, stronger OOD performance.

Clarification on Meta-gMLP's ranking. Although Meta-gMLP incorporates grouping, it does not consistently rank second on ADDMOD-p and 1D-ARC. Under a fixed meta-network budget, its global mixer f_{mixer} processes $[\mathbf{h}_t]_{t=1}^{T} \in \mathbb{R}^{T \cdot d_{\text{meta}}}$, which forces d_{meta} to shrink as T grows and creates a capacity bottleneck on larger $S^\star$. By contrast, Meta-MLP's capacity scales with its own width (independent of T), and Meta-GNN avoids global concatenation by local message passing with group-shared modules.

Multi-family synthesis (1D–ARC). Beyond single-family arithmetic, the 1D–ARC mixture tests a more "real-world" regime with diverse families and intents (Fig. 2). Table 3 shows that Meta-

Model	Fill		Flip		Hollow		MoveRight		RecolorBar	
	Test	*OOD*	*Test*	*OOD*	*Test*	*OOD*	*Test*	*OOD*	*Test*	*OOD*
Direct Transformer	1.000	1.000	1.000	1.000	1.000	0.999	0.999	0.000	1.000	0.016
Meta-MLP	0.995	0.992	0.138	0.127	0.996	0.997	0.999	0.996	1.000	0.000
Meta-gMLP	0.979	0.967	0.150	0.165	0.965	0.962	0.986	0.981	0.987	0.000
Meta-GNN	0.994	0.992	0.905	0.898	0.789	0.774	0.990	0.921	1.000	0.000

Table 3: Per-family **test** and **OOD test** success on 1D-ARC (mixture) for the best run.

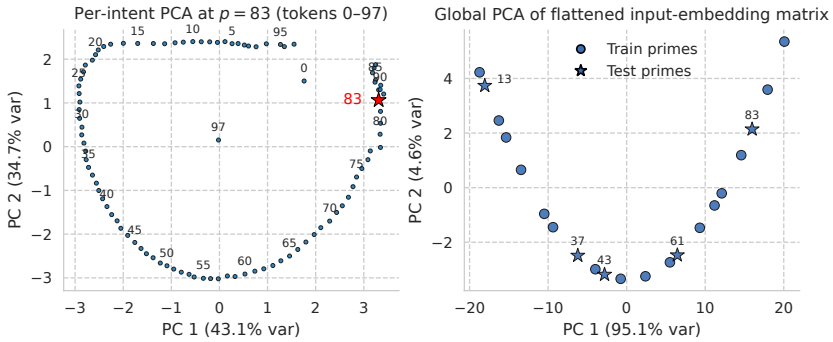


Figure 3: **Clock geometry of the synthesized Transformer.** *Left:* per–intent PCA for the predicted input–embedding matrix W_E of unseen p=83. *Right:* global PCA across primes using the $W_E(p)$.

GNN attains high OOD accuracy across most families. Notably, `RecolorBar(color)` is unlearnable for all meta-learners because the intent is *categorical* (no ordinal structure for unseen colors), whereas a direct Transformer can in principle match the color token via attention; in `MoveRight(dist)`, direct Transformers fail to extrapolate while all meta-learners succeed; and in `Flip`, only Meta-GNN reaches high accuracy thanks to structure-aware learning that reduces weight-space redundancy.

Overall, the empirical results align with our research questions: (i) *neural programs in weight space* can be synthesized end-to-end; (ii) continuous intent$\to\theta$ mappings are learnable and semantically stable, and (iii) structure-aware inductive bias (Meta-GNN) is key to strong OOD generalization across both single- and multi-family settings. Additional distributional analyses appear in App. §D.

5.3 Mechanistic Analysis

For AddMod–p, the target network admits a known closed-form ("clock") solution (Nanda et al., 2023; Zhong et al., 2024): the embedding maps (a, b) to $(\cos\tilde{\theta}, \sin\tilde{\theta})$ with $\tilde{\theta} = \frac{r}{p} + \varphi$; the core (attn+MLP) performs an angle–sum of the two embeddings; and the logit head outputs $\cos\big(w(a+b-c)\big)$. Because downstream blocks are *frequency-agnostic*, a perfect meta-learner need only learn $p \mapsto \big(W_E(p),\, W_U(p)\big)$, leaving the angle–sum circuitry unchanged.

Per–intent geometry (unseen p). For an unseen modulus (p=83), PCA of token embeddings shows the expected clock geometry (Fig. 3, *left*), consistent with the analytic construction.

Across–intent geometry $\Rightarrow$ closed form. Flattening $W_E(p)$ across primes and running a *global* PCA reveals a thin 1D trajectory in $(\mathrm{PC}_1, \mathrm{PC}_2)$ (Fig. 3, *right*), indicating that the synthesized weights are governed by a single latent angle of the form $\frac{r}{p} + \varphi$. Fitting this trajectory yields the closed-form dependence with coefficients and goodness-of-fit (see full derivations in App. §E):

$$x = \cos\big(5.672/p + 3.631\big), \qquad \beta_k = \binom{1/2}{k}(-1)^k;$$

$$\mathrm{PC}_1(p) = 23.43\,x, \qquad \mathrm{PC}_2(p) = b + B\sum_{k=1}^{4}\beta_k x^{2k} + \sum_{j=0}^{3}\alpha_{2j+1}x^{2j+1} \quad \text{(overall } R^2 > 0.94\text{)}.$$

6 SCOPE AND LIMITATIONS

Our benchmarks are intentionally small in symbolic form: three synthetic but structured families (SUMFIRST–n, ADDMOD–p, and 1D–ARC) with fixed architectures and standard losses. This design lets us isolate the feasibility of strict zero-shot intent$\rightarrow \theta$ synthesis and perform mechanistic analyses (e.g., recovering the clock circuit on ADDMOD–p), rather than confounding the results with large, noisy domains. Although the task descriptions are simple, each intent corresponds to a point in a high-dimensional continuous weight space, and the meta-learner must discover smooth structure there without any hand-crafted DSL.

A second limitation is architectural: each task currently uses a single, relatively shallow target template (1-hidden-layer MLP, 1-layer Transformer, or 1-layer seq2seq Transformer). We chose these templates to keep the synthesized programs interpretable and to leverage known ground-truth mechanisms. In our framework, scaling corresponds to enlarging the neural graph $G^\star$ (more nodes and edges), so Meta-GNN's cost grows roughly linearly in $|V|+|E|$; applying the same synthesis principle to deeper or mixed architectures is left to future work.

Finally, we do not compare directly to classical or neuro-symbolic program-synthesis systems. Those methods operate over a hand-crafted DSL or template language, whereas our setting is DSL-free: the program *is* the weight vector. Introducing a bespoke DSL solely to run such baselines would make performance depend largely on the DSL design and obscure the weight-space mechanisms we study, so we instead use capacity-matched direct neural models as baselines.

7 CONCLUSION

We asked whether program synthesis can operate directly in the continuous weight space of neural networks. Our findings answer in the affirmative. Treating a fixed architecture and its weights as a *neural program* (**NeuroP**), we showed that (i) *From intent*, we synthesize MLPs and Transformers that are *semantically correct* in a single forward pass. To the best of our knowledge, this is the first demonstration of this possibility; and (ii) incorporating structural bias—via a structure-aware Meta-GNN—substantially improves OOD accuracy on *unseen* intents over direct intent + input baselines, with mechanistic evidence on ADDMOD-p that the learner internalizes a stable higher-order mapping $U \rightarrow \theta$. Taken together, the results position neural weights as an expressive, differentiable, and *learnable* program modality, and **NeuroP** as a principled route to scalable synthesis. Our future work will focus on extending the proposed framework to more complex program synthesis applications. In particular, realistic user intents could be expressed in natural or other structured languages. Alternatively, user intent could be expressed through a few example input-output pairs, as is the case in the Abstraction and Reasoning Corpus (Chollet, 2019). These complex modalities require a more sophisticated hypernetwork.

ACKNOWLEDGMENTS

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (MSIT) (No. RS-2024-00457882, National AI Research Lab Project).

REFERENCES

Alessandro Abate, Haniel Barbosa, Clark Barrett, Cristina David, Pascal Kesseli, Daniel Kroening, Elizabeth Polgreen, Andrew Reynolds, and Cesare Tinelli. Synthesising programs with non-trivial constants. *Journal of Automated Reasoning*, 67(2):19, May 2023. ISSN 1573-0670. doi: 10.1007/s10817-023-09664-4. URL https://doi.org/10.1007/s10817-023-09664-4.

Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2023. URL https://arxiv.org/abs/2209.04836.

Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pp. 1–8, 2013. URL http://ieeexplore.ieee.org/document/6679385/.

Jacob Beck, Risto Vuorio, Zheng Xiong, and Shimon Whiteson. Recurrent hypernetworks are surprisingly strong in meta-rl. *Advances in Neural Information Processing Systems*, 36:62121–62138, 2023.

Steven Bethard. We need to talk about random seeds. *arXiv preprint arXiv:2210.13393*, 2022.

Woojin Cho, Kookjin Lee, Donsub Rim, and Noseong Park. Hypernetwork-based meta-learning for low-rank physics-informed neural networks. *Advances in Neural Information Processing Systems*, 36:11219–11231, 2023.

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

Guofeng Cui and He Zhu. Differentiable synthesis of program architectures. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ivXd1iOKx9M.

Teja R Emireddy. Neurosketcher: Synthesizing non-differentiable programs with non-differentiable loss functions. *University of Alberta Thesis*, 2023.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. doi: 10.1561/2500000010.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Kangjing Huang, Xiaokang Qiu, Peiyuan Shen, and Yanjun Wang. Reconciling enumerative and deductive program synthesis. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pp. 1159–1174, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136. doi: 10.1145/3385412.3386027. URL https://doi.org/10.1145/3385412.3386027.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pp. 2668–2677. PMLR, 2018.

Jinwoo Kim, Qinheping Hu, Loris D'Antoni, and Thomas Reps. Semantics-guided synthesis. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi: 10.1145/3434311. URL https://doi.org/10.1145/3434311.

Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, and David W. Zhang. Graph neural networks for learning equivariant representations of neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oO6FsMyDBt.

Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. doi: 10.1016/S0893-6080(05)80131-5.

Derek Lim, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ijK5hyxs0n.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Sadanand Modak, Noah Tobias Patton, Isil Dillig, and Joydeep Biswas. Synapse: Symbolic neural-aided preference synthesis engine. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 27529–27537, 2025.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pp. 25790–25816. PMLR, 2023.

Maxwell Nye, Luke Hewitt, Joshua Tenenbaum, and Armando Solar-Lezama. Learning to infer program sketches. In *International Conference on Machine Learning*, pp. 4861–4870. PMLR, 2019.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJ0JwFcex.

Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark Barrett, and Cesare Tinelli. cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In Isil Dillig and Serdar Tasiran (eds.), *Computer Aided Verification*, pp. 74–83, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25543-5.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJgklhAcK7.

Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

Ameesh Shah, Eric Zhan, Jennifer J. Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. Learning differentiable programs with admissible neural heuristics. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California, Berkeley, 2008. URL http://people.csail.mit.edu/asolar/papers/thesis.pdf.

Armando Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013. doi: 10.1007/s10009-012-0249-7. URL https://doi.org/10.1007/s10009-012-0249-7.

Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017. doi: 10.1016/j.acha.2015.12.005.

Yu-Lin Tsai, Chia-Yi Wei, Hung-Yi Tseng, and Hsuan-Tien Lee. Formalizing generalization and adversarial robustness of neural networks to weight perturbations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/a3ab4ff8fa4deed2e3bae3a5077675f0-Paper.pdf.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJgwNerKvB.

Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Wei Ye, Luca Daniel, Cho-Jui Lin, and Cho-Jui Hsieh. Towards certificated model robustness against weight perturbations. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6105.

Kaidi Xu, Huan Li, Sijia Zhang, Yihan Wang, Suman Jana, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper/2020/file/0cbc5671ae26f67871cb914d81ef8fc1-Paper.pdf.

Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias Boutros Khalil. LLMs and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=E8m8oySvPJ.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017. doi: 10.1016/j.neunet.2017.07.002.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Dominic Zhao, Seijin Kobayashi, João Sacramento, and Johannes von Oswald. Meta-learning via hypernetworks. In *4th Workshop on Meta-Learning at NeurIPS 2020 (MetaLearn 2020)*. NeurIPS, 2020.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Allan Zhou, Chelsea Finn, and James Harrison. Universal neural functionals. *arXiv preprint arXiv:2402.05232*, 2024.

A Formal Guarantees for Weight-Space Synthesis

A.1 Formal Guarantees

Meta Learning Setting. Let $\mathcal{U} \subset \mathbb{R}^{d_u}$ be the encoded intent space and $\Theta(S) \subset \mathbb{R}^T$ the weight space for S. A meta-learner (hypernetwork) $M_\phi : \mathcal{U} \to \Theta(S)$ maps $u \mapsto \hat{\theta} = M_\phi(u)$, yielding the composed system

$$F(u, x) = m_{S, M_\phi(u)}(x).$$

A.2 Assumptions

A1 (Realizability). For $u \in \mathcal{U}$ there exists $\theta \in \Theta(S)$ such that $m_{S,\theta}(X) = Y$ for all $(X, Y) \sim \mathcal{D}_u$.

A2 (Continuity). There exists a continuous target mapping $\omega : \mathcal{U} \to \Theta(S)$ on compact $\mathcal{U}$.

A.3 Universal approximation for intent→weights

By universal approximation for ReLU networks (Leshno et al., 1993; Sonoda & Murata, 2017; Yarotsky, 2017):

Theorem 1 (UAT for hypernetworks)**.** *For any $\varepsilon > 0$ there exists a (ReLU) hypernetwork $M_\phi : \mathcal{U} \to \Theta(S)$ such that*

$$\sup_{u \in \mathcal{U}} \|M_\phi(u) - \omega(u)\|_\infty \ < \ \varepsilon.$$

Thus, under A1–A2, any continuous intent→ θ map is uniformly learnable by a hypernetwork.

A.4 Lipschitz semantics: from parameter to function error

For common architectures, certified weight-perturbation bounds can be obtained from layerwise operator norms or LiRPA certificates (Weng et al., 2020; Tsai et al., 2021; Xu et al., 2020). We summarize these as:

Lemma 1 (Parameter–Lipschitz semantics)**.** *There exists $L_\theta < \infty$ (on a parameter region Θ and bounded input domain $\mathcal{X}$) such that*

$$\sup_{x \in \mathcal{X}} \|m_{S,\theta}(x) - m_{S,\theta'}(x)\| \ \le \ L_\theta \, \|\theta - \theta'\| \qquad \forall \, \theta, \theta' \in \Theta.$$

Corollary 1 (Uniform control of semantic error)**.** *If $\sup_{u \in \mathcal{U}} \|M_\phi(u) - \omega(u)\| \le \varepsilon_\theta$, then*

$$\sup_{u \in \mathcal{U}} \sup_{x \in \mathcal{X}} \left\| m_{S, M_\phi(u)}(x) - m_{S, \omega(u)}(x) \right\| \ \le \ L_\theta \, \varepsilon_\theta.$$

For classification, if $m_{S,\omega(u)}$ has margin $\gamma(u) > 0$ on $\mathcal{X}_u$, any $\varepsilon_\theta < \gamma(u)/L_\theta$ preserves the predicted labels on $\mathcal{X}_u$.

Together, Theorem 1 and Corollary 1 establish two formal pillars: (i) *approximability* of continuous intent→θ mappings by ReLU hypernetworks, and (ii) *semantic stability* under small parameter errors—precisely the regime targeted by end-to-end gradient-based training. While these results do not address sample complexity or statistical rates, our empirical sections demonstrate that such mappings can be learned effectively in practice. Full proofs are deferred to Appendix A.5.

A.5 Proofs

This appendix provides fully formal, self-contained arguments with explicit norms and codomain handling. We use only standard facts about (i) universal approximation on compact sets and (ii) layerwise Lipschitz composition.

Standing assumptions, norms, and notation. Let $\mathcal{U} \subset \mathbb{R}^{d_u}$ be compact and $\mathcal{X} \subset \mathbb{R}^{d_x}$ be bounded. Fix a feedforward ReLU architecture

$$h_0(x) = x, \qquad h_\ell(x) = \rho\big(W_\ell h_{\ell-1}(x) + b_\ell\big), \quad \ell = 1, \dots, M,$$

with ρ L_ρ-Lipschitz (ReLU: $L_\rho = 1$). The model is $m_{S,\theta} \coloneqq h_M(\cdot; \theta)$ with parameters $\theta = \{(W_\ell, b_\ell)\}_{\ell=1}^M$. We **flatten** per-layer tensors through a fixed linear bijection $\mathrm{vec} : \Theta_{\text{tensor}}(S) \to \mathbb{R}^T$ and write again $\Theta(S) \subseteq \mathbb{R}^T$ for the admissible set in vectorized form. All proofs are invariant to this reshape.

Norms. Vectors use $\|\cdot\|_2$; matrices use the induced operator norm $\|\cdot\|_2$. On parameters we use the block norm

$$\|\theta\|_{\text{par}} \ := \ \sum_{\ell=1}^{M} \big(\|W_\ell\|_F + \|b_\ell\|_2\big),$$

so that $\sum_\ell(\|\Delta W_\ell\|_F + \|\Delta b_\ell\|_2) \le \|\theta - \theta'\|_{\mathrm{par}}$. When measuring uniform approximation errors of vector-valued maps $\mathcal{U} \to \mathbb{R}^T$, we use $\|\cdot\|_\infty$ on $\mathbb{R}^T$ unless stated otherwise. Since all norms on $\mathbb{R}^T$ are equivalent, there exists $c_{\mathrm{par}\leftarrow\infty} \in (0,\infty)$ with $\|z\|_{\mathrm{par}} \le c_{\mathrm{par}\leftarrow\infty}\|z\|_\infty$ for all $z \in \mathbb{R}^T$.

Assumptions (restated). (A1) (Realizability) For each $u \in \mathcal{U}$ there exists $\theta \in \Theta(S)$ such that $m_{S,\theta}(X) = Y$ for all $(X,Y) \sim \mathcal{D}_u$. (A2) (Continuity) There exists a continuous target mapping $\omega : \mathcal{U} \to \Theta(S)$ on compact $\mathcal{U}$.

We also use the compact image $\mathcal{K} := \omega(\mathcal{U}) \subset \Theta(S)$ and, for $\delta > 0$, the closed δ-thickening $\mathcal{K}^{+\delta} := \{\vartheta \in \mathbb{R}^T : \mathrm{dist}(\vartheta, \mathcal{K}) \le \delta\}$.

A.6 Proof of Theorem 1 (UAT for hypernetworks)

Theorem (Theorem 1, restated with codomain)**.** *Let $\mathcal{U} \subset \mathbb{R}^{d_u}$ be compact and $\omega : \mathcal{U} \to \Theta(S) \subseteq \mathbb{R}^T$ be continuous. For any $\varepsilon > 0$:*

(a) (Unconstrained parameters) *If $\Theta(S) = \mathbb{R}^T$, there exists a ReLU hypernetwork $M_\phi : \mathcal{U} \to \mathbb{R}^T$ such that*

$$\sup_{u\in\mathcal{U}} \|M_\phi(u) - \omega(u)\|_\infty < \varepsilon.$$

(b) (Box-constrained parameters) *If $\Theta(S) = \prod_{t=1}^T [a_t, b_t]$ is a product box, there exists a ReLU hypernetwork $M_\phi : \mathcal{U} \to \Theta(S)$ with the same uniform approximation guarantee.*

More generally, for any closed convex $\Theta(S)$ and any $\eta > 0$, there exists a ReLU hypernetwork $M_\phi^{(\eta)} : \mathcal{U} \to \mathbb{R}^T$ such that $\sup_u \mathrm{dist}(M_\phi^{(\eta)}(u), \Theta(S)) \le \eta$ and $\sup_u \|M_\phi^{(\eta)}(u) - \omega(u)\|_\infty < \varepsilon + \eta$.

Proof. Write $\omega(u) = (\omega_1(u), \dots, \omega_T(u))$ with each $\omega_k \in C(\mathcal{U})$.

Step 1 (coordinatewise approximation). Let $\sigma(z) = \max\{0, z\}$ be ReLU. Since σ is non-polynomial and we allow biases, single-hidden-layer ReLU networks are dense in $C(\mathcal{U})$ under the uniform metric (Leshno et al., 1993, Thm. 1). Thus, for each k there exists a ReLU network $N_k : \mathcal{U} \to \mathbb{R}$ with

$$\sup_{u\in\mathcal{U}} |N_k(u) - \omega_k(u)| < \varepsilon.$$

Step 2 (stacking). Define $M(u) := (N_1(u), \dots, N_T(u)) : \mathcal{U} \to \mathbb{R}^T$ (a vector-valued ReLU network). Then $\sup_u \|M(u) - \omega(u)\|_\infty < \varepsilon$.

Step 3 (codomain enforcement). (a) If $\Theta(S) = \mathbb{R}^T$, take $M_\phi := M$.

(b) If $\Theta(S) = \prod_{t=1}^T [a_t, b_t]$, post-compose with the ***ReLU clamp***

$$\mathrm{clamp}_{[a,b]}(x) \;=\; a + \sigma(x - a) - \sigma(x - b)$$

coordinatewise: $(\Pi(z))_t = \mathrm{clamp}_{[a_t,b_t]}(z_t)$. Then Π is a (shallow) ReLU network, $\Pi \circ M : \mathcal{U} \to \Theta(S)$, and Π is 1-Lipschitz and fixes $\Theta(S)$, hence

$$\sup_{u\in\mathcal{U}} \|\Pi(M(u)) - \omega(u)\|_\infty \;\le\; \sup_u \|M(u) - \omega(u)\|_\infty \;<\; \varepsilon.$$

Set $M_\phi := \Pi \circ M$.

For a general closed convex $\Theta(S)$, let Π_Θ be the Euclidean projection (non-expansive and identity on $\Theta(S)$). Then $\sup_u \|\Pi_\Theta(M(u)) - \omega(u)\|_\infty \le \varepsilon$ and $\Pi_\Theta \circ M$ maps into $\Theta(S)$, although Π_Θ need not itself be exactly representable by a finite ReLU network unless $\Theta(S)$ is polyhedral. For any compact set $\mathcal{C} \supset M(\mathcal{U})$ and any $\eta > 0$, approximate Π_Θ uniformly on $\mathcal{C}$ by a ReLU network P_η within η to obtain $M_\phi^{(\eta)} := P_\eta \circ M$ with the stated properties. □

Context beyond existence. The proof above uses only Leshno et al.'s necessary–and–sufficient density for *non-polynomial* activations with biases. Two later works provide stronger, complementary statements we cite for context:

- **Sonoda & Murata (2017).** Give a *constructive* ridgelet/Radon representation $f(u) = \int a(\theta, b)\, \sigma(\theta^\top u - b)\, d\mu(\theta, b)$ and quantify uniform approximation via quadrature, including explicit control of weight norms. This strengthens classical UAT with an integral formula and error accounting.

- **Yarotsky (2017).** Provides *rates* for deep ReLU approximation on Sobolev classes: for $f \in W_{n,\infty}([0,1]^d)$ there exist networks with depth $O(\log(1/\varepsilon))$ and size $O(\varepsilon^{-d/n}\log(1/\varepsilon))$ achieving $\|N - f\|_\infty \le \varepsilon$, and proves fixed-depth lower bounds $\widetilde{\Omega}\big(\varepsilon^{-1/(2(L-2))}\big)$ for nonlinear C^2 targets. We do not need these rates for the theorem, but they clarify depth/width–error trade-offs.

A.7 Proof of Lemma 1 (Parameter–Lipschitz semantics)

Lemma (Lemma 1, restated with norms). *Let $\Theta \subset \mathbb{R}^T$ be compact and $\mathcal{X} \subset \mathbb{R}^{d_x}$ be bounded. Then there exists $L_\theta = L_\theta(S, \Theta, \mathcal{X}) < \infty$ such that*

$$\sup_{x\in\mathcal{X}} \|m_{S,\theta}(x) - m_{S,\theta'}(x)\|_2 \;\le\; L_\theta \,\|\theta - \theta'\|_{\mathrm{par}} \qquad \forall\, \theta, \theta' \in \Theta.$$

Proof. Let $\Delta_\ell(x) := \|h_\ell(x;\theta) - h_\ell(x;\theta')\|_2$, $\Delta W_\ell := W_\ell - W'_\ell$, $\Delta b_\ell := b_\ell - b'_\ell$. Using ρ's L_ρ-Lipschitzness and submultiplicativity of the operator norm,

$$\Delta_\ell(x) \;\le\; L_\rho \left(\|\Delta W_\ell\|_2 \,\|h_{\ell-1}(x;\theta)\|_2 + \|W'_\ell\|_2 \,\Delta_{\ell-1}(x) + \|\Delta b_\ell\|_2\right). \tag{1}$$

Since $\mathcal{X}$ and Θ are compact and ρ is Lipschitz, there exist finite bounds (by induction)

$$H_\ell \;\ge\; \sup_{\theta\in\Theta}\sup_{x\in\mathcal{X}} \|h_\ell(x;\theta)\|_2, \qquad B_\ell \;\ge\; \sup_{\theta'\in\Theta} \|W'_\ell\|_2. \tag{2}$$

Unrolling equation 1 with equation 2 yields

$$\Delta_M(x) \;\le\; \sum_{\ell=1}^{M} \Big(L_\rho \|\Delta b_\ell\|_2 + L_\rho H_{\ell-1} \|\Delta W_\ell\|_2\Big) \cdot \prod_{j=\ell+1}^{M} (L_\rho B_j). \tag{3}$$

All factors are finite on Θ, hence there is $C_S = C_S(S, \Theta, \mathcal{X}) < \infty$ with

$$\Delta_M(x) \;\le\; C_S \sum_{\ell=1}^{M} (\|\Delta b_\ell\|_2 + \|\Delta W_\ell\|_2) \;\le\; C_S \,\|\theta - \theta'\|_{\mathrm{par}}.$$

Taking $\sup_{x\in\mathcal{X}}$ gives the claim with $L_\theta := C_S$. □

Architectural variants. For residual blocks $h_\ell = h_{\ell-1} + \rho(W_\ell h_{\ell-1} + b_\ell)$, equation 1 becomes

$$\Delta_\ell(x) \le \big(1 + L_\rho \|W'_\ell\|_2\big)\Delta_{\ell-1}(x) + L_\rho H_{\ell-1} \|\Delta W_\ell\|_2 + L_\rho \|\Delta b_\ell\|_2,$$

so the product factor in equation 3 changes to $\prod_{j=\ell+1}^{M} \big(1 + L_\rho B_j\big)$. Convolutions are linear maps; replace $\|W_\ell\|_2$ by the corresponding operator norm.

Certified alternative (LiRPA). Weight-perturbation LiRPA yields a computable constant $\mathrm{Lip}_{\mathrm{LiRPA}}(\Theta, \mathcal{X})$ (for a chosen parameter-space norm) such that

$$\sup_{x\in\mathcal{X}} \|m_{S,\theta}(x) - m_{S,\theta'}(x)\|_2 \le \mathrm{Lip}_{\mathrm{LiRPA}}(\Theta, \mathcal{X}) \,\|\theta - \theta'\|,$$

see Weng et al. (2020), Tsai et al. (2021), Xu et al. (2020). This can tighten L_θ but is not needed for existence.

A.8 Proof of Corollary 1 (Uniform control of semantic error)

Corollary (Corollary 1, restated). *Fix $\delta > 0$ and set $\Theta := \mathcal{K}^{+\delta}$, which is compact. Suppose*

$$\sup_{u\in\mathcal{U}} \|M_\phi(u) - \omega(u)\|_{\mathrm{par}} \;\le\; \varepsilon_\theta \;\le\; \delta.$$

Then

$$\sup_{u\in\mathcal{U}}\sup_{x\in\mathcal{X}} \big\|m_{S,M_\phi(u)}(x) - m_{S,\omega(u)}(x)\big\|_2 \;\le\; L_\theta\, \varepsilon_\theta,$$

where $L_\theta = L_\theta(S, \Theta, \mathcal{X})$ is from Lemma 1.

For multi-class classification with logit outputs in $\mathbb{R}^C$ and ℓ_∞ output norm, if

$$\gamma(u) \;:=\; \inf_{x\in\mathcal{X}_u} \big(f_y(x;\omega(u)) - \max_{y'\neq y} f_{y'}(x;\omega(u))\big) \;>\; 0,$$

then any $\varepsilon_\theta < \gamma(u)/(2L_\theta)$ guarantees that $\arg\max_y f_y(\cdot; M_\phi(u))$ equals $\arg\max_y f_y(\cdot;\omega(u))$ on $\mathcal{X}_u$.

Proof. By construction $\omega(\mathcal{U}) \cup M_\phi(\mathcal{U}) \subseteq \Theta$, so Lemma 1 applies pointwise in u and yields the uniform bound after taking suprema. For the margin statement, let $\delta_{\mathrm{logit}}(x) := \|f(\cdot; M_\phi(u)) - f(\cdot;\omega(u))\|_\infty$. If $\delta_{\mathrm{logit}}(x) \le \delta$, then the margin can shrink by at most 2δ (top logit down by δ, runner-up up by δ). With $\delta = L_\theta \varepsilon_\theta$, requiring $L_\theta \varepsilon_\theta < \gamma(u)/2$ preserves the argmax. □

Using UAT with the parameter norm. If Theorem 1 is invoked with $\|\cdot\|_\infty$ but Corollary 1 is stated in $\|\cdot\|_{\mathrm{par}}$, use norm equivalence: choose the UAT accuracy $\varepsilon = \varepsilon_\theta / c_{\mathrm{par}\leftarrow\infty}$ so that $\|M_\phi(u) - \omega(u)\|_{\mathrm{par}} \leq \varepsilon_\theta$ uniformly in u.

B NEURAL-GRAPH CONSTRUCTION

We convert any PyTorch network into a *directed multi-graph* $G = (V, E)$ whose nodes carry **bias** scalars and whose edges carry **weight** parameters. The rules below are architecture-agnostic (§B.1) and are then specialised to Transformers (§B.2). All code is provided in `neural_graph.py` (Appendix F).

B.1 CORE CONSTRUCTION RULES

1. **Linear layers.** Each weight matrix $W \in \mathbb{R}^{m \times n}$ becomes mn directed edges $(i \to j,\ w_{ji})$; each row-bias b_j is a node.

2. **Residual (skip) connections.** As in Lim et al. (2024); Kofinas et al. (2024), skip edges are dashed, have fixed weight 1, and *do* participate in message passing so that information can bypass intermediate layers.

3. **Normalization layers.** Batch/LayerNorm can be rewritten as $y = \mathrm{diag}(\gamma)\, x + \beta$, hence we add one-to-one edges weighted by γ_i and bias nodes that hold β_i. This treats normalization exactly like a diagonal linear layer and preserves permutation equivariance.

B.2 ATTENTION BLOCKS

A self-attention module is expanded into the sub-graph in Figure 6:

- **Per-head parameter nodes.** For each head h we create $\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h$ nodes that store the corresponding slices of $\mathbf{W}_h^{Q/K/V}$. All heads share a single output-projection node $\mathbf{O}$.

- **Score nodes and `pass_thru` edges.** A dummy $\texttt{Attn\_Score}_h$ node is inserted to link $\mathbf{Q}_h$ and $\mathbf{K}_h$. The dotted red `pass_thru` edges *contain no parameters*; they exist solely to tell the Meta-GNN that $\mathbf{Q}_h$ and $\mathbf{K}_h$ must be *shuffled together*. In other words they allow arbitrary **head permutations**.

- **`align_one2one` edges.** Inside a head, any of the d_{head} feature dimensions may be permuted. We express this by blue dashed `align_one2one` edges that connect corresponding neurons of two layers (e.g. $\mathbf{Q}_h$ row i to $\mathbf{K}_h$ row i). Like `pass_thru`, they have no weights and are masked out during message aggregation; their sole purpose is to inform the permutation-invariant update that **within-head dimensions can be shuffled independently of other heads**.

Remark. Skip-connection and normalization handling is *identical* to the graph Meta-learner recipes of Lim et al. (2024); Kofinas et al. (2024); only the attention-specific `pass_thru`/`align_one2one` mechanism is new, enabling our graphs to remain Π-equivariant for full multi-head Transformers.

B.3 ILLUSTRATIVE GRAPHS

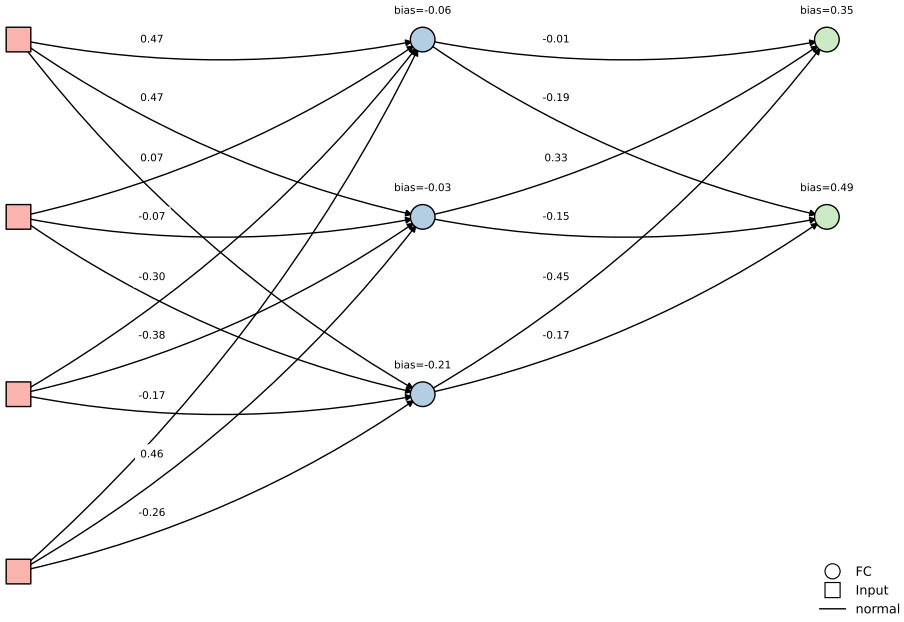


Figure 4: Illustrative neuron-level graph of a *toy* 1-hidden layer MLP. The structure—bias nodes, weight edges, and their grouping—is identical to the subnet used in SUMFIRST-n, but the layer sizes are reduced for clarity.

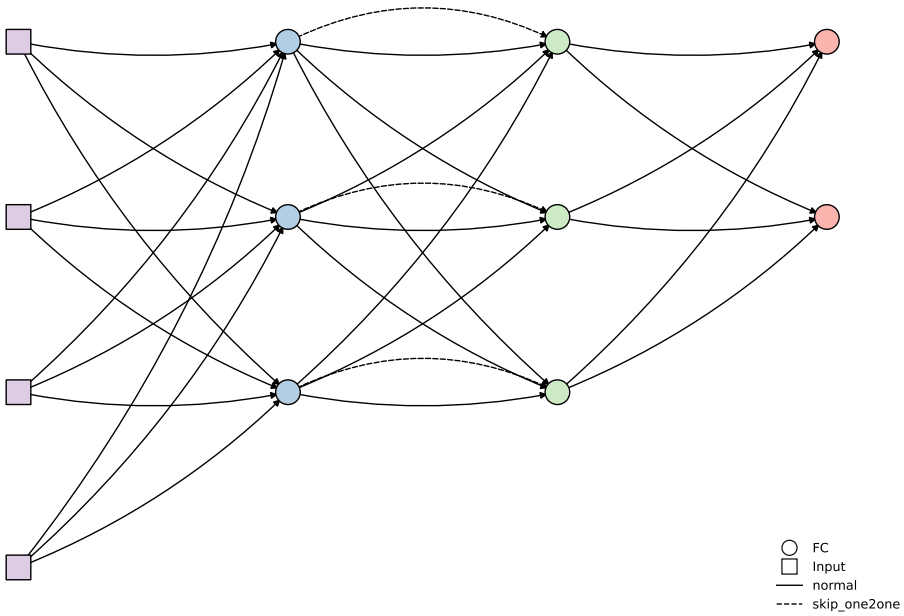


Figure 5: Adding residual (skip) connections to the MLP graph. Dashed edges have fixed weight 1. Layer sizes are again reduced for visual clarity.

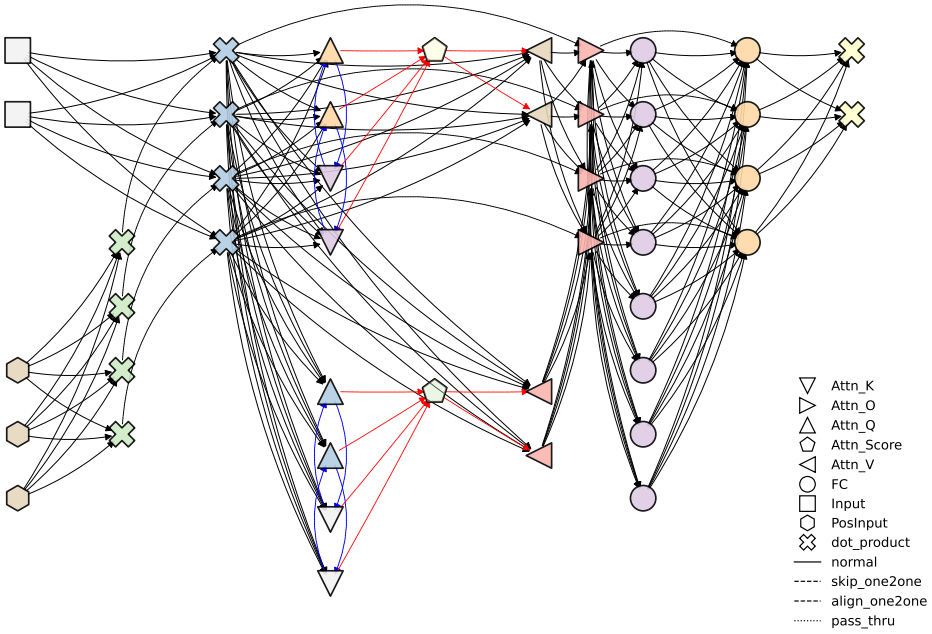


Figure 6: Neuron-graph of a single-layer, two-head Transformer (reduced dimensions). Red dotted `pass_thru` edges allow head permutations; blue dashed `align_one2one` edges allow within-head dimensional shuffling. Both edge types are parameter-free and excluded from message updates.

C Dataset, Architecture and Parameter Details

This appendix records the dataset sizes, exact hyper-parameters, and trainable parameter budgets of every model used in our experiments. All runs use ADAM with learning-rate 10^{-3}, batch size 256 and 50 000 optimizer steps; only the architecture–specific choices differ.

C.1 Dataset Details

Dataset	Train	Test	Val	OOD Test
ADDMOD-p	50 000	1 712	1 997	12 000
SUMFIRST-n	160 000	8 000	10 000	10 000
1D-ARC	144 000	16 000	5 000	5 000

Table 4: Number of examples in each split for the three benchmarks.

1D-ARC meta-split — per-family/parameter composition. We include five families: three *paramless* (`Fill`, `Flip`, `Hollow`) and two *paramful* (`MoveRight`, `RecolorBar`). The generator holds out parameter 7 for OOD evaluation in paramful families.

C.2 Neural program templates (m_θ)

- **SUMFIRST-n:** 1-hidden layer ReLU MLP ($[\,10,\ 16,\ 1\,]$) $\Rightarrow$ 193 weights.
- **ADDMOD-p:** 1-layer causal Transformer (d_{model}=16, n_{heads}=2, n_{ctx}=2) $\Rightarrow$ 6 048 weights.

The templates are *fixed at Meta-train and Meta-test time*; only their weights are synthesized.

C.3 Meta-GNN Component Ablations

To complement the theoretical discussion of structure-awareness in §4.3 and address concerns about isolating Meta-GNN's design choices, we conducted a targeted ablation study on ADDMOD–p with a smaller Meta-GNN (fewer than 50k meta-parameters) for faster sweeps. The goal is to understand how sensitive performance is to (i) initialization strategy, (ii) message aggregation, and (iii) edge directionality and parameter tying, rather than to exhaustively tune every component.

Ablation dimensions. We vary four discrete hyperparameters:

- `init_mode` $\in$ {all, per_group, global}: per-node, per-weight-group, or global initial embeddings for nodes/edges.
- `aggr` $\in$ {attn, avg, sum}: attention-based, mean, or sum aggregation over incoming messages, combined with `use_norm` $\in$ {True, False} to toggle layer normalization in the message-passing block.
- `bidirectional` $\in$ {True, False}: whether to include reverse edges in the neural graph.
- `rev_edge_mode` $\in$ {shared, separate, $--$}: when `bidirectional=True`, reverse edges either share parameters with forward edges (`shared`) or have their own parameters (`separate`); $--$ denotes the purely unidirectional case.

All runs use the same training protocol and target template as in the main ADDMOD–p experiments; results are reported as mean $\pm$ standard deviation over five seeds.

Takeaways. Three trends emerge from Table 5:

- *Group-aware initialization helps, but is not critical.* All three initialization modes (all / per_group / global) with attention aggregation and bidirectional edges achieve similar high accuracies, with per_group slightly ahead. This supports our choice of per-group embeddings as a good trade-off between performance and parameter efficiency.
- *Attention aggregation is consistently strong.* Average and sum aggregation (`avg`/`sum`) remain competitive, but trail attention-based aggregation across settings, especially without normalization. This justifies the use of attention as the default aggregator in the main experiments.
- *Bidirectionality and reverse-edge parameterization give a small but consistent boost.* Adding reverse edges helps over the unidirectional variant, and giving reverse edges separate parameters (`separate`) yields the best overall performance.

init_mode	aggr	use_norm	bidirectional	rev_edge_mode	best_test_qry_acc
all	attn	False	True	shared	0.964 ± 0.016
per_group	attn	False	True	shared	$\mathbf{0.976 \pm 0.029}$
global	attn	False	True	shared	0.969 ± 0.021
per_group	avg	False	True	shared	0.949 ± 0.032
per_group	avg	True	True	shared	0.964 ± 0.013
per_group	sum	False	True	shared	0.947 ± 0.015
per_group	sum	True	True	shared	0.954 ± 0.012
per_group	attn	False	False	–	0.965 ± 0.023
per_group	attn	False	True	separate	$\mathbf{0.979 \pm 0.014}$

Table 5: Meta-GNN ablations on ADDMOD–p with a smaller meta-network. All configurations achieve strong performance; the configuration used in the main paper corresponds to `init_mode` = per_group, `aggr` = attn, `bidirectional` = True, `rev_edge_mode` = separate.

Overall, the differences across configurations are modest: all achieve best_test_qry_acc in the 0.95–0.98 range on ADDMOD–p with a compact Meta-GNN. This suggests that the core structure-aware design (neural graph representation, permutation-equivariant message passing, and group-tied encoders/decoders) is robust, and that our final configuration in the main paper is a reasonable, but not finely tuned, operating point. In the main results we scale Meta-GNN up to roughly match other meta-architectures in parameter count under the same training budget, which slightly lowers absolute accuracies but preserves the same qualitative trends.

C.4 META-LEARNERS

The Meta-learner generates *all* weights of the above template; its own trainable parameters are listed below (they receive gradients during Meta-training).

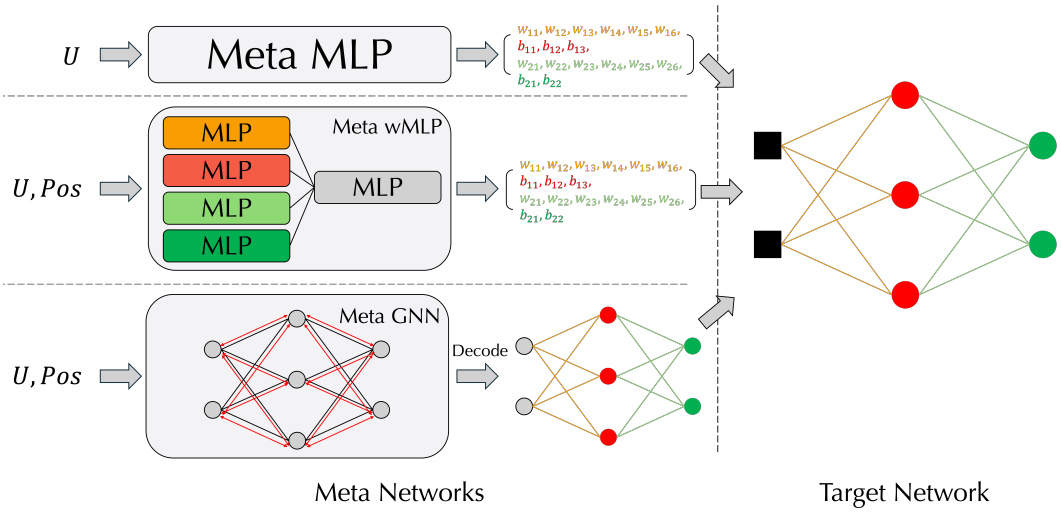


Figure 7: **Meta-learner architectures.** From top to bottom: META-MLP (no structural input), META-gMLP (group & position tags), and META-GNN (full neural-graph connectivity). Boxes denote MLP blocks.

SUMFIRST-n

Model	Hidden configuration	# trainable params
META_MLP	$(32, 128, 512, 256, 128)$	259 650
META_gMLP	groups $(32, 64, 128, 128)$ + mixer (128)	235 235
META_GNN	$d_h = 32,\ k = 4$	189 603
direct MLP (baseline)	$d_h = 226,\ k = 6$	259 449

ADDMOD-p

Model	Hidden configuration	# trainable params
META_MLP	$(32, 32)$	218 176
META_gMLP	groups $(32, 32)$ per group + mixer (16)	237 648
META_GNN	$d_h = 32,\ k = 3$	284 256
direct MLP (baseline)	$d_h = 140,\ k = 9$	218 240
direct Transformer	d_m=64, n_L=4, n_H=4	212 964

1D-ARC

Model	Hidden configuration	# trainable params
META_MLP	$(64, 32, 32)$	589 408
META_gMLP	groups $(32, 4)$ per group + mixer (8)	755 562
META_GNN	$d_h = 32,\ k = 4$	650 018
direct Transformer	d_m=114, n_L=4, n_H=6	649 230

Remarks.

- Parameter counts refer to the *Meta-network* or baseline itself; the weights produced for m_θ are not trainable and therefore do not appear in the optimizer state.
- Budgets for direct baselines were tuned so the largest baseline matches, within $\pm 5\%$, the smallest competing Meta-learner on the same task.
- Compared to a naive fully-connected hypernetwork that maps U directly to a flat list of weights, Meta-GNN uses group-tied encoders/decoders and local message passing on the neural graph. This yields fewer trainable meta-parameters for a given target size and a compute cost that scales with the number of nodes and edges in $G^\star$ rather than the total number of scalar weights. In practice, this makes Meta-GNN more parameter-efficient while preserving or improving OOD performance relative to Meta-MLP.
- All GNNs employ bidirectional edges, attention aggregation, and group-shared MLPs.

Measured cost on one NVIDIA V100 (32 GB).

SumFirst-n (193-weight MLP): Meta-MLP $\approx$ 15 min, Meta-GNN $\approx$ 40 min, VRAM $\approx$ 12 GB.

AddMod-p (6048-weight Transformer): Meta-MLP $\approx$ 30 min, Meta-GNN $\approx$ 90 min, VRAM $\approx$ 20 GB.

1D-ARC (17568-weight Seq2Seq Transformer, on H100-40GB): Meta-MLP $\approx$ 1hr, Meta-GNN $\approx$ 10 hrs, VRAM $\approx$ 30 GB.

Runtime is dominated by executing the synthesized target net each step. Although the Transformer has $\sim 30\times$ more weights than the MLP, the wall-clock gap is only $\sim 2\times$ because kernels for the very small tensors in *SumFirst* are latency-bound, whereas larger tensors in *AddMod* are compute-bound.

D Detailed Performance Analysis

D.1 Results on AddMod-p

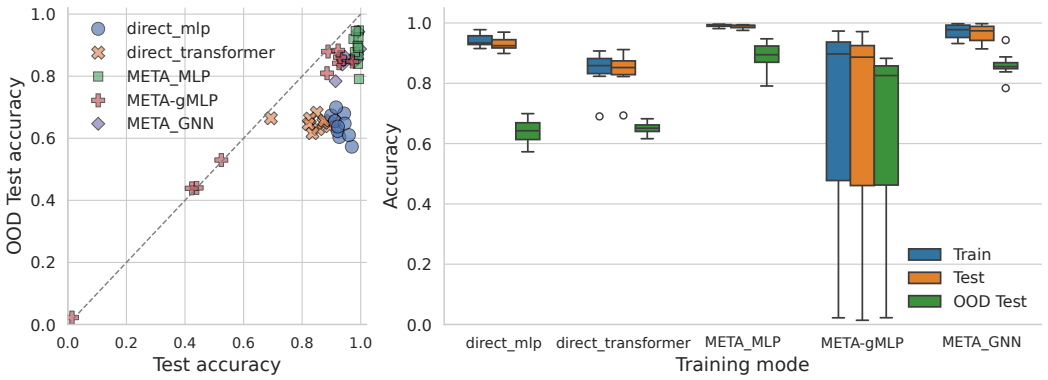
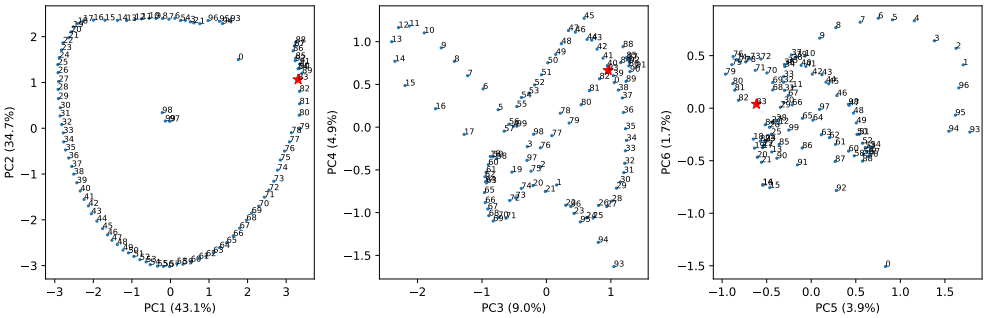


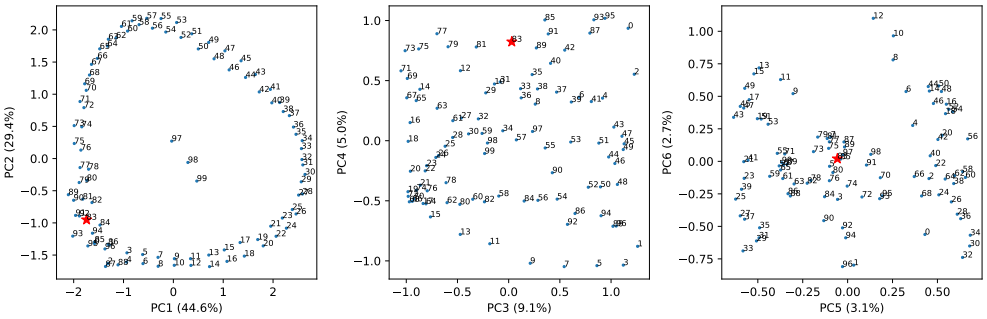
Figure 8: **AddMod-p**. Left: per-seed query (x) vs. test (y) accuracy. Right: distribution over ten seeds. Meta-learners generalize far beyond direct baselines; Meta-GNN combines high accuracy with low variance.

Figure 8 plots query versus test accuracy and the distribution over seeds. All three Meta-learners achieve $\geq 90\%$ accuracy on query examples, demonstrating that they can synthesize complete 6 k-parameter Transformers *in one forward pass*. generalizing to new moduli is harder: the best **direct** baselines reach only 0.65 test accuracy, while Meta-MLP achieves a median 0.89, Meta-gMLP 0.78, and the structure-aware Meta-GNN 0.88 with the lowest variance.

Mechanistic validation. A one-layer Transformer that solves modular addition is expected to discover the *clock* representation (Nanda et al., 2023; Zhong et al., 2024). To test whether our synthesized sub-networks do likewise, we project their token embeddings onto the first two principal components (Figure 9). For an unseen modulus $p = 83$, **Meta-MLP** still exhibits several distinct circular patterns, showing that it has not collapsed its internal states to a unique representation of the computation. **Meta-GNN**, instead, uses its structure-awareness, bidirectional-attention architecture to *canonicalise* the geometry into a **single** clear circle—evidence that it abstracts the task into one consistent internal coordinate system. Such canonicalization is crucial for learning generalizable abstractions and is a desirable property for program synthesis systems. Clock-specific metrics from Zhong et al. (2024) confirm that *all* META models implement valid clock solutions, with Meta-GNN closest to the theoretical optimum.



(a) Meta-MLP, unseen modulus $p = 83$. First two principal components form *several* concentric circles—multiple permutation orbits of the clock solution.



(b) Meta-GNN, unseen modulus $p = 83$. Bidirectional attention collapses the embeddings to a *single* canonical circle, consistent with Meta-GNN's superior zero-shot accuracy.

Figure 9: PCA geometry of the Transformer subnet synthesized for an unseen modulus. Meta-MLP (a) retains multiple permutation orbits; the structure-aware Meta-GNN (b) reduces the search space.

D.2 Results on SumFirst-n

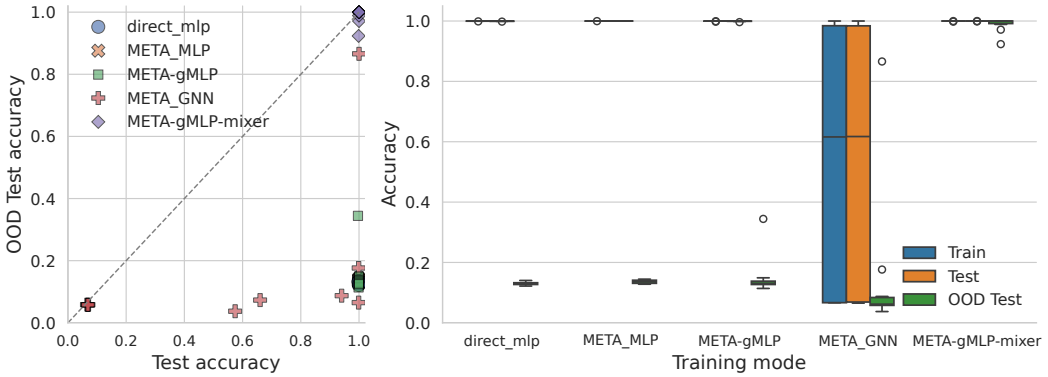


Figure 10: **SumFirst-n**. *Left:* per-seed query (x-axis) versus test (y-axis) accuracy. *Right:* test-accuracy distribution across ten random seeds. Structure-agnostic Meta-MLP behaves like the direct MLP baseline. Introducing weight–group tags (GMLP) occasionally discovers a rule that extrapolates, while the structure-aware Meta-GNN yields both the highest best-case and the most consistent performance. The mixer-free variant GMLP–MX illustrates the effect of an even stronger inductive bias.

Figure 10 compares the four weight generators on a task where many *functionally distinct* parameter settings solve the training data. **Meta-MLP**, devoid of structural cues, mirrors the direct MLP: it fits the support/query splits yet rarely extrapolates to unseen n.

Adding layer–type grouping plus a low-rank mixer (**Meta-gMLP**) helps only sporadically—the median barely moves—but the few runs that *do* generalize indicate that even coarse positional tags can nudge optimization toward a useful pattern.

Structure-aware message passing in **Meta-GNN** makes this nudge systematic: a much larger fraction of seeds exceed the gMLP ceiling and the best models approach 90% test accuracy. Thus the progression MLP $\rightarrow$ gMLP $\rightarrow$ GNN shows that richer structural inductive bias *raises the likelihood* of converging to a rule that transfers, rather than memorising a dataset-specific weight assignment.

Stronger bias via GMLP–MX. Eliminating the mixer freezes each weight group in isolation. This extreme assumption forces the Meta-learner onto a single, easily-interpretable linear-masking strategy and lifts average test accuracy to > 0.98—yet the same rigidity makes the variant fail completely on ADDMOD-p. The contrast underlines a recurring theme: *inductive bias must be strong enough to disambiguate weight space, but not so restrictive that it precludes solutions in other domains*.

D.3 RESULTS ON 1D-ARC

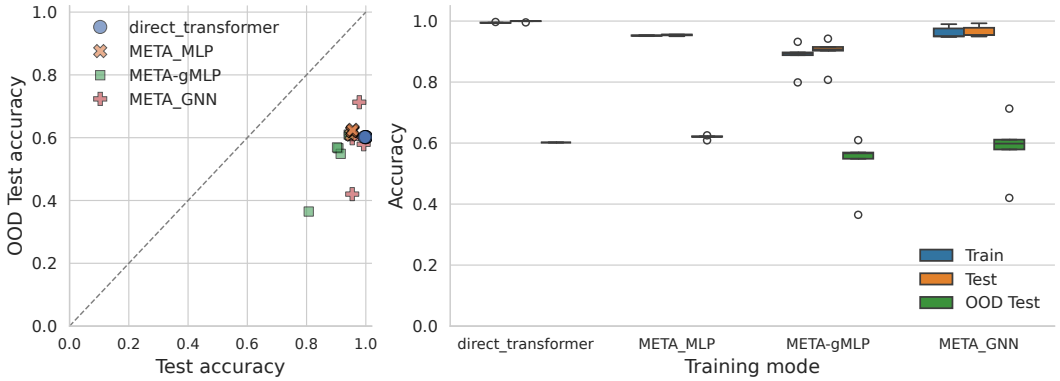


Figure 11: **1D-ARC (mixture)**. *Left:* per-seed query (x-axis) versus test (y-axis) accuracy across families. *Right:* distribution of test accuracy across ten random seeds. Direct Transformers either solve or fail entire families, leading to a highly bimodal distribution. Meta-MLP and Meta-gMLP achieve more stable performance but often collapse on categorical intents such as `RecolorBar`. The structure-aware Meta-GNN attains the most consistent cross-family success, with many seeds generalizing well on `Flip` and `MoveRight`.

Figure 11 shows how different weight generators behave on the multi-family 1D-ARC mixture. Here the challenge is more "real-world": diverse intents, some categorical, others ordinal, all intermixed within the same training distribution.

Direct Transformers exhibit brittle behavior. They memorize input-output mappings for easy families such as `Fill`, `Flip`, or `Hollow`, but fail completely on `MoveRight` (no extrapolation to larger distances) and on `RecolorBar` (no mechanism for categorical generalization). This leads to bimodal seed outcomes—either near-perfect on seen families or near-zero on harder ones.

Meta-MLP and **Meta-gMLP** stabilize training relative to direct Transformers, but still inherit the limitations of their architectures: they extrapolate reasonably on `MoveRight`, thanks to the explicit distance parameter, yet collapse on `RecolorBar`, where the categorical intent provides no ordinal structure.

Meta-GNN again demonstrates the strongest inductive bias. Permutation-aware message passing converts weight-space symmetries into consistent supervision, which pays dividends on `Flip`—a task where collapsing equivalent neuron permutations is critical. Across seeds, Meta-GNN is the only model to consistently achieve high OOD accuracy on both `Flip` and `MoveRight`, while matching or exceeding others on `Fill` and `Hollow`. The sole weak spot remains `RecolorBar`, confirming that categorical intents lacking structure are intrinsically resistant to meta-generalization.

Overall, the distribution on 1D-ARC illustrates a sharper version of the theme seen in SUMFIRST-n: stronger structural bias increases not just the best-case accuracy but the *likelihood* of converging to generalizable rules across diverse families.

D.4 SUMMARY OF FINDINGS

Across ADDMOD-p and SUMFIRST-n the three Meta-learners consistently

1. **Fidelity.** Produce fully functional sub-networks in a *single* forward pass, with best-seed *query* accuracy at or above 99% and mean query accuracy close to that mark for every model.
2. **generalization benefit.** Exceed direct intent$\rightarrow$output baselines on *unseen* intents by $20 - 50$ percentage points in test accuracy.
3. **Role of structure.** Show a clear, monotonic link between stronger structural bias and higher zero-shot performance, in line with the analysis of Section 4.
4. **Robust extrapolation.** Reach, for example, 0.95 test accuracy on moduli never encountered during training (ADDMOD-p) and ≈ 0.87 on list lengths beyond the training range (SUMFIRST-n).

E EMPIRICAL RECOVERY OF THE CLOSED-FORM MAP $p \to W_E, W_U$

This appendix documents—in full detail—the analysis pipeline that lets us **reverse-engineer the exact function the meta-learner uses to turn a prime p into the embedding matrices W_E and W_U.**

E.1 THEORETICAL IDEAL: THE "CLOCK" SOLUTION REVISITED

For ADDMODP the target network has a known closed form (the "clock" algorithm):

1. **Embedding** $W_E : (a, b) \mapsto \big(\cos\tilde{\theta},\ \sin\tilde{\theta}\big)$.
2. **Core (attn+MLP)** performs an *angle–sum* on the two embeddings.
3. **Logit head** outputs $\cos w\big(a + b - c\big)$.
4. With $\tilde{\theta} \ = \ \dfrac{r}{p} + \varphi$.

Because every downstream block is *frequency-agnostic*, a perfect meta-network should learn only

$$p \ \longmapsto \ \big(W_E(p),\, W_U(p)\big),$$

leaving the "angle-sum" section unchanged. This is the hypothesis we set out to verify.

E.2 WHERE DOES THE PRIME INFORMATION LIVE?

Per–index variance. For every scalar weight we compute its standard deviation across all training primes and plot the result as a bar chart (Fig. 12). Nearly the entire parameter vector stays low variance, yet the *same handful of indices* protrudes sharply for *every* prime group we examined. This repeated pattern indicates that a fixed subset of parameters is responsible for carrying the prime signal; the rest of the network is effectively invariant.

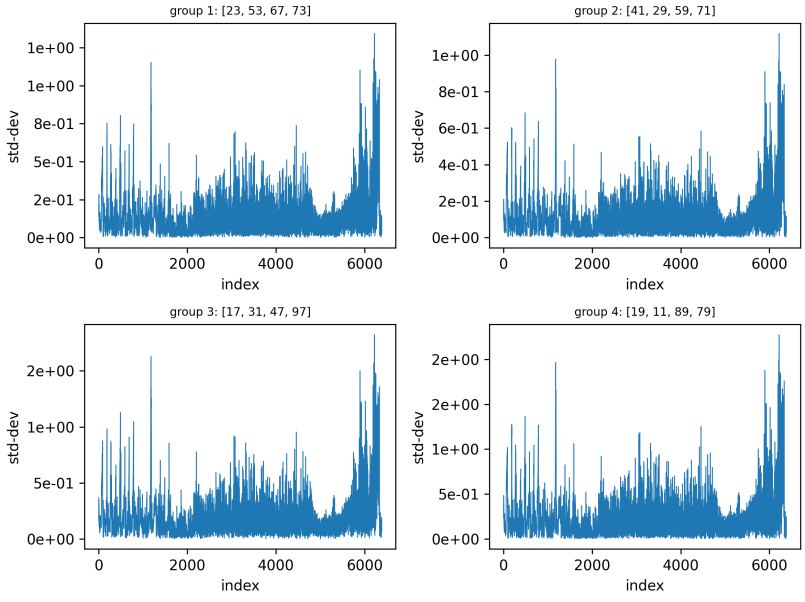


Figure 12: **Per-index standard deviation of weights over primes.** Only a small and *consistent* set of indices shows large variance across groups of p, implying that prime-dependent information is funneled through a tiny portion of the parameter space.

Neural-Graph heat-map. To locate those high-variance indices in the actual architecture we render the checkpoint as a *NeuralGraph* and color each edge (weight matrix W) and node (bias vector b) by the same standard deviation (Fig. 13). Apart from a few isolated pixels inside the MLP blocks, only two contiguous regions blaze red:

- the input embedding matrix W_E,
- the output projection matrix W_U.

This visual confirms the closed-form "clock" hypothesis: all prime-specific variation is concentrated in the mapping $p \to (W_E, W_U)$, while the intermediate angle-sum circuitry remains essentially constant.

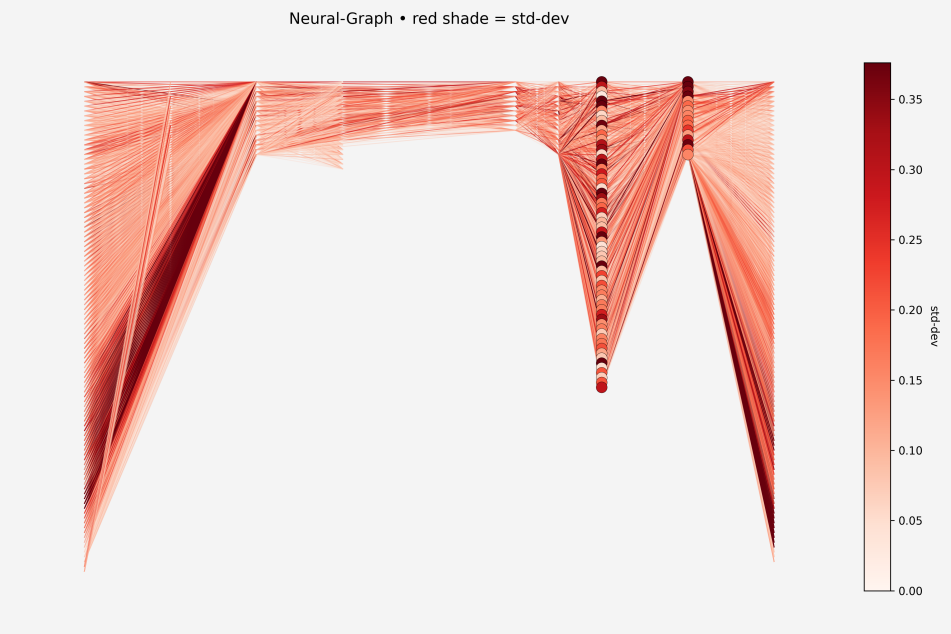


Figure 13: **Prime–sensitivity heat-map on the NeuralGraph.** color encodes the across-prime standard deviation: deep red indicates large change, very light red means almost no change. Edges carry weight matrices W and nodes carry bias vectors b. Aside from a few isolated pixels inside the MLP, only two contiguous regions blaze red—the input embedding W_E (left) and the output projection W_U (right)—echoing the closed-form clock solution. All intermediate edges are essentially prime-agnostic.

E.3 Flatten–then–PCA: turning tensors into points

Flatten step. For every prime p we query the meta learner, obtain $W_E(p) \in \mathbb{R}^{V \times d}$, flatten it to a vector in $\mathbb{R}^{Vd}$ and create a data matrix of shape $|\mathcal{P}| \times (Vd)$.

Why PCA? If each scalar weight obeys

$$w_j(p) \;=\; A_j \cos\theta(p) + B_j \sin\theta(p) + C_j, \quad \theta(p) = \tfrac{r}{p} + \varphi,$$

then *every* flattened vector lives in the 2-D sub-space spanned by the global vectors $A = (A_j)$ and $B = (B_j)$. PCA should return exactly two non-trivial components.

Why FFT? The angle function $\theta(p)$ carries a single frequency $\frac{1}{p}$. If PCA is faithful, the two principal components should share that frequency. FFT on each PC therefore acts as a sanity-check.

E.4 Empirical geometry of the PCA plane

Parabolic curve. Figure 14 plots PC_1 versus PC_2 for all 20 primes. The points lie on an *almost perfect parabola*—visual evidence of a one-dimensional latent path.

Shared frequency profile. The FFT spectra of PC_1 and PC_2 (Table 6) share not only the *dominant* peak (index 1) but also the 2$^{\text{nd}}$ and 3$^{\text{rd}}$ peaks. Higher harmonics are two orders of magnitude smaller. This alignment shows that *both* principal components are driven by exactly the same latent angle and its low-order harmonics, mirroring the multi-frequency "clock" behaviour reported in Nanda et al. (2023); Zhong et al. (2024).

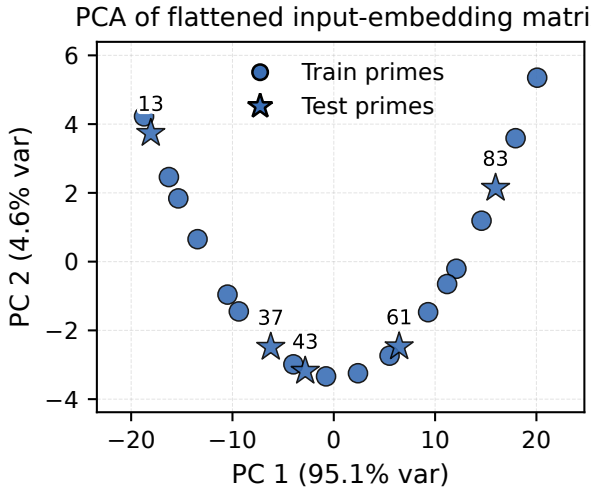


Figure 14: **Global PCA of flattened W_E.** Each dot = one prime, star marks the test prime. The trajectory is a thin parabola, signalling a 1-D latent variable.

Table 6: Top FFT peaks of PC_1 and PC_2. The first three peaks coincide, indicating a shared latent frequency spectrum.

	index	frequency	amplitude
PC_1	**1**	**0.048**	**149.97**
	2	0.095	68.70
	3	0.143	46.99
PC_2	**1**	**0.048**	**39.56**
	2	0.095	6.37
	3	0.143	2.78

E.5 From guesses to the final closed form

Step 1: fit for PC_1. A single–frequency cosine already explains almost all variance:

$$\mathrm{PC}_1(p) \;=\; 23.43\,\cos\!\Big(\tfrac{5.672}{p}+3.631\Big) \quad\Longrightarrow\quad R^2_{\mathrm{PC}_1}=0.945.$$

Step 2: searching for PC_2.

Trial 1 $\mathrm{PC}_2=\sin^2\theta$ $\qquad R^2=0.79$

Trial 2 independent frequency for PC_2 $\qquad R^2=0.79$

Trial 3 $\mathrm{PC}_2=$ **poly$_{\text{even}\le 8}$**$\big(\cos\theta\big)$ + **odd tweaks** $\qquad \mathbf{R^2=0.964}$ ✓

$$\boxed{\begin{aligned}&\mathrm{PC}_1(p)=23.43\,\cos\big(5.672/p+3.631\big), && R^2_{\mathrm{PC}_1}=0.945,\\ &\mathrm{PC}_2(p)=b+B\sum_{k=1}^{4}\beta_k x^{2k}+\sum_{j=0}^{3}\alpha_{2j+1}x^{2j+1}, && R^2_{\mathrm{PC}_2}=0.964,\\ &\qquad x=\cos\big(5.672/p+3.631\big),\ \beta_k=\binom{1/2}{k}(-1)^k.\end{aligned}}$$

E.6 Why an even-degree polynomial? Binomial series & odd terms

Even powers reproduce the binomial series. For $x=\cos\theta$

$$\sqrt{1-x^2}=1-\tfrac{1}{2}x^2-\tfrac{1}{8}x^4-\tfrac{1}{16}x^6-\tfrac{5}{128}x^8-\dots$$

—all *even* powers. On our angle range (0.06–0.52 rad) truncating after x^8 keeps the approximation error below 0.2%. Constraining PC_2 to this form and fitting a single scale B yields the β-ratios

$$\beta_1 = -0.500,\ \beta_2 = -0.125,\ \beta_3 = -0.062,\ \beta_4 = -0.039,$$

in excellent agreement with the binomial coefficients $-\frac{1}{2}, -\frac{1}{8}, -\frac{1}{16}, -\frac{5}{128}$.

Why do *odd* powers appear? PCA enforces zero mean and orthogonality to PC_1. These linear operations transform $\sqrt{1-x^2}$ into $\left[\sqrt{1-x^2} - \text{const}\right] - \alpha x$, which inevitably introduces odd powers when re-expanded in x. Their magnitudes satisfy $|\alpha_{2j+1}/B| \lesssim 10^{-2}$, three orders smaller than the even block, confirming they are statistical artefacts rather than part of the analytic clock function.

Why the meta learner settles on this form. ReLU/tanh networks approximate smooth polynomials far more easily than raw periodic functions. Once the network has produced the cosine feature (PC_1), the cheapest way to supply the complementary sine information is to reuse that feature and pass it through layers that realise the truncated even-degree binomial series, plus the small odd correction required by PCA. This gives a near-perfect fit ($R^2 = 0.964$) without introducing any new latent frequency—exactly what the analytic clock solution predicts.

E.7 Conclusion and outlook

In this appendix we have *manually recovered* a closed-form, symbolic description of the meta-learner's behaviour:

$$p \longmapsto \big(W_E(p),\, W_U(p)\big),$$

where W_E and W_U follow an explicit cosine–and–polynomial law. This shows that the network has indeed learned a **general analytic mapping**, not a table of memorised weights.

Next frontier. Our recovery was hand-crafted; the obvious next step is to *automate* it. A procedure that systematically extracts symbolic expressions from trained neural weights would let us translate *neural programs back to symbolic programs*. Coupled with existing program-synthesis search, such a tool would enable a hybrid *inference-plus-search* pipeline:

1. **Inference stage** A meta-network quickly proposes a neural solution.
2. **Symbolic extraction** An automated analyser converts the weights into a closed-form, human-readable program sketch.
3. **Search / verification** A symbolic synthesiser refines or proves the extracted sketch.

Developing this "neural-to-symbolic bridge" is, we believe, a compelling direction for future work.

F Code

Code repo: https://github.com/chris91219/NeuroP.git

G Third-party Assets and Licences

H Use of AI Assistance

Portions of the manuscript underwent minor language polishing with LLMs (copyediting only). The authors generated all technical content, code, data, and analyses; model suggestions were reviewed and edited by the authors.

Package	Version	Licence	URL / Citation
PyTorch	2.2.1	BSD-style / Apache 2.0	https://pytorch.org
torchmetrics	1.3.2	MIT	https://github.com/Lightning-AI/metrics
NumPy	1.26.4	BSD-3-Clause	https://numpy.org
NetworkX	3.2.1	BSD-3-Clause	https://networkx.org
scikit-learn	1.4.1	BSD-3-Clause	https://scikit-learn.org
matplotlib	3.8.4	PSF-based / MIT	https://matplotlib.org
tqdm	4.66.4	MPL-2.0	https://tqdm.github.io

Table 7: External code dependencies used in our experiments. All are open-source with permissive licences compatible with redistribution of our research code. No external datasets were used; all data are synthetically generated as described in §5.