DTA-GT: Direction- and Topology-Aware Graph Transformer for Neural Network Representation Learning

Yuxiang Zeng[†], **Kun Xie**[†,*], **Yang Wang**[†], **Jigang Wen**[‡], **Xiaocan Li**[†], **Guangxing Zhang**[§], **Gaogang Xie**[§], **Jiannong Cao**[¶]

[†]Hunan University [‡]Hunan University of Science and Technology

[§]Chinese Academy of Sciences [¶]The Hong Kong Polytechnic University

{zengyuxiang@hnu.edu.cn, xiekun@hnu.edu.cn}

Abstract

Architecture attribute predictors reduce the cost of neural architecture search by estimating accuracy or latency from a small set of evaluated candidates. Neural architectures are operation-labeled directed acyclic graphs (DAGs), where edge directions determine computational roles and execution dependencies. Existing graph Transformer predictors incorporate topological and directional information in different ways, but primarily focus on specific mechanisms rather than coordinated encoder-wide modeling of these semantics. We introduce **encoder-wide directional consistency**, a design principle that coordinates the modeling of directed computational semantics across node initialization, global spectral encoding, pairwise interaction, and feature updates. To implement this principle, we propose DTA-GT, a Direction- and Topology-Aware Graph Transformer that integrates direction- and topology-aware node initialization, magnetic Laplacian spectral encoding, direction-dual structural attention, and a Direction-Sensitive MoE-FFN. Across accuracy and latency prediction on NAS-Bench-101 and NAS-Bench-201, DTA-GT consistently outperforms representative RNN-, CNN-, GNN-, Transformer-, and hybrid predictors under limited-budget settings. Ablations and magnetic spectral controls support the contribution of the direction-aware stages and orientation-sensitive spectral information, while mechanistic diagnostics characterize module behavior. These results support encoder-wide directional consistency as an effective design principle for architecture DAG representation.

Code: https://github.com/ZengYuXiang7/DTA-GT

1 Introduction

Deep learning has been widely applied across various fields [1–8]. Neural architecture design shapes a network's computational semantics and execution dependencies, influencing both task performance and execution efficiency. Neural Architecture Search (NAS) automates this design process by searching a candidate architecture space for models that meet performance and deployment constraints [9–14]. However, evaluating candidates during search is expensive: accuracy evaluation typically requires full network training, and latency evaluation requires hardware measurement or profiling. A practical way to reduce this cost is to train predictors on a small subset of evaluated architectures and use them to estimate accuracy or latency for unseen candidates [15–18].

[*]Kun Xie is the corresponding author.

[†]Hunan University, 410082, Changsha, China

College of Computer Science and Electronic Engineering

Ministry of Education Key Laboratory for Supercomputing and Artificial Intelligence Converged Computing

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Table 1: Topology and direction modeling in representative architecture predictors.

Category	Representative Methods	Modeling Paradigm	Topology & Direction Encoding	Directional Features / Relations	Global Directional Spectral Prior	Direction-Specific Updates
CNN-based	[29]	Convolutional tensor encoding	◐	○	○	○
RNN-based	[30–32]	Recurrent sequence encoding	◐	○	○	○
GNN-based	[15, 19–21]	GNN-based message passing	◐	◐	○	◐
Transformer-based	[16, 27, 28]	Topology-aware token-attention	◐	◐	○	○
Hybrid	[17, 18]	GNN–Transformer hybrid	◐	◐	○	◐
Proposed	**DTA-GT**	**Direction- and Topology-Aware Graph Transformer**	●	●	●	●

Note: Symbols summarize the cited representative designs: ● denotes explicit coverage, ◐ denotes partial or varying coverage, and ○ denotes no explicit coverage. The comparison is qualitative and concerns representation design rather than predictive performance.

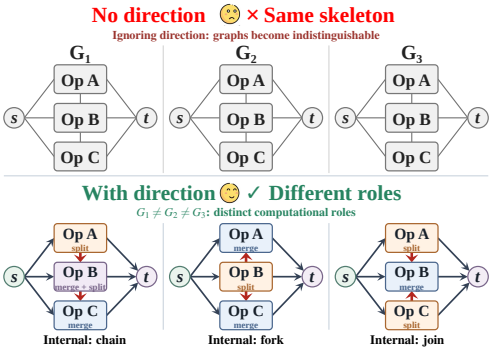


Figure 1: Edge directions distinguish operation order and split/merge roles.

Learning-based predictors rely on architecture representations that capture how operations jointly process information. Each candidate architecture is represented as a directed acyclic graph (DAG), where nodes represent operations and an edge $u \to v$ indicates that the output of operation u feeds into operation v. Directed paths extend these dependencies beyond adjacent operations. The three architectures in Fig. 1 share the same operations and undirected skeleton, but their internal edge directions organize computation differently. In G_1, the internal path follows $A \to B \to C$; in G_2, B feeds both A and C; and in G_3, A and C feed into B. These patterns can affect a network's accuracy by changing how features are transformed and combined to form semantic representations. They can also influence execution latency by shaping computational dependencies: G_1 imposes a serial internal path; G_2 and G_3 permit potential parallelism after and before B, respectively, with B waiting for both branches in G_3. These connections provide a structural basis for predicting accuracy and latency from architecture graphs. Discarding edge directions reduces the three architectures to the same undirected graph, obscuring their different feature flows and execution dependencies. An effective architecture representation should retain these directed computational semantics across local connections and multi-hop paths.

GNN-based predictors exploit graph topology through local message passing and can incorporate edge directions to model dependencies between neighboring operations [15, 19–22]. However, modeling long-range dependencies often requires repeated message passing, which can lead to over-squashing and, with increasing depth, over-smoothing [23, 24]. Pure Transformers enable direct interactions between distant graph elements through global self-attention [25]. Nevertheless, vanilla self-attention lacks inherent inductive biases for graph topology and edge direction, motivating the introduction of graph-specific structural encodings [26]. In neural architecture prediction, Transformer-based and hybrid methods integrate structural and directional information at different levels of encoder design through mechanisms such as topology-aware encodings, structural attention, and bidirectional aggregation [16–18, 27, 28] (see Table 1). Despite these advances, the broader challenge is how to coordinate these mechanisms so that directed computational semantics are modeled consistently throughout the encoder. In architecture DAGs, edge direction determines operation roles, multi-hop dependencies, and distinct upstream and downstream contexts. Because these aspects arise from the same directed computation, a shared design principle should guide how they are modeled during representation construction, interaction, and feature updates.

Guided by this intuition mentioned above, we ask a central question:

> *How can a neural architecture encoder capture DAG topology while consistently modeling the computational semantics of edge direction?*

Our key insight is that directionality in architecture DAGs is more than an auxiliary structural cue: it carries computational semantics that should guide representation learning throughout the encoder. Building on this insight, we propose **encoder-wide directional consistency** as a unifying design

principle that requires node representations, global structural encoding, pairwise interactions, and feature updates to explicitly account for these computational semantics.

Guided by this principle, we develop DTA-GT, a Direction- and Topology-Aware Graph Transformer. Direction- and topology-aware node initialization captures directed node roles, and magnetic Laplacian spectral encoding provides a global prior sensitive to edge orientation. Direction-dual structural attention models pairwise dependencies through separate directional relation channels, and a Direction-Sensitive MoE-FFN adaptively integrates self, predecessor, and successor contexts during feature updates. We evaluate DTA-GT on four prediction settings spanning accuracy and latency prediction over NAS-Bench-101 and NAS-Bench-201. Across multiple limited-budget settings and evaluation metrics, DTA-GT consistently improves over representative methods. Stage-wise ablations assess the contribution of the coordinated direction-aware stages. Magnetic spectral controls isolate the effect of the spectral prior sensitive to edge orientation. Mechanistic diagnostics examine whether the internal behaviors align with the intended directed computational semantics. Our main contributions are:

- We propose **encoder-wide directional consistency** as a unifying design principle that coordinates the modeling of directed computational semantics across the main stages of an architecture encoder.
- We introduce DTA-GT, which integrates direction- and topology-aware node initialization, magnetic Laplacian spectral encoding, direction-dual structural attention, and a Direction-Sensitive MoE-FFN.
- We conduct a unified evaluation on accuracy and latency prediction over standard NAS benchmarks, with stage-wise ablations, magnetic spectral controls, and mechanistic diagnostics to analyze the contribution and behavior of the proposed direction-aware stages.

2 Related Work

Existing architecture predictors include CNN-based, RNN-based, GNN-based, Transformer-based, and GNN–Transformer hybrid models. These families differ in how they represent topology and use edge directions to model dependencies between operations (see Table 1).

CNN- and RNN-based methods. ReNAS [29] uses a CNN predictor with handcrafted architecture features. Peephole [30] and NAO [31] use LSTM encoders to map architecture sequences to vector representations. SemiNAS [32] also uses an LSTM encoder and incorporates unlabeled architectures through semi-supervised training. Nevertheless, these predictors process architectures as tensors or sequences; retaining connectivity information in such inputs does not by itself ensure that graph topology and edge direction are jointly modeled throughout the encoder.

GNN-based methods. BRP-NAS [15], NP [19], NPENAS [22], GATES [20], and DNNPerf [33] model neural architectures as graphs. NNLP [21], the latency prediction component of the NNLQP system, uses graph-based representations to estimate neural network latency. Message passing lets these predictors use graph topology directly and can distinguish incoming from outgoing edges within a local neighborhood. With local message passing, capturing long-range dependencies requires repeated propagation, which can lead to over-squashing and, with increasing depth, over-smoothing [23, 24].

Transformer-based and hybrid methods. NAR-Former [16] encodes operations and their connections as a sequence of tokens. TNASP [27] and PINAT [28] incorporate graph topology through positional encodings based on Laplacian matrices. Among hybrid predictors, NAR-Former V2 [17] uses adjacency to restrict which nodes attend to one another. NN-Former [18] incorporates directional graph relations into attention and feature aggregation. However, the Transformer and hybrid predictors summarized in Table 1 provide only partial coverage of directional modeling across encoder stages. Directional cues introduced at one stage do not by themselves ensure that their computational meaning guides the other stages. The broader design challenge is to coordinate the modeling of operation roles, multi-hop dependencies, and upstream and downstream contexts across representation construction, interaction, and update. In contrast, guided by our proposed encoder-wide directional consistency, DTA-GT makes directed computational semantics an explicit design requirement across node representations, global structural encoding, pairwise interactions, and feature updates.

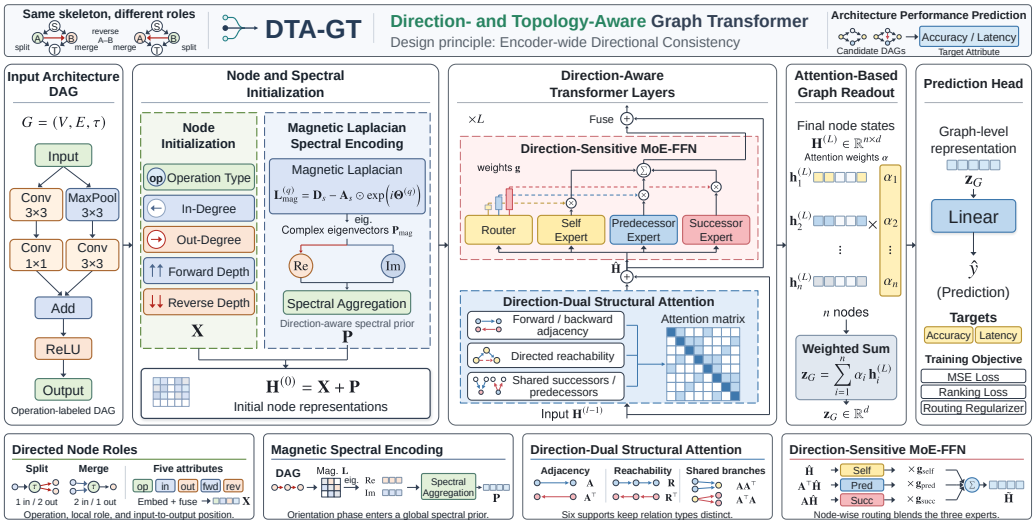


Figure 2: The DTA-GT framework.

3 Problem Formulation and Model Overview

3.1 Problem Formulation

Following established conventions [16–18, 21, 27], let $\mathcal{A}$ be the candidate architecture space. Each architecture $a \in \mathcal{A}$ is represented as an operation-labeled directed acyclic graph (DAG) $G = (V, E, \tau)$, where each node represents a computational operation, such as a convolutional layer or a pooling layer, directed edges represent data flow between operations, and $\tau(v)$ denotes the operation type of node v. We adopt the adjacency convention that $\mathbf{A}_{u,v} = 1$ indicates a directed edge $u \to v$.

Each architecture has a scalar target attribute $y \in \mathbb{R}$, such as accuracy or latency. Given a measured subset $\mathcal{D} = \{(a_i, y_i)\}_{i=1}^{N}$ sampled from $\mathcal{A}$, we learn a predictor $f_\vartheta : \mathcal{A} \to \mathbb{R}$ to estimate attributes for remaining unmeasured candidates in the same architecture space, without fully training or profiling each candidate. We decompose f_ϑ into an encoder h_ϕ and a prediction head p_ψ: $\mathbf{z}_G = h_\phi(G)$ and $\hat{y} = p_\psi(\mathbf{z}_G)$, with $\vartheta = \{\phi, \psi\}$. Our goal is to learn architecture representations that capture operation semantics and DAG topology while consistently modeling directed computational semantics across the encoder.

3.2 Overall Architecture

Fig. 2 illustrates DTA-GT, which maps an architecture DAG G to a graph-level representation $\mathbf{z}_G = h_\phi(G)$ under the encoder-wide directional consistency principle. Specifically, the encoder first initializes node states as $\mathbf{H}^{(0)} = \mathrm{NodeEmbed}(G) + \mathrm{DirSpecEnc}(G)$, combining operation/local directed-role features with a global spectral prior sensitive to edge orientation. It then applies L residual layers [34], each consisting of direction-dual structural attention followed by a direction-sensitive MoE-FFN, using $\mathbf{R}_G$ and $\mathbf{A}$ for relation-aware interaction and predecessor/successor-aware updates, respectively. Finally, attention-based readout produces $\mathbf{z}_G = \mathrm{AttnReadout}(\mathbf{H}^{(L)})$, which is passed to p_ψ for prediction without altering the original architecture DAG.

4 Method

4.1 Direction- and Topology-Aware Node Initialization

Before global spectral encoding and attention-based interaction, each node should carry a basic description of its computation and its directed role in the architecture DAG. Operation labels specify the computation at each node, but they do not distinguish nodes with the same operation type that appear in different local structures or input-to-output positions. Topological-order indices can provide positional cues, but they may introduce ordering artifacts because a DAG can admit multiple valid

topological orders [16, 17, 28]. Consequently, we initialize nodes with topological-order-invariant attributes that encode computation, local directed roles, and input-to-output positions.

Role-aware attributes. Each node v_i is described by five discrete features: operation label $\tau(v_i)$, indegree $\deg^-(v_i)$, outdegree $\deg^+(v_i)$, forward depth $\ell_{\text{fwd}}(v_i)$, and reverse depth $\ell_{\text{rev}}(v_i)$. The forward depth is the maximum length of a directed path from an input node to v_i, and the reverse depth is the maximum length of a directed path from v_i to an output node. The operation label captures computational semantics, indegree and outdegree characterize local merging and branching roles, and $(\ell_{\text{fwd}}, \ell_{\text{rev}})$ locates the node along the directed input-to-output flow. For example, two nodes can have the same forward depth but different longest path lengths to an output. Forward depth alone cannot distinguish these positions, so we also include reverse depth.

So, we embed, concatenate, and project them intuitively:

$$\mathbf{x}_i = \left[\mathbf{e}_{\tau(v_i)}\mathbf{W}_{\text{op}} \,\|\, \mathbf{e}_{\deg^-(v_i)}\mathbf{W}_{\text{in}} \,\|\, \mathbf{e}_{\deg^+(v_i)}\mathbf{W}_{\text{out}} \,\|\, \mathbf{e}_{\ell_{\text{fwd}}(v_i)}\mathbf{W}_{\text{fwd}} \,\|\, \mathbf{e}_{\ell_{\text{rev}}(v_i)}\mathbf{W}_{\text{rev}}\right]\mathbf{W}_{\text{fuse}}, \quad (1)$$

where $\mathbf{e}_{(\cdot)}$ denotes a one-hot vector, the $\mathbf{W}$ terms are learnable projections, $\|$ denotes concatenation, and stacking all $\mathbf{x}_i$ gives $\mathbf{X} \in \mathbb{R}^{n\times d}$.

4.2 Magnetic Laplacian Spectral Encoding

The node initialization $\mathbf{X}$ provides operation semantics and local directed roles, but not a global direction-aware spectral prior over the architecture DAG. Existing predictors usually lack an explicit global direction-aware spectral prior [15–21, 27, 28], while standard Laplacian encodings capture symmetric connectivity but not the asymmetry between $u \to v$ and $v \to u$. Inspired by magnetic Laplacian positional encoding [35], we derive $\mathbf{P}$ from the magnetic Laplacian [36, 37], whose complex phase records edge orientation.

Construction. Given the directed adjacency matrix $\mathbf{A}$, define the symmetrized support $(\mathbf{A}_s)_{u,v} = \frac{1}{2}(\mathbf{A}_{u,v} + \mathbf{A}_{v,u})$, the orientation phase $(\mathbf{\Theta}^{(q)})_{u,v} = 2\pi q(\mathbf{A}_{u,v} - \mathbf{A}_{v,u})$, and the degree $(\mathbf{D}_s)_{u,u} = \sum_{v=1}^{n}(\mathbf{A}_s)_{u,v}$. The magnetic Laplacian is

$$\mathbf{L}^{(q)}_{\text{mag}} = \mathbf{D}_s - \mathbf{A}_s \odot \exp\left(i\mathbf{\Theta}^{(q)}\right). \quad (2)$$

Since $\mathbf{A}_s$ is symmetric and $\mathbf{\Theta}^{(q)}$ is antisymmetric, the magnetic Laplacian is Hermitian. We derive a direction-aware spectral encoding from the magnetic Laplacian.

Two-channel spectral features. We form the retained eigenvector matrix $\mathbf{P}_{\text{mag}} = [\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_k] \in \mathbb{C}^{n\times k}$ from the k low-frequency eigenvectors. For each node u and spectral component r, we separate its real and imaginary parts as $\mathbf{E}_{u,r} = [\text{Re}(\mathbf{P}_{\text{mag}}(u,r)) \| \text{Im}(\mathbf{P}_{\text{mag}}(u,r))]$, giving $\mathbf{E} \in \mathbb{R}^{n\times k\times 2}$. The real and imaginary channels provide a real-valued representation of the complex spectral signal, preserving orientation-dependent phase information.

Spectral aggregation. For each node, SpectralAgg embeds its two-channel spectral components and applies self-attention across the k spectral modes. The outputs are concatenated and projected to d dimensions, producing the learned node encoding $\mathbf{p}_u$. Stacking these encodings gives $\mathbf{P} \in \mathbb{R}^{n\times d}$. The detailed formulation of spectral aggregation is provided in Appendix A.7. The initial encoder states are then obtained by adding this global direction-aware spectral encoding to the node initialization:

$$\mathbf{P} = \text{SpectralAgg}(\mathbf{E}), \qquad \mathbf{H}^{(0)} = \mathbf{X} + \mathbf{P}. \quad (3)$$

4.3 Direction-Dual Structural Attention

Standard self-attention [16, 38] evaluates pairwise interactions from node representations. The magnetic spectral encoding in Sec. 4.2 supplies a node-level global direction-aware spectral prior. Direction-dual structural attention complements this representation with explicit pairwise relation learning, implementing the interaction stage of encoder-wide directional consistency. It organizes pairwise interactions through six directional relation channels. Each channel specifies a distinct structural context for query–key interactions, covering local dependencies, directed reachability, or shared-successor and shared-predecessor structures. Within each channel, content-based attention adaptively weights the admissible keys according to their node representations.

Direction-dual relation channels. We use paired attention channels to distinguish forward from backward dependencies, and to separate node pairs that receive input from the same upstream node from those that feed into the same downstream node. We instantiate three paired relation types, yielding six directional relation channels: (1) **local adjacency**, where $\mathbf{A}$ and $\mathbf{A}^\top$ indicate direct forward and backward dependencies; (2) **directed reachability**, where $\mathbf{R}_{\text{fwd}}(u, v) = 1$ if v is reachable from u, and $\mathbf{R}_{\text{bwd}} = \mathbf{R}_{\text{fwd}}^\top$ captures the reverse relation [39]; and (3) **shared successors/predecessors**, where $\mathbf{A}\mathbf{A}^\top$ captures common successors and $\mathbf{A}^\top\mathbf{A}$ captures common predecessors. Let $\mathbf{R}_G = \{\mathbf{R}^{(r)}\}_{r=1}^{6}$ denote the resulting relation channels. Together, these channels preserve local connectivity, multi-hop directed reachability, and higher-order branch structures as distinct structural contexts.

Relation-constrained attention. For notational brevity, write the input node states of layer l as $\mathbf{H} = \mathbf{H}^{(l-1)} \in \mathbb{R}^{n\times d}$, and let $d_{\text{head}} = d/M$ for M attention heads. For the m-th head, we compute $\mathbf{Q}_m = \mathbf{H}\mathbf{W}_m^Q$, $\mathbf{K}_m = \mathbf{H}\mathbf{W}_m^K$, and $\mathbf{V}_m = \mathbf{H}\mathbf{W}_m^V$ using projection matrices $\mathbf{W}_m^Q, \mathbf{W}_m^K, \mathbf{W}_m^V \in \mathbb{R}^{d\times d_{\text{head}}}$. We use a fixed relation-to-head assignment: in the six-head configuration used in our experiments, head m is associated with directional relation channel $\rho(m) = m$. The assigned relation defines the structural support of that head: we set $\mathbf{B}_m(u, v) = 0$ when $\mathbf{R}^{(\rho(m))}(u, v) > 0$ or $u = v$, and $\mathbf{B}_m(u, v) = -\infty$ otherwise. The attention scores and multi-head output are then computed as

$$\mathbf{S}_m = \frac{\mathbf{Q}_m\mathbf{K}_m^\top}{\sqrt{d_{\text{head}}}} + \mathbf{B}_m, \qquad \text{StructAttn}(\mathbf{H}, \mathbf{R}_G) = \Big[\Big\|_{m=1}^{M} \text{softmax}(\mathbf{S}_m)\mathbf{V}_m\Big]\,\mathbf{W}^O, \tag{4}$$

where $\|$ denotes concatenation over attention heads, $\mathbf{W}^O \in \mathbb{R}^{Md_{\text{head}}\times d}$, and the softmax is applied row-wise over key nodes.

In each attention head, the relation mask specifies which nodes a query node can attend to. Within this set, the query–key scores determine the attention weights. We combine the head outputs to incorporate information from different directional relations into each node representation.

We note that these three relation pairs are a heuristic choice motivated by DAG semantics. The directional consistency principle can guide additional channel pairs that distinguish opposite directions or shared upstream and downstream contexts. For example, dependencies at specific directed path lengths can define new pairs of forward and backward channels (Appendix G).

4.4 Direction-Sensitive Mixture-of-Experts Feed-Forward Network

After structural attention, a standard position-wise FFN updates each node using a shared transformation. However, the relative importance of upstream and downstream information can vary across nodes, and this transformation does not explicitly weight the two directional contexts separately. We therefore replace the standard FFN with a Direction-Sensitive MoE-FFN, which uses three role-specific experts for the node's own state, predecessor context, and successor context. A node-wise routing gate adaptively weights these experts according to the current node state. This keeps the feature-update stage direction-aware without assigning a fixed expert to every node role, since the useful predecessor/successor balance can vary across architectures and prediction targets.

Directional experts. Given the post-attention states $\hat{\mathbf{H}}^{(l)}$, the three expert outputs are

$$\mathbf{H}_{\text{self}} = \hat{\mathbf{H}}^{(l)}\mathbf{W}_{\text{self}}, \qquad \mathbf{H}_{\text{pred}} = \mathbf{A}^\top\hat{\mathbf{H}}^{(l)}\mathbf{W}_{\text{pred}}, \qquad \mathbf{H}_{\text{succ}} = \mathbf{A}\hat{\mathbf{H}}^{(l)}\mathbf{W}_{\text{succ}}. \tag{5}$$

Under the convention $\mathbf{A}_{u,v} = 1$ for $u \to v$, $\mathbf{H}_{\text{self}}$ transforms each node's own state, $\mathbf{A}^\top\hat{\mathbf{H}}^{(l)}$ aggregates predecessor contexts, and $\mathbf{A}\hat{\mathbf{H}}^{(l)}$ aggregates successor contexts. The routing gate produces node-wise expert weights and fuses the expert outputs as

$$[\mathbf{g}_{\text{self}}, \mathbf{g}_{\text{pred}}, \mathbf{g}_{\text{succ}}] = \text{softmax}(\hat{\mathbf{H}}^{(l)}\mathbf{W}_{\text{gate}}),\ \tilde{\mathbf{H}} = \mathbf{g}_{\text{self}} \odot \mathbf{H}_{\text{self}} + \mathbf{g}_{\text{pred}} \odot \mathbf{H}_{\text{pred}} + \mathbf{g}_{\text{succ}} \odot \mathbf{H}_{\text{succ}} \tag{6}$$

where $\mathbf{g}_{\text{self}}, \mathbf{g}_{\text{pred}}, \mathbf{g}_{\text{succ}} \in \mathbb{R}^{n\times 1}$, and the softmax is applied over the three experts. The layer output is $\mathbf{H}^{(l)} = \hat{\mathbf{H}}^{(l)} + \text{ReLU}(\tilde{\mathbf{H}})\mathbf{W}_{\text{proj}}$; $\odot$ denotes broadcast multiplication, and all $\mathbf{W}$ terms are learnable projections.

Routing regularization. To discourage overly sharp routing distributions, we use a negative-entropy regularizer $\mathcal{L}_{\text{aux}} = \frac{1}{n}\sum_{i=1}^{n}\sum_{r\in\{\text{self,pred,succ}\}} g_{i,r}\log(g_{i,r} + \epsilon)$. It encourages more balanced expert weights without imposing role labels on the router.

4.5 Attention-Based Graph Readout

After direction-aware encoding, we summarize the node states into a graph-level representation for attribute prediction. A common approach adds a virtual node to the graph, aggregates information from the other nodes into it, and uses its final state as the graph-level representation. In our framework, adding this node and its connections before graph preprocessing would alter the directional relation channels and magnetic Laplacian of the original architecture. Consequently, we keep the original node set unchanged throughout the encoder and use attention-based readout to aggregate the final node states.

Attention pooling. We use attention-weighted pooling, with $\boldsymbol{\alpha} = \mathrm{softmax}\big(\mathbf{H}^{(L)}\mathbf{r}\big)$ and $\mathbf{z}_G = \sum_{i=1}^{n} \alpha_i \mathbf{H}_i^{(L)}$; $\mathbf{r} \in \mathbb{R}^d$ is a learnable scoring vector, and the softmax is applied over nodes. This pooling yields a graph-level representation $\mathbf{z}_G \in \mathbb{R}^d$ without changing the architecture graph.

4.6 Prediction and Training Objective

Prediction head. Given the graph-level representation $\mathbf{z}_G$, a linear head predicts the target scalar attribute, such as accuracy or latency, as $\hat{y} = p_\psi(\mathbf{z}_G) = \mathbf{w}^\top \mathbf{z}_G + b$, with $\psi = \{\mathbf{w}, b\}$. We keep the predictor head deliberately simple, so most modeling capacity remains in the direction-aware graph encoder rather than in a task-specific regressor. Architecture attribute prediction requires both calibrated value estimates and reliable attribute ranking, so we train with point-wise regression, pairwise ranking, and the routing regularizer introduced in Sec. 4.4.

Training objective. For a mini-batch of size B, the mean squared error (MSE) loss calibrates attribute values as $\mathcal{L}_{\mathrm{mse}} = \frac{1}{B}\sum_{i=1}^{B}(\hat{y}_i - y_i)^2$. To preserve the relative order of candidate architectures under the target attribute, we use a pairwise ranking loss over sampled non-tied architecture pairs $\mathcal{P}$: $\mathcal{L}_{\mathrm{rank}} = \frac{1}{|\mathcal{P}|}\sum_{(i,j)\in\mathcal{P}} \log\big(1 + \exp(-\mathrm{sgn}(y_i - y_j)(\hat{y}_i - \hat{y}_j))\big)$. The final objective is $\mathcal{L}_{\mathrm{total}} = \lambda_{\mathrm{mse}}\mathcal{L}_{\mathrm{mse}} + \lambda_{\mathrm{rank}}\mathcal{L}_{\mathrm{rank}} + \lambda_{\mathrm{aux}}\mathcal{L}_{\mathrm{aux}}$, with $\mathcal{L}_{\mathrm{aux}}$ denoting the negative-entropy routing regularizer for the Direction-Sensitive MoE-FFN; the λ terms control the relative weights of the three objectives.

5 Experiments

5.1 Experimental Setup

We evaluate accuracy and latency prediction on NAS-Bench-101 [40] and NAS-Bench-201 [41]; dataset statistics, sources of latency labels, and splits are provided in Appendix A.2. We compare with ReNAS [29], NAO [31], BRP-NAS [15], NNLP [21], Graphormer [26], TNASP [27], NAR-Former [16], PINAT [28], GraphTrans [42], NAR-Former V2 [17], and NN-Former [18], and report Kendall's τ (KT) and MAPE, where higher τ and lower MAPE are better. We evaluate how well predictors trained on a small labeled subset estimate architecture attributes and recover the ranking of the full candidate pool. Tables 2–4 report five-run means. Appendix H provides the corresponding standard deviations. Baseline sources, reproduction strategies, hardware, and code release information are provided in Appendix A, and DTA-GT hyperparameters and sensitivity analyses for k, L, and d are given in Appendix E.

5.2 Main Results

Across the 16 task–budget combinations in Table 2, DTA-GT achieves the highest Kendall's τ and the lowest MAPE among the evaluated predictors. The design differences are summarized in Table 1. CNN- and RNN-based predictors retain connectivity through tensor or sequence representations without explicitly organizing interactions around directed graph relations. GNN-based predictors model local neighborhoods, with longer-range dependencies obtained through repeated propagation. Transformer-based and hybrid predictors incorporate structural and directional information into selected encoder components. Guided by encoder-wide directional consistency, DTA-GT models directed node roles during initialization, introduces a global spectral prior sensitive to edge orientation, distinguishes directional relations during attention, and adaptively weights upstream and downstream information during feature updates. These stages jointly describe how operations are positioned, connected, and combined in directed computation, providing architecture representations for both accuracy and latency prediction.

Table 2: Main results on NAS-Bench-101/201. Higher Kendall's τ (KT) and lower MAPE are better. Complete results with std are provided in Appendix H.1.

Panel A: Accuracy prediction on NAS-Bench-101

Backbone	Method	Publication	KT ↑ by training budget				MAPE ↓ by training budget			
			0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)
CNN	ReNAS	CVPR 2021	0.0963	0.1240	0.1596	0.5141	2.8409	0.8343	0.5456	0.0660
LSTM	NAO	NeurIPS 2018	0.3395	0.4015	0.4202	0.4472	0.0577	0.0559	0.0493	0.0407
GNN	NNLP	ICPP 2022	0.4452	0.4936	0.4987	0.7809	0.0665	0.0412	0.0399	0.0252
	BRP-NAS	NeurIPS 2020	0.4129	0.4338	0.4549	0.7643	0.0672	0.0548	0.0424	0.0236
Transformer	Graphormer	NeurIPS 2021	0.2229	0.2610	0.2991	0.5610	0.0589	0.0536	0.0477	0.0355
	TNASP	NeurIPS 2021	0.5091	0.5749	0.6404	0.7663	0.0443	0.0393	0.0377	0.0250
	NAR-Former	CVPR 2023	0.4228	0.4986	0.5699	0.8181	0.0538	0.0423	0.0406	0.0257
	PINAT	AAAI 2023	0.3496	0.3994	0.4243	0.7162	0.1339	0.0516	0.0406	0.0271
Hybrid	GraphTrans	NeurIPS 2021	0.4300	0.4438	0.4666	0.7702	0.2735	0.1401	0.1337	0.0252
	NAR-Former V2	NeurIPS 2023	0.5300	0.5500	0.6000	0.8150	0.0900	0.0850	0.0399	0.0220
	NN-Former	CVPR 2025	0.5428	0.5642	0.6057	0.8209	0.0895	0.0907	0.0368	0.0219
	DTA-GT	–	**0.6954**	**0.7278**	**0.7980**	**0.8740**	**0.0327**	**0.0312**	**0.0283**	**0.0202**

Panel B: Latency prediction on NAS-Bench-101

Backbone	Method	Publication	KT ↑ by training budget				MAPE ↓ by training budget			
			0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)
CNN	ReNAS	CVPR 2021	0.5803	0.7127	0.8028	0.9222	0.2232	0.1588	0.1080	0.0405
LSTM	NAO	NeurIPS 2018	0.2569	0.2583	0.3049	0.3964	0.3057	0.3068	0.3062	0.2878
GNN	NNLP	ICPP 2022	0.8056	0.8379	0.8685	0.9190	0.1064	0.0858	0.0695	0.0417
	BRP-NAS	NeurIPS 2020	0.7937	0.8487	0.8702	0.9270	0.1105	0.0811	0.0682	0.0372
Transformer	Graphormer	NeurIPS 2021	0.3582	0.3711	0.4084	0.4467	0.2862	0.2715	0.2630	0.2617
	TNASP	NeurIPS 2021	0.8024	0.8325	0.8734	0.9394	0.1110	0.0897	0.0647	0.0311
	NAR-Former	CVPR 2023	0.8337	0.8670	0.8836	0.9434	0.0881	0.0686	0.0606	0.0287
	PINAT	AAAI 2023	0.8317	0.8621	0.8817	0.9389	0.0860	0.0704	0.0599	0.0315
Hybrid	GraphTrans	NeurIPS 2021	0.8182	0.8619	0.8862	0.9365	0.0949	0.0710	0.0592	0.0329
	NAR-Former V2	NeurIPS 2023	0.7741	0.8314	0.8752	0.9380	0.1238	0.0888	0.0652	0.0317
	NN-Former	CVPR 2025	0.8385	0.8694	0.9082	0.9462	0.0907	0.0662	0.0513	0.0286
	DTA-GT	–	**0.8646**	**0.8930**	**0.9094**	**0.9490**	**0.0758**	**0.0613**	**0.0503**	**0.0271**

Panel C: Accuracy prediction on NAS-Bench-201

Backbone	Method	Publication	KT ↑ by training budget				MAPE ↓ by training budget			
			1% (@156)	3% (@469)	5% (@781)	10% (@1563)	1% (@156)	3% (@469)	5% (@781)	10% (@1563)
CNN	ReNAS	CVPR 2021	0.1816	0.2236	0.4097	0.7130	1.9822	0.8425	0.3444	0.1676
LSTM	NAO	NeurIPS 2018	0.6254	0.6436	0.6445	0.6580	0.0478	0.0454	0.0399	0.0283
GNN	NNLP	ICPP 2022	0.7109	0.7380	0.7824	0.8196	0.0410	0.0245	0.0172	0.0115
	BRP-NAS	NeurIPS 2020	0.6953	0.7177	0.7575	0.7768	0.0629	0.0506	0.0211	0.0201
Transformer	Graphormer	NeurIPS 2021	0.7052	0.7311	0.7461	0.7650	0.0270	0.0185	0.0171	0.0149
	TNASP	NeurIPS 2021	0.7405	0.7770	0.8061	0.8314	0.0302	0.0158	0.0145	0.0110
	NAR-Former	CVPR 2023	0.7540	0.7839	0.8188	0.8554	0.0256	0.0188	0.0141	0.0107
	PINAT	AAAI 2023	0.7027	0.7084	0.7589	0.7920	0.0338	0.0273	0.0269	0.0198
Hybrid	GraphTrans	NeurIPS 2021	0.6957	0.6959	0.7781	0.8258	0.0975	0.0613	0.0493	0.0153
	NAR-Former V2	NeurIPS 2023	0.7418	0.7432	0.8004	0.8391	0.1363	0.0441	0.0340	0.0256
	NN-Former	CVPR 2025	0.7400	0.7420	0.8050	0.8400	0.1350	0.0450	0.0330	0.0260
	DTA-GT	–	**0.8015**	**0.8925**	**0.9138**	**0.9304**	**0.0244**	**0.0125**	**0.0096**	**0.0084**

Panel D: Latency prediction on NAS-Bench-201

Backbone	Method	Publication	KT ↑ by training budget				MAPE ↓ by training budget			
			1% (@156)	3% (@469)	5% (@781)	10% (@1563)	1% (@156)	3% (@469)	5% (@781)	10% (@1563)
CNN	ReNAS	CVPR 2021	0.6166	0.6506	0.6725	0.6863	0.0844	0.0777	0.0707	0.0640
LSTM	NAO	NeurIPS 2018	0.6963	0.6980	0.6984	0.7012	0.0628	0.0602	0.0600	0.0592
GNN	NNLP	ICPP 2022	0.6349	0.6437	0.6527	0.6567	0.0807	0.0726	0.0716	0.0702
	BRP-NAS	NeurIPS 2020	0.6690	0.6861	0.6923	0.6986	0.0717	0.0640	0.0615	0.0594
Transformer	Graphormer	NeurIPS 2021	0.6799	0.6972	0.7030	0.7030	0.0651	0.0600	0.0595	0.0589
	TNASP	NeurIPS 2021	0.6646	0.6873	0.6926	0.6990	0.0713	0.0633	0.0613	0.0595
	NAR-Former	CVPR 2023	0.5916	0.6014	0.6275	0.6522	0.1309	0.1299	0.1229	0.1226
	PINAT	AAAI 2023	0.6859	0.6992	0.6950	0.6996	0.0645	0.0607	0.0602	0.0593
Hybrid	GraphTrans	NeurIPS 2021	0.6441	0.6871	0.6964	0.6993	0.0762	0.0643	0.0612	0.0600
	NAR-Former V2	NeurIPS 2023	0.6237	0.6474	0.6541	0.6781	0.0786	0.0719	0.0699	0.0680
	NN-Former	CVPR 2025	0.6819	0.6891	0.6947	0.6981	0.0695	0.0631	0.0609	0.0594
	DTA-GT	–	**0.8375**	**0.8409**	**0.8435**	**0.8492**	**0.0305**	**0.0293**	**0.0280**	**0.0276**

5.3 Core Design Ablations

Table 3 examines the overall benefit of directional modeling and the contribution of each encoder stage under low-budget settings. The undirected control uses symmetric adjacency and removes directional distinctions from node features, attention relations, and gated feature updates. Magnetic Laplacian encoding is also omitted (Appendix A.8). The "w/o Directed Node Features" variant retains operation embeddings but omits in-degree, out-degree, forward-depth, and reverse-depth

Table 3: Ablation results under low-budget predictor training. Crosses indicate the absence of the corresponding component or directional design in each variant.

Component key	Overall Directionality				Stage Contributions								Readout / Objective			
	Directed		Undirected		w/o Dir. Feat.		w/o Spec.		w/o Attn.		w/o MoE		CLS Readout		MSE Only	
N: Dir. Node Feat.	✓		✗		✗		✓		✓		✓		✓		✓	
P: Spec. Enc.	✓		✗		✓		✗		✓		✓		✓		✓	
A: Struct. Attn.	✓		✗		✓		✓		✗		✓		✓		✓	
F: MoE-FFN	✓		✗		✓		✓		✓		✗		✓		✓	
Metric	KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE
NB101 Acc @100	**0.6954**	**0.0327**	0.5882	0.0408	0.6522	0.0361	0.6894	0.0341	0.6552	0.0331	0.6542	0.0331	0.6701	0.0399	0.6720	0.0356
NB101 Lat @100	**0.8646**	**0.0758**	0.8062	0.1074	0.8482	0.0908	0.8615	0.0806	0.8360	0.1033	0.8495	0.0860	0.8420	0.1203	0.8610	0.0808
NB201 Acc @156	**0.8015**	**0.0244**	0.7215	0.0617	0.7198	0.0729	0.7452	0.0600	0.6123	0.1037	0.7248	0.0418	0.7535	0.0704	0.6937	0.0686
NB201 Lat @156	**0.8375**	**0.0305**	0.8178	0.0349	0.8202	0.0326	0.8192	0.0325	0.8227	0.0317	0.8298	0.0313	0.8209	0.0348	0.8201	0.0321

features. DTA-GT outperforms its undirected counterpart across all reported metrics. Removing directed node features or magnetic spectral encoding reduces performance. This supports using both node roles and the global pattern of edge directions for prediction. Removing structural attention or MoE-FFN also reduces performance. This supports distinguishing directed relations during attention and adaptively combining upstream and downstream information during updates. The size of these effects varies across tasks. Together, these ablations support the value of encoder-wide directional consistency. Attention pooling outperforms CLS readout. It summarizes the existing nodes without changing graph connectivity, preserving the directional relations used by structural attention and magnetic spectral encoding. The MSE-only control also underperforms the proposed training objective. MSE fits numerical attribute values, and the ranking term supervises the relative order of candidate architectures.

5.4 Magnetic Spectral Encoding Analysis

Table 4: Symmetric and magnetic Laplacian spectral encoding controls. Bold marks the best value.

Variant	Sym. Lap.	Mag. Lap.	NB101 Acc @100		NB101 Lat @100		NB201 Acc @156		NB201 Lat @156	
			KT	MAPE	KT	MAPE	KT	MAPE	KT	MAPE
No Directional Encoding	✗	✗	0.5882	0.0408	0.8062	0.1074	0.7215	0.0617	0.8178	0.0349
No Direction + Symmetric Lap.	✓	✗	0.5878	0.0386	0.8071	0.1066	0.7135	0.0485	0.8206	0.0330
Directed with Symmetric Lap.	✓	✗	0.6894	0.0341	0.8615	0.0806	0.7148	0.0792	0.8192	0.0325
Directed with Magnetic Lap.	✗	✓	**0.6954**	**0.0327**	**0.8646**	**0.0758**	**0.8015**	**0.0244**	**0.8375**	**0.0305**

Table 4 compares four configurations to examine the role of direction-aware spectral encoding. The two undirected variants either omit spectral encoding or use symmetric Laplacian encoding. This tests whether global connectivity information helps when direction is ignored. The two directed variants retain the same direction-aware node initialization, structural attention, and MoE-FFN, but use symmetric or magnetic Laplacian encoding. This isolates the benefit of edge orientation in the spectral prior. Symmetric spectral encoding captures connectivity but cannot distinguish different edge orientations on the same undirected skeleton. Magnetic spectral encoding makes the global structural context sensitive to edge direction, complementing the directional information used in node representations, attention, and feature updates. Its better performance across the evaluated tasks supports extending directional consistency to the global spectral stage, rather than relying only on the other direction-aware components. Sensitivity to the magnetic phase parameter q is reported in Appendix B.1 and Table 7.

5.5 Mechanistic Analysis

We analyze the spectral encoding, MoE router, and structural attention to examine whether their internal behavior matches the intended directional design (Figure 3). Reversing all edges changes the magnetic Laplacian at $q = 0.25$ but leaves it unchanged at $q = 0$. This shows that the spectral operator captures edge orientation even when undirected connectivity is unchanged. Routing preferences vary across tasks and node roles, consistent with adapting the balance of upstream and downstream context to different computational roles and prediction targets. Local edge and shared-neighbor relations receive more attention relative to their coverage than directed reachability. These differences illustrate how attention is distributed across direct, multi-hop, and branch-related dependencies within the supplied relation channels. Together, these observations are consistent with the intended roles of global orientation-sensitive encoding, relation-specific interaction, and adaptive directional updates under encoder-wide directional consistency. Tables 2–4 assess the predictive benefits, and Appendix B provides the diagnostic definitions and detailed results.

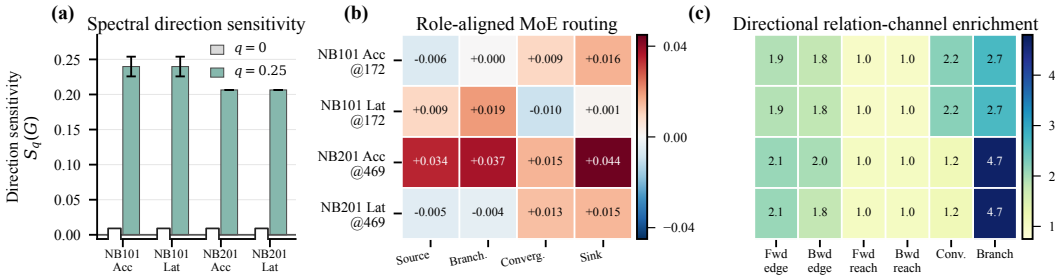


Figure 3: Mechanistic analysis. (a) Magnetic Laplacian sensitivity to edge reversal at $q = 0$ and 0.25. (b) Role-aligned MoE gate contrasts (positive: toward aligned context). (c) Density-normalized relation attention (> 1: above the density baseline).

5.6 Compact Model Evaluation

Table 5: Compact model comparison: parameter count, inference time, and prediction quality under the lowest training budgets (100 for NB101 and 156 for NB201).

Model	Parameters (M)	Inference (ms/sample)	NB101 Acc		NB101 Lat		NB201 Acc		NB201 Lat	
			KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
Graphormer	0.61	0.050	0.2229	0.0589	0.3582	0.2862	0.7052	**0.0270**	0.6799	0.0651
DTA-GT-S	**0.36**	**0.049**	**0.6603**	**0.0359**	**0.8612**	**0.0777**	**0.7114**	0.0486	**0.8183**	**0.0371**

To assess whether DTA-GT's performance gains arise from its directional design or simply a larger parameter count, we evaluate a compact variant, DTA-GT-S. It retains the directional components and uses fewer parameters than Graphormer (Table 5). Even with fewer parameters and comparable inference time, DTA-GT-S generally outperforms Graphormer, showing that these gains do not require a larger parameter budget. DTA-GT-S preserves encoder-wide directional consistency across node representations, global spectral encoding, attention, and feature updates. Together with the ablations in Table 3, these results support the contribution of this coordinated directional design to predictive performance. Appendix D provides the detailed configuration and analysis.

6 Discussion

Limitations. Our evaluation of architecture prediction focuses on NAS-Bench-101 and NAS-Bench-201. Zero-shot cross-space transfer remains unverified, and adapting to other spaces requires aligned operation vocabularies, node granularity, and dependency semantics. Our large-DAG profiling covers up to 32,768 nodes (Appendix F.2). Reachability can still become dense, leading to quadratic storage and attention costs; million-node scaling remains unresolved. The effect of improved predictor ranking on end-to-end NAS remains to be evaluated.

Future work. We will investigate a shared operation-level representation for cross-space pretraining and adaptation, and explore sparse or hierarchical attention with compact reachability representations to support larger computational graphs. We also plan to integrate DTA-GT into NAS pipelines and evaluate the selected architectures and total search cost under a fixed end-to-end evaluation budget. Appendix G elaborates on these directions.

7 Conclusion

We formulated **encoder-wide directional consistency** as an intuitive and unifying design principle for systematically modeling directed computational semantics throughout an architecture encoder. We developed DTA-GT, a Direction- and Topology-Aware Graph Transformer for neural architecture attribute prediction, to realize this principle. Experiments on NAS-Bench-101 and NAS-Bench-201 demonstrate consistent gains in accuracy and latency prediction across limited-budget settings. Ablations and magnetic spectral controls support the contributions of the direction-aware components, and mechanistic diagnostics characterize their learned behavior.

Acknowledgments

The work was supported in part by the National Natural Science Foundation of China under Grants 62321166652 and U25B2038, in part by the Zhejiang Provincial Key Research and Development Program under Grant 2026C02A1244, and in part by the Shaoguan Data Industry Research Institute under Grant 26ND0205.

References

[1] Tairan Huang, Siyu Shang, Qiang Chen, Xiu Su, and Yi Chen. Tools as continuous flow for evolving agentic reasoning. *arXiv preprint arXiv:2605.07339*, 2026.

[2] Yongcong Wang, Chengchao Shen, Guangwei Gao, Wei Wang, Pengwen Dai, Dianjie Lu, Guijuan Zhang, and Zhuoran Zheng. LiBrA-Net: Lie-algebraic bilateral affine fields for real-time 4K video dehazing. *arXiv preprint arXiv:2605.11508*, 2026.

[3] Luankang Zhang, Kenan Song, Yi Quan Lee, Wei Guo, Hao Wang, Yawen Li, Huifeng Guo, Yong Liu, Defu Lian, and Enhong Chen. Killing two birds with one stone: Unifying retrieval and ranking with a single generative recommendation model. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2224–2234, 2025.

[4] Yuxiang Zeng, Yuhui Li, Zhiyu Xia, Zibo Du, Jialin Wang, Ruimin Lian, and Jianlong Xu. Qoseraser: A data erasable framework for web service qos prediction. In *2023 IEEE International Conference on Software Services Engineering (SSE)*, pages 89–97, 2023. doi: 10.1109/SSE60056.2023.00022.

[5] Yuxiang Zeng, Jianlong Xu, Zhuohua Zhang, Caiyi Chen, Qianyu Ling, and Jialin Wang. GATCF: Graph attention collaborative filtering for reliable blockchain services selection in BaaS. *Sensors*, 23(15):6775, 2023. doi: 10.3390/s23156775. URL https://www.mdpi.com/1424-8220/23/15/6775.

[6] Yixuan Ye, Yang Zhang, Liang Peng, Rui Li, Cheng Liu, Si Wu, and Hau-San Wong. Cross-view neighborhood contrastive multi-view clustering with view mixup feature learning. In *2025 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2025.

[7] Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, Fangqiang Ding, and Ismini Lourentzou. PALM: Progress-aware policy learning via affordance reasoning for long-horizon robotic manipulation. *arXiv preprint arXiv:2601.07060*, 2026.

[8] Yifan Shen, Yuanzhe Liu, Jingyuan Zhu, Xu Cao, Xiaofeng Zhang, Yixiao He, Wenming Ye, James M. Rehg, and Ismini Lourentzou. Fine-grained preference optimization improves spatial reasoning in VLMs. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0606.

[9] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pages 678–687. PMLR, 2018.

[10] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*, pages 19–34, 2018.

[11] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[12] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HylVB3AqYm.

[13] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[14] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[15] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. BRP-NAS: Prediction-based NAS using GCNs. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.

[16] Yun Yi, Haokui Zhang, Wenze Hu, Nannan Wang, and Xiaoyu Wang. NAR-Former: Neural architecture representation learning towards holistic attributes prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7715–7724, 2023.

[17] Yun Yi, Haokui Zhang, Rong Xiao, Nannan Wang, and Xiaoyu Wang. NAR-Former V2: Rethinking transformer for universal neural network representation learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 62727–62739. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/c60bd92a01804b7df0540ed7ca2f7c05-Paper-Conference.pdf.

[18] Ruihan Xu, Haokui Zhang, Yaowei Wang, Wei Zeng, and Shiliang Zhang. NN-Former: Rethinking graph structure in neural architecture representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10004–10014, 2025.

[19] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020.

[20] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In *European Conference on Computer Vision*, pages 189–204. Springer, 2020.

[21] Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. NNLQP: A multi-platform neural network latency query and prediction system with an evolving database. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–14, 2022.

[22] Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. NPENAS: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 34(11):8441–8455, 2022.

[23] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2006.05205.

[24] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1905.10947.

[25] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 14582–14595. Curran Associates, Inc., 2022. doi: 10.52202/068431-1060. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5d84236751fe6d25dc06db055a3180b0-Paper-Conference.pdf.

[26] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors,

Advances in Neural Information Processing Systems, volume 34, pages 28877–28888. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f1c1592588411002af340cbaedd6fc33-Paper.pdf.

[27] Shun Lu, Jixiang Li, Jianchao Tan, Sen Yang, and Ji Liu. TNASP: A transformer-based NAS predictor with a self-evolution framework. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15125–15137. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/7fa1575cbd7027c9a799983a485c3c2f-Paper.pdf.

[28] Shun Lu, Yu Hu, Peihao Wang, Yan Han, Jianchao Tan, Jixiang Li, Sen Yang, and Ji Liu. PINAT: A permutation INvariance augmented transformer for NAS predictor. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:8957–8965, Jun. 2023. doi: 10.1609/aaai.v37i7.26076. URL https://ojs.aaai.org/index.php/AAAI/article/view/26076.

[29] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. ReNAS: Relativistic evaluation of neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4411–4420, 2021.

[30] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017.

[31] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

[32] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *Advances in Neural Information Processing Systems*, 33:10547–10557, 2020.

[33] Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. Runtime performance prediction for deep learning models with graph neural network. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice*, pages 368–380, 2023. doi: 10.1109/ICSE-SEIP58684.2023.00039.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[35] Simon Geisler, Yujia Li, Daniel J Mankowitz, Ali Taylan Cemgil, Stephan Günnemann, and Cosmin Paduraru. Transformers meet directed graphs. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 11144–11172. PMLR, 2023. URL https://proceedings.mlr.press/v202/geisler23a.html.

[36] Satoshi Furutani, Toshiki Shibahara, Mitsuaki Akiyama, Kunio Hato, and Masaki Aida. Graph signal processing for directed graphs based on the hermitian laplacian. In Ulf Brefeld, Elisa Fromont, Andreas Hotho, Arno Knobbe, Marloes Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 447–463, Cham, 2020. Springer International Publishing. ISBN 978-3-030-46150-8.

[37] Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. MagNet: A neural network for directed graphs. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27003–27015. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/e32084632d369461572832e6582aac36-Paper.pdf.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[39] Yuankai Luo, Veronika Thost, and Lei Shi. Transformers over directed acyclic graphs. In *Advances in Neural Information Processing Systems*, volume 36, pages 47764–47782, 2023. doi: 10.52202/075280-2070. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/94e85561a342de88b559b72c9b29f638-Abstract-Conference.html.

[40] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019.

[41] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

[42] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 13266–13279. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/6e67691b60ed3e4a55935261314dd534-Paper.pdf.

[43] Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/5fd0b37cd7dbbb00f97ba6ce92bf5add-Abstract-round1.html.

[44] Kesheng Wu and Horst Simon. Thick-restart lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000. doi: 10.1137/S0895479898334605.

A Additional Experimental Details

A.1 Implementation Platform

The NAS-Bench-101 and NAS-Bench-201 attribute-prediction experiments are implemented in PyTorch 2.4.0 and conducted on a workstation running Ubuntu 22.04 with an Intel Core i9-13900K CPU, 64 GB RAM, and a single NVIDIA RTX 4090 GPU with 24 GB memory. The large-DAG scalability experiments are conducted separately on an NVIDIA A800-SXM4-80GB GPU, with the corresponding hardware configuration and timing protocol reported in Appendix F.

A.2 Datasets, Splits, and Training Budgets

NAS-Bench-101 [40] contains 423,624 unique cell-based architectures with at most 7 nodes and 9 edges per cell. NAS-Bench-201 [41] contains 15,625 candidate cells with 4 nodes and 6 edges per cell. For NAS-Bench-201, we represent edge operations as nodes in the predictor input graph, whose encoding topology is fixed across operation assignments. The NAS-Bench-201 authors trained candidate architectures on CIFAR-10 for 200 epochs and recorded their validation accuracies, alongside latency measurements collected with a batch size of 256. These measurements serve as labels for our accuracy and latency prediction tasks, with accuracy averaged over the available benchmark runs for each architecture. NAS-Bench-101 provides accuracy labels but not latency labels; therefore, we measure NAS-Bench-101 latency on an NVIDIA RTX 4090 GPU using batch size 1 to complete the four-task evaluation. Because latency labels are benchmark- and device-specific, latency magnitudes are compared only within the same benchmark setting.

For each task and training budget, each method is evaluated over five runs using a shared set of five random seeds. For each seed, architectures are randomly shuffled to form the training and validation partitions. The partitions vary across seeds, and all methods share the same partitions for a given seed. The @ notation in the result tables denotes the number of architectures used for predictor fitting, and the validation set contains 1,024 architectures. Because the predictor serves the candidate screening stage of NAS, its objective is to score and rank the complete architecture pool and identify promising candidates for costly downstream evaluation. Following the full-pool evaluation setting of NAR-Former [16], we evaluate ranking quality over the full benchmark architecture set, including the training and validation architectures and all remaining candidates. Kendall's τ and MAPE are both computed over this set.

A.3 Fair Comparison and Reproducibility

Within each seed, all methods use the same training and validation partitions, full-set evaluation target, training budgets, and metrics. When reproducing baselines, we use the official implementations when available and preserve their original model configurations; otherwise, hyperparameters are selected by grid search within the reported or standard ranges. DTA-GT uses one fixed default configuration across tasks and budgets. The main comparison and ablation tables report means over five runs; Appendix H also reports the corresponding standard deviations.

A.4 Descriptions of Baselines

We compare DTA-GT with representative RNN-, CNN-, GNN-, Transformer-, and hybrid architecture predictors. NAO uses an LSTM architecture encoder and is included as the RNN-based baseline in Table 2. For methods with public source code, we use the released implementation as the primary reference. For methods for which we did not identify an official public implementation, we reproduce the predictor from the paper description and tune only within the reported or standard ranges under the shared split, budget, and evaluation protocol described above.

RNN- and CNN-based predictors. NAO [31] encodes architectures as operation sequences with an encoder–predictor–decoder framework and performs prediction in a continuous architecture embedding space. ReNAS [29] represents architectures with handcrafted tensor features and trains a relativistic predictor that focuses on ranking architectures rather than only regressing absolute performance. These baselines provide comparisons with sequence- and tensor-based architecture representations.

Table 6: Public source links and reproduction status for compared baselines. "Self-reproduced" denotes paper-based implementation under the shared protocol when no official public code was identified.

Method	Public source link	License/availability	Reproduction note
ReNAS	–	No official public code identified	Self-reproduced (paper).
NAO	renqianluo/NAO	GPL-3.0	Official code used as reference.
BRP-NAS	zheng-ningxin/brp-nas	Apache-2.0	Public code used as reference.
NNLP	ModelTC/NNLQP	Apache-2.0	Public code available.
Graphormer	microsoft/Graphormer	MIT	Official code used as reference.
TNASP	–	No official public code identified	Self-reproduced (paper + supp.).
NAR-Former	yuny220/NAR-Former	Not specified in repository	Official code used as reference.
PINAT	ShunLu91/PINAT	MIT	Official code used as reference.
GraphTrans	ucbrise/graphtrans	Apache-2.0	Official code used as reference.
NAR-Former V2	yuny220/NAR-Former-V2	MIT	Official code used as reference.
NN-Former	XuRuihan/NNFormer	MIT	Official code used as reference.

GNN-based predictors. BRP-NAS [15] uses GCN-based architecture predictors and emphasizes binary relation prediction for comparing candidate architectures. NNLP [21] is a multi-platform neural network latency predictor built on graph-structured model representations. These baselines evaluate local message-passing predictors and latency-oriented graph predictors.

Transformer-based predictors. Graphormer [26] is a general graph Transformer with structural encodings such as degree, shortest-path, and edge-related biases; we adapt it to architecture DAG prediction. TNASP [27] is a Transformer-based NAS predictor that combines operation embeddings, Laplacian positional encoding, and a self-evolution framework. NAR-Former [16] tokenizes operation and topology information into architecture sequences and uses a multi-stage fusion Transformer for holistic attribute prediction. PINAT [28] augments a Transformer-like NAS predictor with permutation-invariance modules and Laplacian positional information. These predictors provide comparisons with alternative Transformer-based representations of architecture structure.

Hybrid predictors. GraphTrans [42] combines a GNN module with global Transformer attention to capture long-range graph context. NAR-Former V2 [17] rethinks Transformer-based neural network representation learning for broader architecture-attribute prediction. NN-Former [18] combines topology-aware token mixing and bidirectional graph-isomorphism feed-forward updates to strengthen neural architecture representations. These hybrid predictors provide comparisons with alternative ways of combining graph structure and attention-based representation learning.

Comparison dimensions in Table 1. Topology & Direction Encoding assesses whether the initial node representations explicitly include topological and directional attributes. Retaining connectivity in the architecture input provides partial coverage when these attributes are not explicitly represented at the node level. Directional Features / Relations covers dedicated encodings of directional node roles or relations, such as incoming and outgoing edges or predecessor and successor relationships. Global Directional Spectral Prior denotes a spectral representation derived from the graph as a whole whose construction depends on edge orientation. Direction-Specific Updates denotes feature propagation or updates that explicitly use directed adjacency or distinguish predecessor and successor contexts.

A.5 Code and Data Access

Code and experimental scripts are available in the public repository at `https://github.com/ZengYuXiang7/DTA-GT`. The repository contains the proposed DTA-GT implementation, experiment scripts, required benchmark data files, and the measured NAS-Bench-101 latency labels used for the four-task evaluation. This appendix provides the corresponding dataset, split, budget, hardware, latency-label, and hyperparameter details needed to interpret and reproduce the reported results.

A.6 Direction-Aware Spectral Encoding Settings

Due to space constraints, the main text presents the combinatorial magnetic Laplacian in Equation (2) and omits the numerical details of degree normalization, graph scaling, and eigenvector phase

normalization. Inspired by Geisler et al. [35], we compute spectral features from its degree-normalized counterpart using the effective parameter $q_G = q/\max\{1, \min(m, n)\}$, where n and m denote the numbers of nodes and directed edges. We use the nominal parameter $q = 0.25$. The same normalized operator and graph-scaled potential are used for the edge-reversal diagnostic in Appendix B.1. We retain the first $k = 4$ eigenvectors ordered by ascending eigenvalues.

To reduce eigenvector ambiguity, we use the numerical normalization convention of Geisler et al. [35]. We first adjust the sign of each eigenvector so that its largest-magnitude real component is nonnegative. We then select the reference node with the largest phase in the first eigenvector and rotate each retained eigenvector to set its phase at this node to zero. The resulting real and imaginary components form the two-channel spectral inputs. This numerical convention reduces phase ambiguity but does not guarantee invariance under arbitrary basis transformations, particularly for repeated or nearly repeated eigenvalues.

A.7 Magnetic Spectral Aggregation Details

For each node u and spectral mode r, we embed the two-dimensional vector $\mathbf{E}_{u,r}$ containing the real and imaginary parts:

$$\mathbf{Z}_{u,r} = f_{\mathrm{elem}}(\mathbf{E}_{u,r}). \tag{7}$$

For each node u, the resulting sequence $\mathbf{Z}_u = [\mathbf{Z}_{u,1}, \ldots, \mathbf{Z}_{u,k}]$ represents its k spectral modes. We aggregate this sequence with self-attention over the spectral dimension:

$$\widetilde{\mathbf{Z}}_u = \mathrm{MHA}(\mathbf{Z}_u, \mathbf{Z}_u, \mathbf{Z}_u). \tag{8}$$

The aggregated spectral signal is then flattened across modes and projected to the model dimension:

$$\mathbf{p}_u = f_{\mathrm{mix}}\Big(\mathrm{Concat}_{r=1}^{k}\, \widetilde{\mathbf{Z}}_{u,r}\Big). \tag{9}$$

Stacking all $\mathbf{p}_u$ gives the spectral encoding $\mathbf{P} \in \mathbb{R}^{n \times d}$, which is added to the node initialization in Eq. 3.

A.8 Undirected Control

We construct the undirected counterpart of DTA-GT using a symmetric adjacency matrix. Node degree and depth features are computed from the graph structure; for each node, the indegree and outdegree inputs have identical values, as do the forward and reverse depth inputs. Magnetic Laplacian encoding is omitted. The six directional relation channels are reduced to three for undirected adjacency, reachability, and shared neighbors. Forward and backward adjacency are merged into one relation, forward and backward reachability into another, and shared successor and shared predecessor relations into a third. The FFN retains a nonlinear transformation of each node's own state and a GNN aggregation branch over the undirected neighborhood. A learned gate combines these two representations without separate predecessor and successor aggregation branches.

B Additional Mechanistic Diagnostics

These diagnostics examine how the proposed modules encode edge orientation, adapt directional context aggregation across node roles, and distribute attention over complementary structural relations. The mechanistic diagnostics use 172 training architectures for NAS-Bench-101 and 469 for NAS-Bench-201. The accompanying phase-parameter experiment uses the lowest training budget. The mechanistic diagnostics characterize internal module behavior and complement the main results and ablations rather than directly attribute performance gains. The magnetic phase diagnostic checks operator-level orientation sensitivity, while the MoE and attention diagnostics describe task-dependent routing and attention allocation over predefined relation channels. The following subsections provide the definitions, numerical results, and per-module views corresponding to Fig. 3.

In Fig. 3, NB101/NB201 denote NAS-Bench-101/NAS-Bench-201, and Acc/Lat denote accuracy/latency. The notation @172 or @469 specifies the number of architectures used to train the predictor. In panel (b), Branch. and Converg. denote branching and converging node roles. In panel (c), Fwd and Bwd denote forward and backward directions, reach denotes reachability, and Conv. and Branch denote relations defined by shared successors and shared predecessors, respectively.

B.1 Magnetic Phase Sensitivity and Parameter Analysis

Graph-scaled phase. As specified in Appendix A.6, the spectral encoder and the edge-reversal diagnostic use the degree-normalized magnetic Laplacian with a graph-scaled phase parameter. In the reported graph-scaled setting, the shared parameter q is converted to the effective phase parameter $q_G = q/c_G$, where $c_G = \max\{1, \min(m, n)\}$ and n and m are the numbers of nodes and edges in the encoded graph. For a directed edge $u \to v$, substituting q_G into Eq. (2) gives conjugate phase factors $e^{i2\pi q_G}$ and $e^{-i2\pi q_G}$, whose separation is

$$\left|e^{i2\pi q_G} - e^{-i2\pi q_G}\right| = 2\left|\sin(2\pi q_G)\right|. \tag{10}$$

The factors are $+i$ and $-i$ when the effective parameter $q_G = 0.25$. With graph scaling, the nominal setting $q = 0.25$ generally gives $q_G \neq 0.25$, so phase separation depends on both q and the graph scale c_G.

Edge-reversal diagnostic. We quantify sensitivity to reversing all edges by

$$S_q(G) = \frac{\|\widetilde{L}_q(A) - \widetilde{L}_q(A^\top)\|_F}{\|\widetilde{L}_q(A)\|_F}. \tag{11}$$

Here, $\widetilde{L}_q$ denotes the normalized magnetic Laplacian constructed with the effective phase q_G defined above. Larger values indicate stronger sensitivity to edge reversal. At $q = 0$, removing the phase term gives zero edge-reversal sensitivity in all four settings. At $q = 0.25$, the reported mean sensitivity is 0.2398 on NAS-Bench-101 and 0.2065 on NAS-Bench-201 for both prediction targets. The variation across NAS-Bench-101 graphs reflects their different topologies, while the fixed NAS-Bench-201 encoding topology gives a constant response across operation assignments.

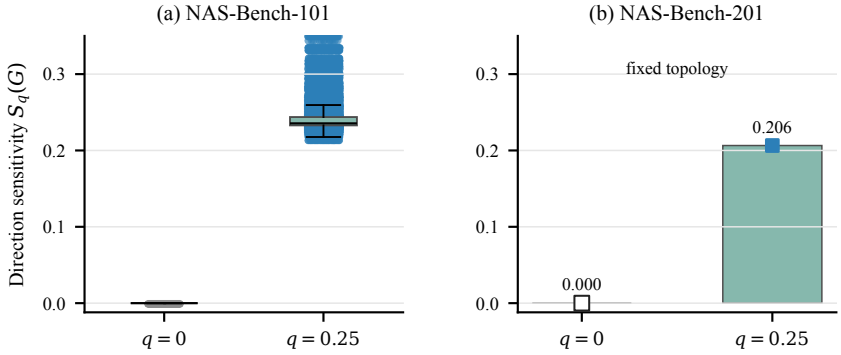


Figure 4: Magnetic spectral response to reversing all edge directions. NAS-Bench-101 yields a distribution over topology-diverse cells, whereas NAS-Bench-201 has a fixed encoding topology and therefore produces a constant response across operation assignments.

Prediction sensitivity to q. We also evaluate $q \in \{0, 0.125, 0.25, 0.5\}$ under the lowest training budget of each task, keeping all other parameters at the shared configuration. Table 7 reports means over five runs. This experiment evaluates the predictive effect of the nominal phase parameter, complementing the operator-level edge-reversal diagnostic above.

Table 7: Effect of the magnetic phase parameter q under the lowest training budget of each task. The tabulated q is the shared nominal parameter before graph scaling. All other parameters are fixed at the shared configuration. Bold indicates the best result for each metric among the tested values.

q	NB101 Acc @100		NB101 Lat @100		NB201 Acc @156		NB201 Lat @156	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
0	0.6894	0.0341	0.8615	0.0806	0.7148	0.0792	0.8192	0.0325
0.125	0.6719	0.0330	0.8642	0.0767	0.7343	0.0528	0.8238	0.0325
0.25	**0.6954**	**0.0327**	**0.8646**	**0.0758**	**0.8015**	**0.0244**	**0.8375**	**0.0305**
0.5	0.6737	0.0337	0.8633	0.0766	0.7386	0.0461	0.8235	0.0323

The shared setting $q = 0.25$ achieves the highest Kendall's τ and the lowest MAPE in all four evaluated tasks among the tested values. These results support its predictive effectiveness within the evaluated settings, while the phase-separation expression above makes its graph-dependent directional effect explicit.

B.2 Detailed MoE Routing by Node Role

We group nodes by directed structural role and summarize the learned weights of the self, predecessor, and successor experts. For source and branching nodes, the role-aligned contrast is $g_{\mathrm{succ}} - g_{\mathrm{pred}}$; for converging and sink nodes, it is $g_{\mathrm{pred}} - g_{\mathrm{succ}}$. A positive contrast indicates greater weight on the context aligned with that role.

Table 8: Mean role-aligned MoE routing contrasts. Positive values indicate greater successor-context weight for source and branching nodes and greater predecessor-context weight for converging and sink nodes.

Setting	Source	Branching	Converging	Sink
NB101 Acc @172	-0.0063	$+0.0002$	$+0.0093$	$+0.0156$
NB101 Lat @172	$+0.0094$	$+0.0192$	-0.0096	$+0.0014$
NB201 Acc @469	$+0.0342$	$+0.0371$	$+0.0152$	$+0.0439$
NB201 Lat @469	-0.0051	-0.0039	$+0.0134$	$+0.0149$

NAS-Bench-201 accuracy prediction exhibits positive role-aligned contrasts for all four roles. The contrasts are 0.0342, 0.0371, 0.0152, and 0.0439 for source, branching, converging, and sink nodes, respectively. Across the remaining settings, the signs and magnitudes vary with the prediction target and node role. These patterns illustrate how the learned router adapts the balance between predecessor and successor context across tasks and structural roles.

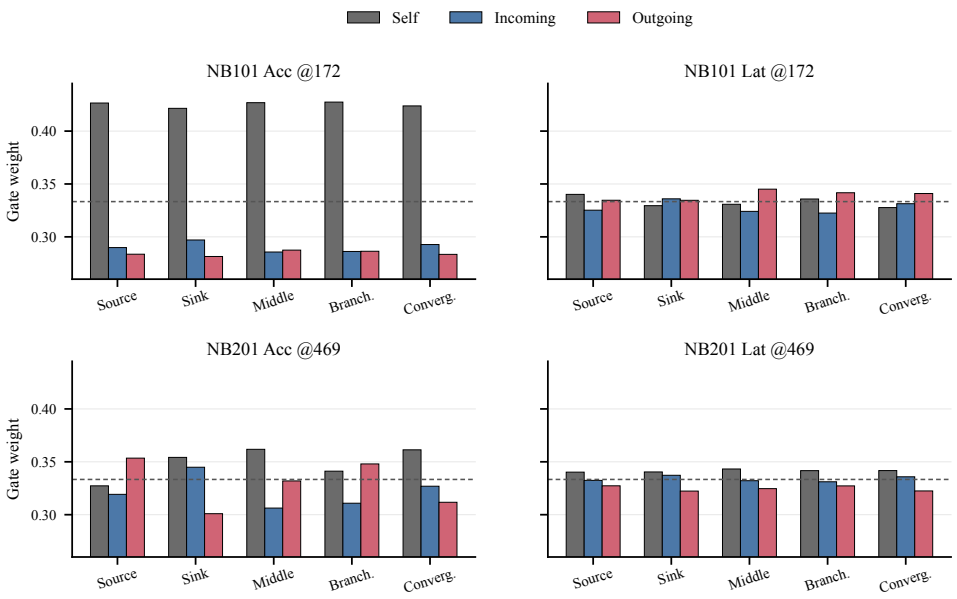


Figure 5: Learned MoE gate weights by directed node role across the four tasks. Incoming/outgoing denote predecessor/successor contexts; dashed lines mark uniform gate weights of $1/3$.

B.3 Detailed Relation-Channel Statistics

For each directional relation r, Figure 6 reports its density, the attention mass assigned to the node pairs covered by that relation, and density-normalized enrichment. Relation density is the fraction of off-diagonal node pairs in the encoded graphs covered by the binary relation indicator M_r, and attention mass is the fraction of off-diagonal attention assigned to those pairs. Enrichment is attention mass divided by relation density, averaged across evaluated runs, layers, and heads. An enrichment

value of 1 corresponds to attention allocation proportional to relation coverage, placing local edges, directed reachability, and shared-neighbor relations on a common coverage-adjusted scale.

Across the four tasks, the six relation channels exhibit distinct enrichment patterns. On NAS-Bench-101, forward and backward edge relations have enrichment values of approximately 1.89–1.91 and 1.78–1.81, respectively, while the directed-reachability relations are close to 0.98. The shared-successor and shared-predecessor relations reach approximately 2.20 and 2.70. On NAS-Bench-201, the edge relations remain enriched, directed reachability is close to 1.04, and the shared-predecessor relation reaches 4.67. These statistics characterize how the learned attention weights distribute mass over the local, multi-hop, and branch-related structural supports supplied to the six directional channels.

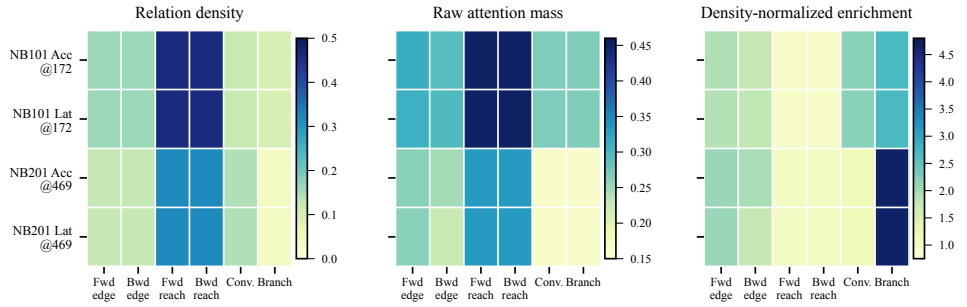


Figure 6: Attention allocation over the six directional relation channels across the four tasks. Relation density is the fraction of valid node pairs covered by a relation, attention mass is the fraction of attention assigned to those pairs, and enrichment is their ratio.

C Complexity Analysis

Let n and m denote the numbers of nodes and directed edges, respectively. Let d denote the hidden dimension, k the number of retained spectral modes, and L the number of encoder layers. We treat the number of attention heads and the six directional relation channels as constants.

C.1 Time Complexity

Degrees, directed depths, and a sparse magnetic Laplacian can be constructed in $O(n+m)$ time from a sparse DAG representation. A dense construction of reachability and common-neighbor relations has $O(n^3)$ worst-case time and $O(n^2)$ storage, and a full dense Hermitian eigendecomposition has the same worst-case time and storage orders. These graph-dependent quantities are computed once and reused during subsequent encoder evaluations.

Aggregating k spectral modes per node costs $O(nkd^2 + nk^2d)$, which becomes $O(nd^2)$ for fixed k. Each encoder layer uses $O(nd^2)$ work for feature projections and $O(n^2d)$ work for dense masked attention. When adjacency is represented densely, the directional context updates remain within the same $O(n^2d + nd^2)$ per-layer bound. The resulting encoder-evaluation cost is

$$T_{\text{eval}} = O\big(nkd^2 + nk^2d + L(n^2d + nd^2)\big)\,. \tag{12}$$

The first-use cost adds graph-dependent construction to encoder evaluation:

$$T_{\text{first}} = T_{\text{relations}} + T_{\text{spectral}} + T_{\text{features}} + T_{\text{eval}}. \tag{13}$$

With fixed k and dense preprocessing, the conservative first-use bound is $O(n^3 + L(n^2d + nd^2))$.

C.2 Space Complexity

Dense relation masks require $O(n^2)$ storage, retained eigenvectors require $O(nk)$, node states require $O(nd)$, and encoder parameters require $O(Ld^2)$. For inference at batch size one, these terms give

$$M_{\text{inference}} = O(n^2 + nk + nd + Ld^2). \tag{14}$$

A conventional backward pass that retains per-layer attention and node activations adds $O(L(n^2+nd))$ storage. The measured GPU memory in Appendix F.2 complements these asymptotic bounds for the profiled implementation.

C.3 Design Choices for Practical Scalability

The analysis above identifies two scale-dependent costs: graph-dependent construction at first use and dense pairwise storage and computation during encoder evaluation. DTA-GT addresses these costs through three complementary execution choices that preserve the same direction-aware representation.

Sparse retained-mode spectral computation. The large-graph spectral path represents the magnetic Laplacian sparsely and uses thick-restart Lanczos to compute only the retained eigenpairs through sparse matrix-vector products in a compact Krylov subspace. The solver computes the spectral modes needed by the encoder without computing the full set of eigenvectors. Its runtime depends on graph sparsity, solver tolerance, and the number of iterations needed for convergence.

Reusable graph-dependent preprocessing. The retained eigenpairs and directional relation masks depend only on the architecture graph, so they are constructed once and reused across encoder layers, training epochs, and repeated predictions of the same architecture. This design separates construction of graph inputs from model evaluation and amortizes structural and spectral construction whenever an architecture is revisited.

Topology- and support-aware relation handling. Deployment-oriented architecture design balances predictive quality against computational cost, making sparse-to-moderately-dense connectivity a practically important regime for support-aware execution. The practical motivation for support-aware relation handling is that computational graphs exhibit structured, architecture-dependent connectivity rather than a single density regime.

ResNet-style architectures introduce selected shortcut edges, whereas DenseNet-style blocks create a denser support pattern by connecting each layer to all preceding layers within a block. Deriving relation supports from the observed topology preserves this variation and provides the structural basis for sparse execution whenever the graph admits it.

The six directional relations expose their supported node pairs explicitly, allowing relation construction to exploit known graph structure. Let S denote the total number of admissible query–key pairs across the six heads, including the self-pairs allowed in each head. A support-indexed attention implementation evaluates only stored supported pairs, yielding an $O(Sd)$ attention term, while sparse predecessor and successor aggregation costs $O(md)$. The explicit relation supports therefore provide the execution interface for support-proportional computation as graph scale increases.

Appendix D reports predictor resource costs on NAS-Bench-201. Appendix F.1 evaluates classification on MalNet-derived DAGs, and Appendix F.2 profiles computational costs on real and generated DAGs. The staged large-DAG profiles separately report costs of constructing graph inputs, model-forward latency, and peak forward GPU memory.

D Computational Efficiency

We construct DTA-GT-S by reducing the hidden dimension to 60 and the encoder depth to four while retaining the direction-aware design. DTA-GT-S contains 0.36M parameters, compared with 0.61M for Graphormer and 6.99M for DTA-GT. We compare their prediction quality under the same lowest-budget setting of each task. Prediction results are averaged over five runs.

Table 9: Parameter counts and inference measurements for the compact-model comparison. DTA-GT-S retains the direction-aware design with reduced width and depth.

Model	Parameters (M)	Inference (ms/sample)
Graphormer	0.61	0.050
DTA-GT-S ($d = 60$, $L = 4$)	**0.36**	**0.049**

Table 5 in the main text reports the corresponding comparison of prediction quality.

Table 10: Prediction quality of Graphormer and DTA-GT-S under the lowest training budget of each task.

Task	Graphormer		DTA-GT-S	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓
NB101 Acc @100	0.2229	0.0589	**0.6603**	**0.0359**
NB101 Lat @100	0.3582	0.2862	**0.8612**	**0.0777**
NB201 Acc @156	0.7052	**0.0270**	**0.7114**	0.0486
NB201 Lat @156	0.6799	0.0651	**0.8183**	**0.0371**

DTA-GT-S uses $19.4\times$ fewer parameters than DTA-GT and remains smaller than Graphormer. It achieves higher Kendall's τ than Graphormer in all four settings and lower MAPE in three of them. Graphormer obtains the lower MAPE on NAS-Bench-201 accuracy prediction, while DTA-GT-S retains the higher ranking correlation in that setting. The recorded inference times are 0.049 ms per sample for DTA-GT-S and 0.050 ms per sample for Graphormer. These results show that the direction-aware design remains effective at a compact parameter scale.

We report parameter count, inference latency, peak GPU memory usage, and training time to characterize the computational cost of representative architecture predictors on NAS-Bench-201. Latency is measured in milliseconds per architecture sample, and training time denotes the wall-clock time required to complete one training run. For each method, latency, memory, and training time are summarized over four NAS-Bench-201 training budgets: 156, 469, 781, and 1563 training samples. When a metric varies across budgets, we report its average value to obtain a compact method-level efficiency profile; for range-valued entries, the midpoint is used for aggregation. The parameter count is reported once since it is independent of the training budget.

Table 11: Comparison of computational efficiency among architecture predictors. Memory denotes peak GPU usage. Latency, memory, and training time are summarized over four NAS-Bench-201 training budgets: 156, 469, 781, and 1563 training samples. For range-valued entries, the midpoint is used.

Method	Params (M) ↓	Avg. latency (ms/sample) ↓	Avg. memory (GB) ↓	Avg. training time (h) ↓
NAO	0.12	0.040	0.030	0.072
ReNAS	1.04	0.050	0.048	0.077
NNLP	0.64	0.050	0.040	0.083
BRP-NAS	1.09	0.050	0.062	0.089
Graphormer	0.61	0.050	0.055	0.101
TNASP	0.61	0.050	0.055	0.101
GraphTrans	0.93	0.053	0.073	0.123
DTA-GT	**6.99**	**0.121**	**0.320**	**0.180**
NAR-Former V2	8.58	0.087	0.355	0.188
NN-Former	4.93	0.120	0.215	0.256
NAR-Former	4.82	0.135	0.180	0.284
PINAT	0.19	0.210	0.037	0.366

DTA-GT uses 6.99M parameters and has an average inference latency of 0.121 ms per sample under this NAS-Bench-201 profile. Its average training time is 0.180 h, compared with 0.188 h for NAR-Former V2, 0.256 h for NN-Former, and 0.284 h for NAR-Former. This resource profile complements the predictive results by quantifying the computational cost of the direction-aware design.

E Design Rationale and Parameter Sensitivity

DTA-GT uses three structurally defined update paths for the node's own state, predecessor context, and successor context. The number of experts follows this decomposition of directed computational context. A node-dependent router learns adaptive mixing weights over the three contexts from the

current node representation. This allows each node to combine its local state with upstream and downstream information according to the prediction task.

The four directional stages provide complementary information throughout the encoder. Node initialization represents local directed roles, and magnetic spectral encoding supplies a global orientation-sensitive prior. Structural attention organizes interactions through directed node-pair relations. The MoE-FFN adapts each node update to its self, predecessor, and successor contexts. Together, these stages coordinate the modeling of directed computational semantics across representation construction, interaction, and feature updates.

Source and branching nodes distribute information along downstream paths, whereas converging and sink nodes summarize upstream computation. These roles motivate separate predecessor and successor contexts, with the learned gate adapting their relative contributions at each node. The routing diagnostics in Appendix B examine how these weights vary across node roles and prediction targets.

The routing regularizer $\mathcal{L}_{\text{aux}}$ in Section 4.4 is the negative entropy of the routing distribution; minimizing it with a positive auxiliary weight encourages higher entropy and discourages overly concentrated expert assignments.

Baseline sources and reproduction strategies are described in Appendix A. The encoder blocks use standard layer normalization. DTA-GT is optimized with AdamW using a batch size of 128, a learning rate of 1×10^{-4}, and weight decay of 0.01 for at most 3000 epochs. For both accuracy and latency prediction, early stopping monitors validation Kendall's τ with a patience of 100, and evaluation uses the checkpoint with the highest validation τ. The learning rate is linearly warmed up over the first 10% of training steps and then follows a cosine decay schedule. The default DTA-GT model uses embedding dimension $d = 180$, backbone depth $L = 10$, and loss weights $\lambda_{\text{mse}} = 1$, $\lambda_{\text{rank}} = 0.2$, and $\lambda_{\text{aux}} = 0.001$. We chose the default configuration in advance. We used the model sizes and optimizer settings of NAR-Former [16], NAR-Former V2 [17], and NN-Former [18] as references. We use the same configuration across all tasks and budgets without task- or budget-specific tuning.

The shared configuration is $(q, k, L, d) = (0.25, 4, 10, 180)$ for both benchmarks, both prediction targets, and all reported NAS training budgets. The q sensitivity experiment is reported alongside the magnetic mechanism analysis in Appendix B.1 and Table 7. That experiment and the k, L, and d experiments below assess sensitivity around the configuration fixed a priori; we retain the same defaults for the main comparisons without task- or budget-specific parameter selection. Each experiment changes one parameter while keeping the remaining settings fixed. We evaluate $k \in \{1, 2, 4\}$, $L \in \{4, 7, 10, 12\}$, and $d \in \{150, 180, 210\}$ under the lowest training budget of each task. The tables below report means over five runs. Bold indicates the best result for each metric among the tested values.

E.1 Retained Spectral Modes

After decomposing the magnetic Laplacian, we order the eigenvectors by ascending eigenvalue and retain the first k low-frequency modes, giving $\mathbf{P}_{\text{mag}} \in \mathbb{C}^{n \times k}$. Each retained mode contributes real and imaginary channels that preserve its orientation-dependent phase information, and the spectral aggregation module combines the resulting k components into the node representation. The parameter k therefore controls the resolution of the global spectral basis together with its storage and aggregation cost. We use $k = 4$ as a compact basis for the evaluated architecture graphs and compare it with $k = 1$ and $k = 2$ below.

Table 12: Effect of the number of retained magnetic spectral modes k under the lowest training budget of each task. All other parameters are fixed at the shared configuration. Bold indicates the best result for each metric among the tested values.

k	NB101 Acc @100		NB101 Lat @100		NB201 Acc @156		NB201 Lat @156	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
1	0.6785	0.0337	0.8606	0.0795	0.7383	0.0481	0.8159	0.0343
2	0.6831	0.0335	0.8625	0.0781	0.7406	0.0429	0.8199	0.0353
4	**0.6954**	**0.0327**	**0.8646**	**0.0758**	**0.8015**	**0.0244**	**0.8375**	**0.0305**

Retaining four spectral modes achieves the highest Kendall's τ and the lowest MAPE in all four evaluated tasks among the tested values of k. For NAS-Bench-201 accuracy prediction, increasing k from 1 to 4 changes Kendall's τ from 0.7383 to 0.8015 and MAPE from 0.0481 to 0.0244. These results support the four-mode representation as the shared compact spectral basis for the evaluated benchmarks and prediction targets.

E.2 Model Depth and Width

The encoder depth L and hidden dimension d form the two standard model-scaling axes considered in our controlled sweeps. The former changes the number of stacked directional encoder blocks, while the latter changes the width of node states and learnable projections. Both parameters affect representational capacity and computation, and we vary them one at a time while keeping the remaining configuration fixed.

Depth. Each DTA-GT encoder layer composes direction-dual structural attention with the node-adaptive self, predecessor, and successor context update. The depth L therefore determines how many times pairwise structural interaction and direction-sensitive feature updating are repeated before graph readout. Increasing L expands the sequence of directional transformations as well as the corresponding computation. We use $L = 10$ in the shared configuration and compare it with $L \in \{4, 7, 12\}$ below.

Table 13: Effect of encoder depth L under the lowest training budget of each task. All other parameters are fixed at the shared configuration. Bold indicates the best result for each metric among the tested values.

L	NB101 Acc @100		NB101 Lat @100		NB201 Acc @156		NB201 Lat @156	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
4	0.6662	0.0409	0.8623	0.0803	0.7303	0.0645	0.8219	0.0346
7	0.6731	0.0368	0.8629	0.0796	0.7420	0.0449	0.8238	0.0335
10	**0.6954**	**0.0327**	**0.8646**	**0.0758**	**0.8015**	**0.0244**	**0.8375**	**0.0305**
12	0.6765	0.0333	0.8603	0.0827	0.7387	0.0437	0.8226	0.0332

Increasing the encoder depth from 4 to 10 improves Kendall's τ and reduces MAPE in all four evaluated tasks. At $L = 12$, Kendall's τ decreases and MAPE increases in all four tasks relative to $L = 10$, showing that additional depth does not continue the improvement observed up to ten layers. The shared ten-layer configuration therefore gives the best predictive results among the evaluated depths across both benchmarks and prediction targets.

Width. The hidden dimension d sets the width of the node states and the attention and directional-context projections throughout the encoder. We use $d = 180$ in the shared configuration and compare it with $d = 150$ and $d = 210$ below.

Table 14: Effect of hidden dimension d under the lowest training budget of each task. All other parameters are fixed at the shared configuration. Bold indicates the best result for each metric among the tested values.

d	NB101 Acc @100		NB101 Lat @100		NB201 Acc @156		NB201 Lat @156	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
150	0.6793	0.0337	0.8643	0.0769	0.7430	0.0486	0.8244	0.0336
180	**0.6954**	**0.0327**	**0.8646**	**0.0758**	**0.8015**	**0.0244**	**0.8375**	**0.0305**
210	0.6734	0.0332	0.8583	0.0804	0.7514	0.0444	0.8185	0.0337

The shared width $d = 180$ achieves the highest Kendall's τ and the lowest MAPE in all four evaluated tasks. Increasing the width to $d = 210$ lowers Kendall's τ and increases MAPE in all four tasks relative to $d = 180$. Across the evaluated settings, $d = 180$ therefore provides the strongest overall predictive performance among the tested widths.

F Evaluation on Larger DAGs

F.1 Predictive Accuracy on MalNet-Derived DAGs

We further evaluate the direction-aware encoder on larger DAGs derived from the five-class MalNet-Tiny graph-classification dataset [43]. The derived graphs contain 5–4,993 nodes, with a mean of 1,407 nodes. This experiment examines the effectiveness of the directional design on larger graphs under limited training supervision.

We contract each strongly connected component into a node to obtain a condensation DAG. Each node receives a shared operation token, so the compared predictors must derive their graph representations from the structure of the resulting DAG. All compared methods use the same derived graphs and data partitions.

We use a stratified 5%/5%/90% training/validation/test split and fit each model on the 5% training partition. All classification metrics are computed on the held-out 90% test partition. The five classes contain equal numbers of graphs, and stratification preserves their proportions in each partition. We compare DTA-GT with BRP-NAS, Graphormer, and NN-Former and report accuracy, macro-F1, macro-precision, and macro-recall. Table 15 reports the mean over five runs.

Table 15: Classification results on DAGs derived from MalNet-Tiny using a stratified 5%/5%/90% training/validation/test split. All methods use the same derived graphs and partitions. Values are mean percentages.

Model	Accuracy	Macro-F1	Macro-Precision	Macro-Recall
BRP-NAS	57.0	53.1	56.9	57.0
Graphormer	69.5	67.9	70.9	69.5
NN-Former	59.3	58.8	62.4	59.3
DTA-GT	**74.1**	**73.8**	**74.8**	**74.1**

DTA-GT achieves 74.1% accuracy and 73.8% macro-F1, leading the compared methods on all four metrics. Its accuracy exceeds Graphormer by 4.6 percentage points and BRP-NAS by 17.1 percentage points. With a shared node token, these results support the effectiveness of the direction- and topology-aware representation on substantially larger DAGs.

F.2 Computational Efficiency on Large DAGs

We profile representative MalNet-derived DAGs and generated DAGs on an NVIDIA A800-SXM4 GPU with 80 GB memory. The real-graph profiles range from 16 nodes and 25 edges to 3,416 nodes and 9,215 edges, while the generated profiles range from 8 nodes and 16 edges to 32,768 nodes and 65,536 edges. The real graphs have median edge density below 0.3%. All methods use identical graphs and reported model settings with batch size one. The real-DAG comparison reports the graph preprocessing required by each method followed by one forward pass through the model. For generated DAGs, we report each method's measured prediction latency. The DTA-GT values include only the model forward pass and exclude graph-input construction. For the largest generated DAG, we also report a separate breakdown of the first-use cost, including graph-input construction. The real-DAG DTA-GT stage profile uses ten warm-up and thirty timed measurements for construction of graph inputs, followed by thirty warm-up and five hundred timed measurements for model forward. The independently instrumented profile on the largest generated DAG uses one warm-up and five timed measurements for each construction stage, followed by thirty warm-up and fifty timed measurements for model forward.

The full-size DTA-GT predictor uses a hidden dimension of $d = 180$ in the main NAS prediction experiments. For the large-DAG latency and memory measurements in this section, we use $d = 192$ with ten encoder layers, six attention heads, four retained spectral modes, and a nominal phase parameter of $q = 0.25$. Graph-dependent structural masks and spectral features are constructed once per graph and shared by all encoder layers. For larger graphs, we represent the magnetic Laplacian in sparse CSR form and compute only the retained eigenpairs using thick-restart Lanczos [44]; smaller graphs use dense eigendecomposition. We report construction of structural masks, magnetic spectral construction, and model forward separately in Table 16.

Table 16: DTA-GT stage latencies on representative MalNet-derived DAGs. Graph inputs are constructed before model forward. Columns identify node and edge counts, and entries report mean latency in milliseconds.

Component	16–25	88–180	362–859	1,427–3,844	3,416–9,215
Construction of structural masks	0.39	0.39	0.39	1.25	11.46
Magnetic spectral construction	0.64	2.89	6.15	7.33	7.48
Model forward	5.19	5.30	5.52	12.13	40.01

On the 3,416-node DAG, construction of structural masks, spectral construction, and model forward take 11.46 ms, 7.48 ms, and 40.01 ms, respectively. The sum of these reported stages is 58.95 ms. For repeated predictions of the same architecture, the constructed graph inputs can be reused and only the model-forward stage is executed again.

Table 17 compares end-to-end method-specific preprocessing and inference latency on representative MalNet-derived DAGs. On the largest profiled real DAG, DTA-GT requires 58.95 ms, compared with 122.56 ms for PINAT and 161.20 ms for NN-Former. BRP-NAS requires 4.24 ms on the same graph, showing the lower computational cost of the GCN predictor in this profile.

Table 17: End-to-end latency on representative MalNet-derived DAGs. Each method includes its required graph preprocessing and one inference pass. Values are mean milliseconds per graph on the A800, and columns identify node and edge counts.

Method	16–25	88–180	362–859	1,427–3,844	3,416–9,215
BRP-NAS	**0.55**	**0.55**	**0.55**	**1.35**	**4.24**
PINAT	10.74	10.65	10.52	29.97	122.56
NAR-Former	43.50	238.11	1,339.62	11,612.44	55,393.56
NAR-Former V2	8.58	30.27	119.67	449.76	1,185.62
NN-Former	7.29	7.27	7.44	22.71	161.20
DTA-GT	6.22	8.58	12.06	<u>20.71</u>	<u>58.95</u>

Table 18 extends the profile to generated DAGs with up to 32,768 nodes and 65,536 edges. DTA-GT completes model forward after construction of graph inputs on the largest generated DAG in 3,125.12 ms and reaches 42.249 GiB of peak GPU memory. PINAT and NAR-Former V2 record 9,684.47 ms and 12,036.15 ms, respectively, on their profiled prediction paths. NAR-Former and NN-Former encounter out-of-memory failures at this graph size under their reported settings, while BRP-NAS records 313.80 ms.

Table 18: Recorded prediction-path latency on generated DAGs. The DTA-GT row reports model forward after construction of graph inputs; Table 19 reports its complete first-use stage sum on the largest graph. Values are mean milliseconds per graph on the A800. OOM denotes an out-of-memory failure under the reported settings.

Method	8–16	64–128	512–1,024	4,096–8,192	32,768–65,536
BRP-NAS	0.56	0.54	0.58	6.08	313.80
PINAT	10.66	10.55	10.58	191.52	9,684.47
NAR-Former	28.23	166.74	2,248.00	78,731.93	OOM
NAR-Former V2	6.01	23.96	161.23	1,298.03	12,036.15
NN-Former	7.43	7.24	7.76	268.80	OOM
DTA-GT (forward)	5.20	5.21	5.41	61.08	3,125.12

Table 19 independently profiles four DTA-GT stages on the largest generated DAG. Model forward after construction of graph inputs takes 3,125.12 ms and reaches 42.249 GiB of peak GPU memory. The independently measured means for feature preprocessing, spectral construction, construction of structural masks, and model forward sum to 25,601.083 ms for first use. The stage breakdown shows that graph-dependent preprocessing is a substantial part of the first-use cost at this scale.

Table 19: Staged DTA-GT profile for the generated DAG with 32,768 nodes and 65,536 edges. Stage means are measured independently, and the first-use value is their arithmetic sum. Peak GPU memory refers to model forward.

Stage or resource	Recorded value
Feature preprocessing	14,616.178 ms
Magnetic spectral construction	9.923 ms
Construction of structural masks	7,849.865 ms
Model forward	3,125.117 ms
First-use stage sum	25,601.083 ms
Peak forward GPU memory	42.249 GiB

For generated DAGs containing a spanning directed chain, we additionally evaluate a topology-specific relation-construction procedure motivated by the support-aware design discussed in Appendix C.3. On the 32,768-node graph, this procedure constructs the six masks in 33.340 ms, and the resulting dense Boolean masks occupy 6 GiB. This result separates time required to construct relations from relation storage and provides measured evidence for topology-aware relation handling.

The MalNet-derived DAG experiments show that DTA-GT achieves the highest accuracy and macro-F1 among the compared predictors, supporting the effectiveness of the complete encoder on the evaluated larger graphs. These results assess the combined directional design at the tested graph scales and do not isolate the predictive contribution of magnetic spectral encoding. As graph size increases, graph scaling reduces the orientation phase per edge, which may weaken the spectral directional signal in some large-graph regimes. BRP-NAS has lower latency in the large-graph profiles. The staged A800 measurements support the intended roles of retained-mode spectral computation, reusable graph-dependent preprocessing, and topology-aware relation construction within the evaluated range. The reported scaling evidence covers graphs with up to 32,768 nodes; sparse or hierarchical attention and compact reachability representations remain important directions for substantially larger computational graphs.

G Extended Discussion

Cross-space representation and adaptation. A node in a cell-based CNN space may denote a primitive convolution or pooling operation, whereas a node in another space may denote an entire Transformer layer or Mamba block. These different computational granularities motivate a canonical operation-level representation for cross-space adaptation. Composite blocks could be decomposed into shared semantic units or mapped to a common operation vocabulary with aligned edge definitions and target protocols. For spaces with compatible operation definitions and dependency semantics, future work can investigate reusing DTA-GT's directional encoder and graph preprocessing, with new vocabulary entries and embeddings for additional operation types. This would enable studies of cross-space pretraining followed by adaptation on a small evaluated subset from the target space.

Task adaptation and NAS evaluation. For a new DAG-level task with compatible atomic units and semantics of directed dependencies, adapting the prediction head and fine-tuning the encoder on a small labeled subset provides a natural extension. This would allow the same direction-aware representation principle to be studied under different graph-level targets. A complementary direction is to integrate the predictor into a NAS search loop and evaluate how candidate ranking affects architecture selection under a fixed end-to-end evaluation budget.

Relation-channel design and extension. The three paired relation families in Section 4.3 are heuristic choices motivated by DAG semantics, providing a concrete instantiation of direction-dual structural attention. They capture local dependencies, multi-hop reachability, and shared predecessor/successor structure. Additional directional information, such as hop-specific dependencies or directed motif patterns, can be represented by further ordered or role-specific relation supports, with a corresponding extension of the relation-to-head assignment. The relation set can therefore be tailored to the directional semantics available in a target graph domain. Our experiments evaluate

the six-channel instantiation; the predictive value and computational cost of expanded relation sets remain to be studied.

Computational scaling. Directed reachability can grow quadratically even when the original adjacency is sparse. For example, a directed chain has $n-1$ edges but $n(n-1)/2$ reachable ordered pairs. Dense relation masks therefore require quadratic storage, and dense masked attention incurs quadratic pairwise computation for a fixed hidden dimension. The A800 profiles in Appendix F.2 demonstrate execution on generated DAGs with up to 32,768 nodes. Extending graph Transformer predictors to substantially larger computational graphs calls for sparse or hierarchical attention together with compact representations of directed reachability. Combining these mechanisms with reusable spectral preprocessing could extend the encoder to larger computational graphs while retaining its direction-aware representation. Future evaluations should measure prediction quality, construction of graph inputs on first use, latency of the model's forward pass, and GPU memory under matched graph and hardware settings.

H Complete Results with Standard Deviations

This section reports five-run means and standard deviations for the main comparisons, ablations, and spectral controls in Tables 2–4.

H.1 NAS-Bench-101 and NAS-Bench-201 Prediction

The complete results for all four prediction tasks are reported as means and standard deviations in Tables 20–23.

Table 20: Complete accuracy-prediction results on NAS-Bench-101. Each cell places the mean above its standard deviation.

Method	KT ↑ by training budget				MAPE ↓ by training budget			
	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)
ReNAS	0.0963 ± 0.0579	0.1240 ± 0.0453	0.1596 ± 0.0471	0.5141 ± 0.1137	2.8409 ± 2.7629	0.8343 ± 0.1109	0.5456 ± 0.1241	0.0660 ± 0.0338
NAO	0.3395 ± 0.0445	0.4015 ± 0.0218	0.4202 ± 0.0123	0.4472 ± 0.0017	0.0577 ± 0.0149	0.0559 ± 0.0030	0.0493 ± 0.0108	0.0407 ± 0.0017
NNLP	0.4452 ± 0.0188	0.4936 ± 0.0319	0.4987 ± 0.0470	0.7809 ± 0.0046	0.0665 ± 0.0192	0.0412 ± 0.0023	0.0399 ± 0.0118	0.0252 ± 0.0007
BRP-NAS	0.4129 ± 0.0116	0.4338 ± 0.0107	0.4549 ± 0.0093	0.7643 ± 0.0007	0.0672 ± 0.0079	0.0548 ± 0.0086	0.0424 ± 0.0051	0.0236 ± 0.0002
Graphormer	0.2229 ± 0.0687	0.2610 ± 0.0314	0.2991 ± 0.0202	0.5610 ± 0.0013	0.0589 ± 0.0081	0.0536 ± 0.0124	0.0477 ± 0.0054	0.0355 ± 0.0003
TNASP	0.5091 ± 0.0079	0.5749 ± 0.0125	0.6404 ± 0.0109	0.7663 ± 0.0035	0.0443 ± 0.0020	0.0393 ± 0.0010	0.0377 ± 0.0009	0.0250 ± 0.0003
NAR-Former	0.4228 ± 0.0048	0.4986 ± 0.0120	0.5699 ± 0.0097	0.8181 ± 0.0012	0.0538 ± 0.0055	0.0423 ± 0.0011	0.0406 ± 0.0017	0.0257 ± 0.0005
PINAT	0.3496 ± 0.0553	0.3994 ± 0.0050	0.4243 ± 0.0078	0.7162 ± 0.0153	0.1339 ± 0.0558	0.0516 ± 0.0101	0.0406 ± 0.0025	0.0271 ± 0.0012
GraphTrans	0.4300 ± 0.0229	0.4438 ± 0.0419	0.4666 ± 0.0067	0.7702 ± 0.0035	0.2735 ± 0.1625	0.1401 ± 0.0688	0.1337 ± 0.0583	0.0252 ± 0.0004
NAR-Former V2	0.5300 ± 0.0100	0.5500 ± 0.0040	0.6000 ± 0.0040	0.8150 ± 0.0025	0.0900 ± 0.0450	0.0850 ± 0.0300	0.0399 ± 0.0090	0.0220 ± 0.0007
NN-Former	0.5428 ± 0.0107	0.5642 ± 0.0034	0.6057 ± 0.0037	0.8209 ± 0.0024	0.0895 ± 0.0472	0.0907 ± 0.0314	0.0368 ± 0.0016	0.0219 ± 0.0007
DTA-GT	**0.6954 ± 0.0105**	**0.7278 ± 0.0047**	**0.7980 ± 0.0038**	**0.8740 ± 0.0001**	**0.0327 ± 0.0001**	**0.0312 ± 0.0006**	**0.0283 ± 0.0004**	**0.0202 ± 0.0017**

Table 21: Complete latency-prediction results on NAS-Bench-101. Each cell places the mean above its standard deviation.

Method	KT ↑ by training budget				MAPE ↓ by training budget			
	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)	0.02% (@100)	0.04% (@172)	0.1% (@424)	1% (@4236)
ReNAS	0.5803 ± 0.0108	0.7127 ± 0.0278	0.8028 ± 0.0048	0.9222 ± 0.0024	0.2232 ± 0.0051	0.1588 ± 0.0130	0.1080 ± 0.0026	0.0405 ± 0.0009
NAO	0.2569 ± 0.0372	0.2583 ± 0.0509	0.3049 ± 0.0752	0.3964 ± 0.0004	0.3057 ± 0.0016	0.3068 ± 0.0009	0.3062 ± 0.0007	0.2878 ± 0.0009
NNLP	0.8056 ± 0.0180	0.8379 ± 0.0034	0.8685 ± 0.0014	0.9190 ± 0.0024	0.1064 ± 0.0103	0.0858 ± 0.0019	0.0695 ± 0.0009	0.0417 ± 0.0013
BRP-NAS	0.7937 ± 0.0030	0.8487 ± 0.0027	0.8702 ± 0.0015	0.9270 ± 0.0006	0.1105 ± 0.0008	0.0811 ± 0.0021	0.0682 ± 0.0004	0.0372 ± 0.0003
Graphormer	0.3582 ± 0.0079	0.3711 ± 0.0094	0.4084 ± 0.0176	0.4467 ± 0.0007	0.2862 ± 0.0076	0.2715 ± 0.0015	0.2630 ± 0.0025	0.2617 ± 0.0013
TNASP	0.8024 ± 0.0082	0.8325 ± 0.0056	0.8734 ± 0.0034	0.9394 ± 0.0009	0.1110 ± 0.0046	0.0897 ± 0.0033	0.0647 ± 0.0015	0.0311 ± 0.0005
NAR-Former	0.8337 ± 0.0023	0.8670 ± 0.0031	0.8836 ± 0.0031	0.9434 ± 0.0007	0.0881 ± 0.0011	0.0686 ± 0.0011	0.0606 ± 0.0015	0.0287 ± 0.0003
PINAT	0.8317 ± 0.0119	0.8621 ± 0.0006	0.8817 ± 0.0018	0.9389 ± 0.0005	0.0860 ± 0.0050	0.0704 ± 0.0004	0.0599 ± 0.0011	0.0315 ± 0.0003
GraphTrans	0.8182 ± 0.0148	0.8619 ± 0.0026	0.8862 ± 0.0011	0.9365 ± 0.0007	0.0949 ± 0.0085	0.0710 ± 0.0008	0.0592 ± 0.0005	0.0329 ± 0.0004
NAR-Former V2	0.7741 ± 0.0115	0.8314 ± 0.0069	0.8752 ± 0.0039	0.9380 ± 0.0019	0.1238 ± 0.0069	0.0888 ± 0.0038	0.0652 ± 0.0024	0.0317 ± 0.0007
NN-Former	0.8385 ± 0.0011	0.8694 ± 0.0022	0.9082 ± 0.0006	0.9462 ± 0.0006	0.0907 ± 0.0005	0.0662 ± 0.0011	0.0513 ± 0.0004	0.0286 ± 0.0003
DTA-GT	**0.8646 ± 0.0036**	**0.8930 ± 0.0012**	**0.9094 ± 0.0010**	**0.9490 ± 0.0007**	**0.0758 ± 0.0009**	**0.0613 ± 0.0021**	**0.0503 ± 0.0022**	**0.0271 ± 0.0010**

Table 22: Complete accuracy-prediction results on NAS-Bench-201. Each cell places the mean above its standard deviation.

Method	KT ↑ by training budget				MAPE ↓ by training budget			
	1% (@156)	3% (@469)	5% (@781)	10% (@1563)	1% (@156)	3% (@469)	5% (@781)	10% (@1563)
ReNAS	0.1816 ± 0.1025	0.2236 ± 0.1178	0.4097 ± 0.0893	0.7130 ± 0.0338	1.9822 ± 1.7129	0.8425 ± 0.5541	0.3444 ± 0.0594	0.1676 ± 0.0185
NAO	0.6254 ± 0.0108	0.6436 ± 0.0170	0.6445 ± 0.0071	0.6580 ± 0.0100	0.0478 ± 0.0026	0.0454 ± 0.0048	0.0399 ± 0.0057	0.0283 ± 0.0016
NNLP	0.7109 ± 0.0230	0.7380 ± 0.0151	0.7824 ± 0.0332	0.8196 ± 0.0299	0.0410 ± 0.0272	0.0245 ± 0.0075	0.0172 ± 0.0070	0.0115 ± 0.0018
BRP-NAS	0.6953 ± 0.0112	0.7177 ± 0.0120	0.7575 ± 0.0113	0.7768 ± 0.0089	0.0629 ± 0.0249	0.0506 ± 0.0115	0.0211 ± 0.0025	0.0201 ± 0.0034
Graphormer	0.7052 ± 0.0039	0.7311 ± 0.0062	0.7461 ± 0.0128	0.7650 ± 0.0024	0.0270 ± 0.0044	0.0185 ± 0.0002	0.0171 ± 0.0006	0.0149 ± 0.0010
TNASP	0.7405 ± 0.0080	0.7770 ± 0.0073	0.8061 ± 0.0016	0.8314 ± 0.0013	0.0302 ± 0.0007	0.0158 ± 0.0019	0.0145 ± 0.0044	0.0110 ± 0.0006
NAR-Former	0.7540 ± 0.0107	0.7839 ± 0.0039	0.8188 ± 0.0036	0.8554 ± 0.0009	0.0256 ± 0.0017	0.0188 ± 0.0005	0.0141 ± 0.0006	0.0107 ± 0.0005
PINAT	0.7027 ± 0.0167	0.7084 ± 0.0006	0.7589 ± 0.0021	0.7920 ± 0.0034	0.0338 ± 0.0055	0.0273 ± 0.0048	0.0269 ± 0.0123	0.0198 ± 0.0110
GraphTrans	0.6957 ± 0.0056	0.6959 ± 0.0055	0.7781 ± 0.0023	0.8258 ± 0.0025	0.0975 ± 0.0510	0.0613 ± 0.0086	0.0493 ± 0.0224	0.0153 ± 0.0031
NAR-Former V2	0.7418 ± 0.0049	0.7432 ± 0.0086	0.8004 ± 0.0052	0.8391 ± 0.0055	0.1363 ± 0.0511	0.0441 ± 0.0102	0.0340 ± 0.0074	0.0256 ± 0.0062
NN-Former	0.7400 ± 0.0050	0.7420 ± 0.0060	0.8050 ± 0.0040	0.8400 ± 0.0040	0.1350 ± 0.0500	0.0450 ± 0.0100	0.0330 ± 0.0070	0.0260 ± 0.0060
DTA-GT	**0.8015 ± 0.0133**	**0.8925 ± 0.0028**	**0.9138 ± 0.0010**	**0.9304 ± 0.0011**	**0.0244 ± 0.0105**	**0.0125 ± 0.0007**	**0.0096 ± 0.0012**	**0.0084 ± 0.0022**

Table 23: Complete latency-prediction results on NAS-Bench-201. Each cell places the mean above its standard deviation.

Method	KT ↑ by training budget				MAPE ↓ by training budget			
	1% (@156)	3% (@469)	5% (@781)	10% (@1563)	1% (@156)	3% (@469)	5% (@781)	10% (@1563)
ReNAS	0.6166 ± 0.0032	0.6506 ± 0.0070	0.6725 ± 0.0005	0.6863 ± 0.0004	0.0844 ± 0.0009	0.0777 ± 0.0009	0.0707 ± 0.0004	0.0640 ± 0.0006
NAO	0.6963 ± 0.0004	0.6980 ± 0.0003	0.6984 ± 0.0002	0.7012 ± 0.0002	0.0628 ± 0.0000	0.0602 ± 0.0000	0.0600 ± 0.0001	0.0592 ± 0.0000
NNLP	0.6349 ± 0.0062	0.6437 ± 0.0016	0.6527 ± 0.0017	0.6567 ± 0.0019	0.0807 ± 0.0009	0.0726 ± 0.0007	0.0716 ± 0.0003	0.0702 ± 0.0002
BRP-NAS	0.6690 ± 0.0021	0.6861 ± 0.0005	0.6923 ± 0.0011	0.6986 ± 0.0005	0.0717 ± 0.0003	0.0640 ± 0.0001	0.0615 ± 0.0002	0.0594 ± 0.0001
Graphormer	0.6799 ± 0.0010	0.6972 ± 0.0010	0.7030 ± 0.0017	0.7030 ± 0.0006	0.0651 ± 0.0005	0.0600 ± 0.0001	0.0595 ± 0.0002	0.0589 ± 0.0001
TNASP	0.6646 ± 0.0069	0.6873 ± 0.0010	0.6926 ± 0.0008	0.6990 ± 0.0005	0.0713 ± 0.0010	0.0633 ± 0.0001	0.0613 ± 0.0001	0.0595 ± 0.0002
NAR-Former	0.5916 ± 0.0637	0.6014 ± 0.0393	0.6275 ± 0.0101	0.6522 ± 0.0027	0.1309 ± 0.0070	0.1299 ± 0.0096	0.1229 ± 0.0001	0.1226 ± 0.0001
PINAT	0.6859 ± 0.0107	0.6992 ± 0.0008	0.6950 ± 0.0019	0.6996 ± 0.0001	0.0645 ± 0.0010	0.0607 ± 0.0002	0.0602 ± 0.0003	0.0593 ± 0.0001
GraphTrans	0.6441 ± 0.0059	0.6871 ± 0.0013	0.6964 ± 0.0006	0.6993 ± 0.0008	0.0762 ± 0.0013	0.0643 ± 0.0006	0.0612 ± 0.0001	0.0600 ± 0.0003
NAR-Former V2	0.6237 ± 0.0201	0.6474 ± 0.0060	0.6541 ± 0.0032	0.6781 ± 0.0030	0.0786 ± 0.0060	0.0719 ± 0.0009	0.0699 ± 0.0002	0.0680 ± 0.0009
NN-Former	0.6819 ± 0.0014	0.6891 ± 0.0012	0.6947 ± 0.0014	0.6981 ± 0.0017	0.0695 ± 0.0003	0.0631 ± 0.0003	0.0609 ± 0.0001	0.0594 ± 0.0000
DTA-GT	**0.8375 ± 0.0048**	**0.8409 ± 0.0008**	**0.8435 ± 0.0025**	**0.8492 ± 0.0003**	**0.0305 ± 0.0007**	**0.0293 ± 0.0004**	**0.0280 ± 0.0010**	**0.0276 ± 0.0001**

H.2 Ablation Analysis

Table 24 reports the uncertainty estimates corresponding to the mean-only ablation results in Table 3.

Table 24: Complete ablation results under the lowest training budget of each task. Values are mean ± standard deviation.

Variant	NAS-Bench-101				NAS-Bench-201			
	Accuracy		Latency		Accuracy		Latency	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
Directed DTA-GT	**0.6954 ± 0.0105**	**0.0327 ± 0.0001**	**0.8646 ± 0.0036**	**0.0758 ± 0.0009**	**0.8015 ± 0.0133**	**0.0244 ± 0.0105**	**0.8375 ± 0.0048**	**0.0305 ± 0.0007**
Undirected	0.5882 ± 0.0063	0.0408 ± 0.0016	0.8062 ± 0.0085	0.1074 ± 0.0014	0.7215 ± 0.0110	0.0617 ± 0.0253	0.8178 ± 0.0103	0.0349 ± 0.0031
w/o Directed Node Features	0.6522 ± 0.0236	0.0361 ± 0.0019	0.8482 ± 0.0056	0.0908 ± 0.0058	0.7198 ± 0.0205	0.0729 ± 0.0140	0.8202 ± 0.0050	0.0326 ± 0.0016
w/o Spec. Enc.	0.6894 ± 0.0016	0.0341 ± 0.0015	0.8615 ± 0.0008	0.0806 ± 0.0031	0.7452 ± 0.0089	0.0600 ± 0.0055	0.8192 ± 0.0003	0.0325 ± 0.0009
w/o Struct. Attn.	0.6552 ± 0.0144	0.0331 ± 0.0007	0.8360 ± 0.0202	0.1033 ± 0.0017	0.6123 ± 0.0339	0.1037 ± 0.0087	0.8227 ± 0.0033	0.0317 ± 0.0029
w/o MoE-FFN	0.6542 ± 0.0095	0.0331 ± 0.0007	0.8495 ± 0.0020	0.0860 ± 0.0022	0.7248 ± 0.0137	0.0418 ± 0.0062	0.8298 ± 0.0063	0.0313 ± 0.0024
CLS Readout	0.6701 ± 0.0101	0.0399 ± 0.0048	0.8420 ± 0.0047	0.1203 ± 0.0096	0.7535 ± 0.0141	0.0704 ± 0.0103	0.8209 ± 0.0030	0.0348 ± 0.0012
MSE Only	0.6720 ± 0.0109	0.0356 ± 0.0022	0.8610 ± 0.0056	0.0808 ± 0.0060	0.6937 ± 0.0065	0.0686 ± 0.0006	0.8201 ± 0.0021	0.0321 ± 0.0007

H.3 Magnetic Spectral Controls

Table 25 provides the available repeated-run statistics corresponding to the mean-only controls in Table 4. The "Directed with Symmetric Lap." row reports the $q = 0$ results from the five-run phase-parameter sensitivity experiment.

Table 25: Complete symmetric- and magnetic-Laplacian control results under the lowest training budget of each task. Cells report mean ± standard deviation.

Variant	NAS-Bench-101 @100				NAS-Bench-201 @156			
	Accuracy		Latency		Accuracy		Latency	
	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓	KT ↑	MAPE ↓
No Directional Encoding	0.5882 ± 0.0063	0.0408 ± 0.0016	0.8062 ± 0.0085	0.1074 ± 0.0014	0.7215 ± 0.0110	0.0617 ± 0.0253	0.8178 ± 0.0103	0.0349 ± 0.0031
No Direction + Symmetric Lap.	0.5878 ± 0.0090	0.0386 ± 0.0018	0.8071 ± 0.0169	0.1066 ± 0.0082	0.7135 ± 0.0472	0.0485 ± 0.0123	0.8206 ± 0.0192	0.0330 ± 0.0055
Directed with Symmetric Lap.	0.6894 ± 0.0161	0.0341 ± 0.0008	0.8615 ± 0.0051	0.0806 ± 0.0014	0.7148 ± 0.0159	0.0792 ± 0.0101	0.8192 ± 0.0096	0.0325 ± 0.0042
Directed with Magnetic Lap.	**0.6954 ± 0.0105**	**0.0327 ± 0.0001**	**0.8646 ± 0.0036**	**0.0758 ± 0.0009**	**0.8015 ± 0.0133**	**0.0244 ± 0.0105**	**0.8375 ± 0.0048**	**0.0305 ± 0.0007**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims are scoped to the architecture-prediction settings evaluated in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 6 and Appendix G.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not state formal theorems or lemmas.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Datasets, per-seed shared splits, training settings, and reproduction details are provided in Appendices A–E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The public code and data repository is linked in Appendix A.5.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and evaluation settings are provided in Section 5 and Appendices A and E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix H reports five-run means and standard deviations for Tables 2–4, using the repeated-run protocol in Appendix A.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Hardware and selected resource profiles are reported in Appendices A, D, and F, but complete per-run and total compute accounting is not provided.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The experiments use public benchmarks and involve no research with human subjects or personal data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This is foundational graph-representation research on architecture benchmarks and MalNet-derived DAGs, with no direct societal deployment.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work does not release assets with a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Existing assets are credited and used under applicable licenses or permissions; Table 6 reports code sources and license or availability information.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Code and latency-label assets are linked in Appendix A.5, with reproduction details in Appendices A–E.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work involves no crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: No research with human subjects is conducted.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not an important, original, or non-standard component of the core method.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.