Transformers Meet Directed Graphs

Simon Geisler[1, 2], **Cosmin Paduraru**[1], **Yujia Li**[1], **Ali Taylan Cemgil**[1]
[1]DeepMind, [2]Technical University of Munich
s.geisler@tum.de, {paduraru, yujiali, taylancemgil}@google.com

Abstract

Transformers were originally proposed as a sequence to sequence model for text but have become vital for a wide range of modalities, including images, audio, video, and *undirected* graphs. However, transformers for *directed* graphs are a surprisingly underexplored topic, despite their applicability to fundamental domains including source code and logic circuits. While spectral encodings based on the combinatorial Laplacian relate to the sinusoidal positional encodings for ordered sequences, they cannot capture directedness in graphs. To overcome this limitation, we propose to use the Magnetic Laplacian – a direction-aware generalization of the combinatorial Laplacian. Empirically, we show that the extra directionality information is useful in various downstream tasks, including correctness testing of sorting networks and source code understanding. Together with a data-flow-centric graph construction, our model outperforms the prior state of the art on the Open Graph Benchmark Code2 relatively by 14.7%[1].

1 Introduction

Transformers have become the central component in many state-of-the-art approaches spanning a wide range of modalities. For example, transformers are used to generate solutions for competitive programming tasks from textual descriptions (Li et al. 2022) or find approximate solutions to combinatorial optimizations problems like the Traveling Salesman Problem (Kool, Hoof, and Welling 2019). Transformers also have had first success on graph learning tasks (Min et al. 2022). The biggest challenge with graphs as input modality is arguably how the attention mechanism can become structure-aware. Prevalent strategies include (1) *modifying the attention* mechanism itself to incorporate structural information (Ying et al. 2021), (2) *hybrid architectures* that also contain Graph Neural Networks (GNNs) (Mialon et al. 2021; Chen, O'Bray, and Borgwardt 2022), and (3) approaches relying on *positional encodings* for graphs (Dwivedi and Bresson 2021).

We find that transformers for directed graphs have received surprisingly little attention so far, despite their applicability to important tasks, including graph representations for source code. Specifically, most of the literature for structure-aware positional encodings (3) either uses basic measures like pairwise shortest path distances (Guo et al. 2020) or symmetrizes the graph for theoretically motivated positional encodings, e.g. based on graph spectral theory (Dwivedi and Bresson 2021). Importantly, symmetrizing the graph neglects any information about the edges' direction and therefore comes with severe limitations. For this reason, we propose to use the eigenvectors of the *Magnetic Laplacian* (§ 3), a natural direction-aware generalization of the well-known combinatorial Laplacian for undirected graphs (see Fig. 1). We argue that such a positional encoding is an important step towards general-purpose transformers that universally handle both undirected and directed graphs (see Fig. 4 for architecture).

We show that directional information can be beneficial for downstream tasks. We study two semantically rich high-level reasoning tasks. Namely, *function name prediction* on the Open Graph Benchmark (OGB) Code2 dataset (Hu et al. 2020) and *correctness prediction* of sorting networks (see § 4). Sorting networks are an old problem in theoretical computer science (Knuth 1973) and here the goal is to predict whether a given network is correct, i.e., sorts all sequences of a fixed length. Studying sorting networks also reveals important intuitions about the importance of directed graphs: (a) symmetrization maps both correct and incorrect sorting networks to the same input graph. (b) a sequence of program instructions can be understood as a single topological sort of the underlying directed graph. This allows quantifying how many equivalent programs map to the same graph. In other words, directed graphs can drastically reduce the effective input dimensionality. The topological sort analogy is an important insight since treating source code as a sequence is still the de facto standard. These insights motivate us to rethink the graph construction for source code (see § 5).

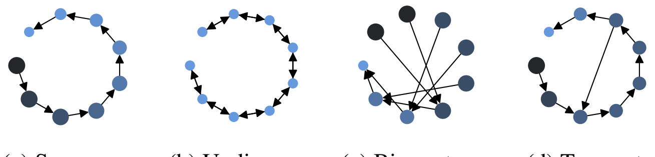

(a) Sequence (b) Undir. seq. (c) Binary tree (d) Trumpet

Figure 1: First eigenvector γ_0 of Magnetic Laplacian (b also comb. Lap.). Node size shows real and color imaginary value.

[1] Code: github.com/deepmind/digraph_transformer

Contributions. (I) We propose spectral *positional encodings* that also generalize to directed graphs. (II) We introduce the task of predicting the correctness of *sorting networks*, a canonical ambiguity-free application where directionality is of the essence. (III) We set a new *state on the art* for the OGB Code2 dataset (2.85% higher F1 score, 14.7% relatively).

2 Sinusoidal and Laplacian Encodings

It is commonly argued that the eigenvectors of the (combinatorial) Laplacian generalize the sinusoidal positional encodings, due to their relationship via the Graph Fourier Transformation (GFT) and Discrete Fourier Transformation (DFT). Even though sinusoidal positional encodings capture the direction, eigenvectors of the Laplacian do not. We next discuss their relationship and differences for a sequence (Fig. 1a,b).

Sinusoidal encodings $\mathrm{PE}_{v,2k} \coloneqq \cos({}^{v}\!/_{10{,}000^{2k/d_{\text{model}}}})$ and $\mathrm{PE}_{v,2k+1} \coloneqq \sin({}^{v}\!/_{10{,}000^{2k/d_{\text{model}}}})$ encode the token position $0 \le v < n$ to the dimensions of the model d_{model} with $0 \le k < {}^{d_{\text{model}}}\!/_{2}$. The DFT also yields pos. encodings PE':

$$\mathrm{X}_k \coloneqq \sum_{v=0}^{n-1} x_v \Big[\underbrace{\cos\Big(\frac{2\pi}{n}kv\Big)}_{\mathrm{PE}'_{v,2k}} - i \cdot \underbrace{\sin\Big(\frac{2\pi}{n}kv\Big)}_{\mathrm{PE}'_{v,2k+1}} \Big] \tag{1}$$

By design (Vaswani et al. 2017), sinusoidal encodings (a) sweep the frequencies using a geometric series instead of linearly; (b) also contain frequencies below ${}^{1}\!/_{n}$; and (c) have d_{model} components instead of $2n$ (i.e. $0 \le k < n$ in Eq. 1).

Eigenvectors of Laplacian. Similarly, we can define a "Graph Fourier Transformation" via the eigendecomposition of the combinatorial Laplacian $\boldsymbol{L} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^{-1}$, with diagonal matrix $\boldsymbol{\Lambda}$ of eigenvalues and orthogonal matrix $\boldsymbol{\Gamma}$ of eigenvectors. $\boldsymbol{\Gamma}$ can be used as pos. encodings. The real, symmetric, and positive semi-definite unnormalized Laplacian $\boldsymbol{L}_U$ as well as degree-normalized Laplacian $\boldsymbol{L}_N$ are defined as:

$$\boldsymbol{L}_U \coloneqq \boldsymbol{D}_s - \boldsymbol{A}_s \quad (2) \qquad \boldsymbol{L}_N \coloneqq \mathbf{I} - (\boldsymbol{D}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{D}_S^{-1/2}) \quad (3)$$

are defined based on the diagonal degree matrix $\boldsymbol{D}_s$ of the symmetrized adjacency matrix $\boldsymbol{A}_s = \boldsymbol{A} \vee \boldsymbol{A}^\top$ with $\boldsymbol{A} \in \{0,1\}^{n \times n}$. Symmetrization is required s.t. $\boldsymbol{L}$ is guaranteed to be diagonalizable and that $\boldsymbol{\Gamma}$ form an orthogonal basis. We order eigenvalues and eigenvectors s.t. $0 \le \boldsymbol{\lambda}_0 \le \boldsymbol{\lambda}_1 \le \dots \le \boldsymbol{\lambda}_{n-1}$. We call $\boldsymbol{\lambda}_0$ or $\boldsymbol{\Lambda}_{0,0}$ the first eigenvalue and $\boldsymbol{\gamma}_0$ or $\boldsymbol{\Gamma}_{:,0}$ the first eigenvector.

Two notable differences to the DFT are (1) the real-valued eigenvectors of the Laplacian and (2) the *sign invariance*. Specifically, if we consider a sequence (Eq. 6), the eigenvectors are given by the Cosine Transformation Type II: $\boldsymbol{\Gamma}_{j,k} = \pm\cos((j + {}^{1}\!/_{2})k{}^{\pi}\!/_{n})$, where we must choose the same sign per k or, equivalently, an eigenvector. The Cosine Transformation typically fixes $\pm$ to $+$. Thus, the encodings of the first token/node are non-negative. However, for general graphs it is not that simple to resolve this ambiguity (e.g. multiple sink and source nodes). Thus, we typically use an arbitrary sign for each $\boldsymbol{\gamma}$ (Dwivedi and Bresson 2021).

Due to this *sign invariance*, the eigenvectors $\boldsymbol{\gamma}$ of the combinatorial Laplacian cannot distinguish the direction. Alternatively, from symmetrization, we see that distinguishing direction is not possible. That is, we obtain the same symmetrized graph regardless of how we reverse edges.

3 Directional Spectral Positional Encodings

For the ability to distinguish direction, we propose to use the Magnetic Laplacian, a generalization of the combinatorial Laplacian. The Magnetic Laplacian is a Hermitian matrix that encodes the direction of edges with complex numbers. This way we can have the best of both worlds: (1) the eigenvectors form an orthogonal basis and (2) we capture the directionality.

The Magnetic Laplacian is defined as

$$\boldsymbol{L}_U^{(q)} \coloneqq \boldsymbol{D}_S - \boldsymbol{A}_S \odot \exp\Big(i\boldsymbol{\Theta}^{(q)}\Big) \tag{4}$$

with Hadamard product $\odot$, element-wise $\exp$, potential q and $i = \sqrt{-1}$. Recall, $\boldsymbol{D}_S$ is the symmetrized degree matrix and $\boldsymbol{A}_S$ the symmetrized adjacency matrix. Except for the $\exp$ of

$$\boldsymbol{\Theta}_{u,v}^{(q)} \coloneqq 2\pi q(\boldsymbol{A}_{u,v} - \boldsymbol{A}_{v,u}), \qquad q \ge 0 \tag{5}$$

Eq. 4 is equivalent to the Laplacian $\boldsymbol{L}_U \coloneqq \boldsymbol{D}_S$ - $\boldsymbol{A}_S$. The Magnetic Laplacian for a sequence with $q = 0$ and $q = {}^{1}\!/_{4}$ are given next as well as their first eigenvec. in Fig. 1b and a.

$$\boldsymbol{L}_U^{(0)} = \begin{bmatrix} 1 & -1 & \cdots & 0 & 0 \\ -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & \cdots & -1 & 1 \end{bmatrix} \quad (6) \qquad \boldsymbol{L}_U^{(1/4)} = \begin{bmatrix} 1 & -i & \cdots & 0 & 0 \\ i & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & -i \\ 0 & 0 & \cdots & i & 1 \end{bmatrix} \quad (7)$$

In our experiments, we use the degree-normalized counterpart $\boldsymbol{L}_N^{(q)} \coloneqq \mathbf{I} - \Big(\boldsymbol{D}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{D}_S^{-1/2}\Big) \odot \exp\big(i\boldsymbol{\Theta}^{(q)}\big)$

The $\exp\big(i\boldsymbol{\Theta}^{(q)}\big)$ term encodes the direction of edges. It resolves either to 1 if an edge exists in both directions ($\boldsymbol{A}_{u,v} - \boldsymbol{A}_{v,u} = 0$) or to $\exp(\pm i 2\pi q)$, with the sign encoding the edge direction. The *potential* q determines the ratio of real and imaginary part (recall $\exp(ia) = \cos(a) + i\sin(a)$). $\boldsymbol{A}_{u,v} = \boldsymbol{A}_{v,u} = 0$ is not of interest due to the multiplication with the zero entries in $\boldsymbol{A}_S$ or $\boldsymbol{D}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{D}_S^{-1/2}$. The Magnetic Laplacian contains the comb. Laplacian as a special case ($q = 0$). If the graph is undirected (i.e., $\boldsymbol{A} = \boldsymbol{A}^\top$), we obtain the comb. Laplacian for $q \le 0.25$. However, directed edges are purely imaginary if $q = {}^{1}\!/_{4}$ (see Eq. 7).

Although the Magnetic Laplacian has been used for a Spectral GNN (Zhang et al. 2021), Community Detection (Fanuel et al. 2016) or visualization of directed graphs (Fanuel et al. 2018), there is not much guidance on how to choose q and prior work in the computer science literature treats q as a (hyper-) parameter. Fortunately, the first eigenvector $\boldsymbol{\gamma}_0$ of Eq. 4 solves a relaxation of the so-called "angular synchronization problem" (Bandeira, Singer, and Spielman 2013):

$$\begin{aligned} \angle(\boldsymbol{\gamma}_0) \in \arg\min\nolimits_{\boldsymbol{\alpha} \in [0,2\pi)^n} \eta(\boldsymbol{\alpha}) \quad \text{with} \\ \eta(\boldsymbol{\alpha}) = \sum\nolimits_{u,v \in E} |\exp(i\boldsymbol{\alpha}_v) - \exp(i\boldsymbol{\alpha}_u + i\boldsymbol{\Theta}_{u,v})|^2 \end{aligned} \tag{8}$$

In angular synchronization, we seek the L^2 optimal estimate of n angles $\boldsymbol{\alpha}$ given m (noisy) measurements of phase offsets $\boldsymbol{\alpha}_i - \boldsymbol{\alpha}_j \mod 2\pi$ where $i, j \in \{0, 1, \dots, n-1\}$. Hence, each directed edge (u, v) encourages a rotation of the (otherwise constant) first eigenvector between nodes u and v. If there are no contradicting edges (i.e., in a tree) and node u is an ancestor of v, then the total phase shift amounts to $2\pi qh \mod 2\pi$ where h is the number of steps to reach v from u Additionally, if $\boldsymbol{A}_{u,v} = \boldsymbol{A}_{v,u} = 1$, a synchronization of the "angle difference" between nodes u and v is encouraged.

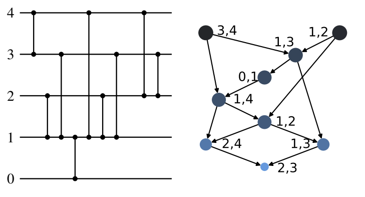
We choose $q = {q'}/{\min(\vec{m}, n, 1)}$, with *relative potential* $q' = 1/4$ and number of *purely directed edges* $\vec{m} = |\{(u, v) \in E \,|\, (v, u) \notin E\}|$. Then, the maximal angle difference between any two nodes is bounded $|\angle(\mathbf{\Gamma}_{v,0}, \mathbf{\Gamma}_{u,0})| \leq 1/2\, \pi$. In other words, the longest possible simple path is of length $\min(\vec{m}, n)$ (called h above). We empirically verify the choice of q' in Fig. 6. $q' = 1/4$ is among the best and for high values of q the performance drops severely (absolute $q > 0.05$).

Scale and rotation. Eigenvectors are neither guaranteed to be unique nor does Eq. 8 imply an absolute angle for any node (we only have relative "measurements"). In fact, if $\boldsymbol{\gamma}$ is an eigenvector of $\boldsymbol{L}$ then so is $c\boldsymbol{\gamma}$, even if $c \in \mathbb{C}$ with $|c| > 0$ (proof: $c\boldsymbol{L}\boldsymbol{\gamma} = c\boldsymbol{\lambda}\boldsymbol{\gamma} \implies \boldsymbol{L}(c\boldsymbol{\gamma}) = \boldsymbol{\lambda}(c\boldsymbol{\gamma})$). For real eigenvectors, there is the convention to choose $c \in \mathbb{R} \setminus \{0\}$ (possibly $c < 0$). Similarly, we choose the sign and rotate the eigenvectors according to a convention, that we find to work sufficiently well in our experiments. Our convention fixes the rotation for one of the source nodes – much like for sinusoidal encodings. First, we choose the maximum real magnitude of each eigenvector to be positive. This resolves the sign-ambiguity for the lowest eigenvector $\boldsymbol{\gamma}_0$. For the other eigenvectors, this convention fails to resolve the sign ambiguity if minimum and maximum have identical magnitude. Then, we rotate all eigenvectors s.t. $\boldsymbol{\gamma}_{v,u} = 0$ where we index a source node u based on the first eigenvector $u = \arg\max_{u'} \Im(\boldsymbol{\gamma}_{0,u'})$ with imaginary component $\Im$. Here we leverage our choice $q' \leq 1/4$, s.t., no neighboring node overflows the range $[-\pi, \pi]$ before determining u.

MagLapNet. Similarly as prior approaches (Lim et al. 2022; Kreuzer et al. 2021), we also preprocess the eigenvectors before using them as positional encodings. We consider the eigenvectors associated with the k lowest eigenvalues $\mathbf{\Gamma}_{:,:k}$ and treat k as hyperparameter. To tackle the sign ambiguity of eigenvectors $\mathbf{\Gamma}$, we follow a similar approach as SignNet (Lim et al. 2022) (excluding first eigenvector $\boldsymbol{\gamma}_0$). SignNet is invariant to sign changes since each eigenvector is processed like $f_{\text{elem}}(-\boldsymbol{\gamma}_j) + f_{\text{elem}}(\boldsymbol{\gamma}_j)$, where f_{elem} is permutation invariant over the nodes (e.g. MLP on each node or GNN; we use an MLP). The crucial piece is that $\boldsymbol{\gamma}_j$ is a complex number and that real and imaginary parts are processed jointly via f_{elem}. We stress that we use a lightweight network using $\approx 0.5\%$ of the total # parameters. After this first block, we stack the different components associated to each node and apply LayerNorm, Self-Attention, and Dropout. Similarly, to Kreuzer et al. (2021) we apply the self-attention independently for each node over the k eigenvectors. The last reshape stacks the positional encodings associated with each node and the MLP f_{re} matches the transformer dimensions.

4 Application: Sorting Networks' Correctness

Sorting networks (Knuth 1973) are directed graphs for a certain class of comparison based algorithms that sort any input sequence of fixed size with a sequence of comparators. Sorting networks are a particularly interesting application since they mark the middle ground between logical statements and source code. In fact, asserting their correctness is related to satisfiability. Interestingly, we use the sorting network task to make the implications of symmetrization (Laplacian encodings) and sequentialization (sinusoidal encodings) explicit.



(a) Sorting Net. (b) Dir. Graph

Figure 2: Example sorting network for sequence length 5. (a) shows typical illustration of sorting networks and (b) the directed graph with instructions as nodes.

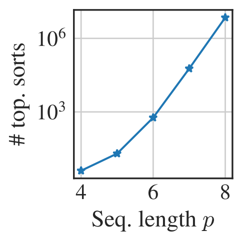


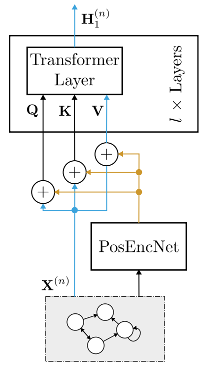
Figure 3: Number of topological sorts for Batcher even odd mergesort networks.

We consider sorting networks that consist of a sequence of conditional exchange instructions. In Fig. 2a, each horizontal line represents an element of the sequence of variables and the vertical lines are the comparators. Thus, a sorting network can also be expressed as a sequence of length n consisting of `v_i, v_j = sorted((v_i, v_j))` statements, where `v_i` and `v_j` are two of the p variables, i.e. $i, j \in \{0, 1, \ldots, p-1\}$. In our graph construction (Fig. 2b), we treat every instruction as a node with i and j as features (sinusoidal encoding). If a node operates on indices i and j, we add an edge from the last occurrences of i and j (if there are any). Thus, all nodes have exactly two incoming and two outgoing edges, except for the source and sink nodes. Due to the in-place operations or vice versa lack of buffers, the edges are typically connected to close ancestor nodes.

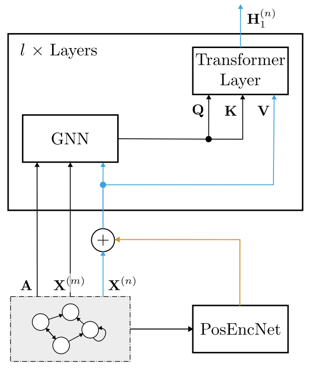
Directed graph vs. sequence. An important intuition is that each topological sort of the directed graph is an equivalent "program", which corresponds to a particular ordering of the statements, and there are many such semantically equivalent "programs" for the same graph. In Fig. 3, we show the number of topological sorts for a type of compact and deterministically constructed sorting networks for different input sequence lengths. For such a network and sequences length of just 8, the number of equivalent sequentializations already exceeds 1 million. Representing directed graphs as sequences, therefore, introduces a huge amount of arbitrary undesirable orderedness. Hence, using directed graph representations can significantly reduce the size of the input space compared to using a sequence representation as many equivalent sequences are collapsed into just one directed graph.

Symmetrization hurts. There exist correct and incorrect sorting networks that map to the same undirected graph. A model using such an undirected graph cannot distinguish these cases. For example, the *correct sorting network* for length three with comparators $[(0, 2), (0, 1), (1, 2)]$ and its reversed version (*incorrect*) map to the same undirected graph.

Dataset. We construct a dataset consisting of 800,000 training instances for equally probable sequence lengths $7 \leq p_{\text{train}} \leq 11$, generate the validation data with $p_{\text{val}} = 12$, and assess performance on sequence lengths $13 \leq p_{\text{test}} \leq 16$. We construct the sorting networks greedily until we have a correct sorting network. For this, we draw a random pair of comparators, excluding immediate duplicates and compara-



(a) W/o GNN



(b) W/ GNN

Figure 4: Architecture with optional GNN. (b) resembles the Structure Aware Transformer (SAT) (Chen, O'Bray, and Borgwardt 2022). For simplicity we only show the first of l layers. The subsequent layers directly operate on the input data besides the node embeddings $\boldsymbol{H}^{(k)}$ for $1 \leq k \leq l$, although, the $\boldsymbol{H}^{(l)}$ is then used for the actual prediction

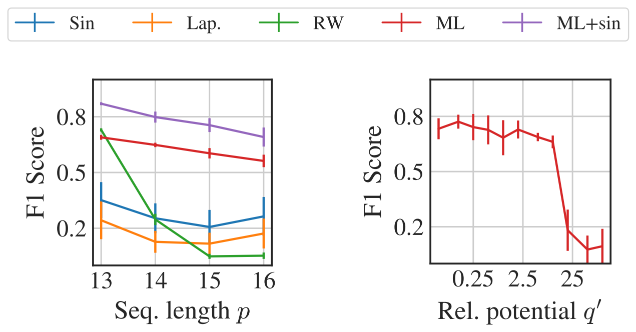


Figure 5: Sorting network: F1 over length p using transformer w/o GNN (see Fig. 4). 5 reruns with error of mean.

Figure 6: Sorting network: F1 score over relative potential values $q = {q'}/{\min(\vec{m}, n, 1)}$ with $k = 25$ eigenvectors.

tors between inputs that are already sorted. We then generate an incorrect example via omitting the last comparator (i.e., train is balanced). Moreover, we add additional incorrect sorting networks by reversing the directions of the correct networks to make the test set more challenging. Thus, the test and validation data consist of $1/3$ correct sorting networks (20,000) and $2/3$ of incorrect ones (40,000). The challenge of the task is therefore to generalize the correctness prediction to longer sequences and reversed sorting networks.

Models. We use a transformer that becomes structure-aware solely because of positional encodings (see Fig. 4a). We compare our Magnetic Laplacian (ML) positional encodings (§ 3) with a direction-aware random walk (RW)[2] as well as sinusoidal positional encodings (Sin) and eigenvectors of the combinatorial Laplacian (Lap.). Moreover, as we aim for general-purpose transformers for directed graphs, we do not explicitly study any heuristics that can be considered "directional". For example, for directed trees, it might be sufficient to have features for source and sink nodes.

Results. The eigenvectors of the Magnetic Laplacian (ML) outperform all other positional encodings as shown in Fig. 5 with just one exception. Thus, without bells and whistles, the Magnetic Laplacian can give a vanilla transformer a considerable structural awareness for directed graphs. On the other hand, the eigenvectors of the Laplacian barely outperform the naïve baseline that randomly chooses a class based on the prior probabilities. The random walk encodings seem to hold the middle ground between Laplacian and Magnetic Laplacian. Random Walk encodings are slightly better for short sequences but struggle to generalize to longer ones.

Sinusoidal encodings can perform poorly on their own. However, the sinusoidal encodings combined with the Magnetic Laplacian outperform all other encodings. At first glance, it appears surprising that including sinusoidal positional encodings helps (recall a directed graph reflects many equivalent sequences). We hypothesize that there are complementary challenges in understanding the directed graph in comparison to a single manifestation of the program. Having both information appears to bring certain synergies that the model can leverage for better generalization. The biggest drawback of adding sinusoidal encodings is that we sacrifice the robustness w.r.t. "meaningless reorderings". In other words, one has to trade-off accuracy and robustness.

[2] We use the random walk matrix $\boldsymbol{R} = \boldsymbol{A}\boldsymbol{D}^{-1}$ and perform 3 random walk steps in both directions also using $\boldsymbol{R}$ for $\boldsymbol{A}^\top$. An MLP then aggregates the landing probabilities for each node.

5 Application: Function Name Prediction

We consider the task of function name prediction since it is an established task in the graph learning community (Hu et al. 2020) where the direction of edges is important. Similar to sorting networks, each program is one particular ordering of the program statements, and there could be many equivalent programs with different orderings. Thus, it is surprising that graphs for source code used for machine learning are inherently sequential. Specifically, most (if not all) graphs in prior work connect nodes sequentially to reflect the order of statements in the source. For example, the Open Graph Benchmark Code2 dataset represents the 450,000 functions with its Abstract Syntax Tree (AST) and *sequential connections*. Since the input space of sequences can be much larger than the input space of directed graphs (see § 4), for some tasks such a graph construction is an unfortunate choice.

Our Graph Construction is inspired by Bieber et al. (2022). They also connect instructions sequentially, albeit, they handle control flow. Notable differences are: (a) We instead construct a Directed Acyclic Graph (DAG) for each "block" (e.g. body of if statement) that reflects the dependencies between instructions. We then connect the statements between blocks considering the control flow. (b) we address the (non-) commutative properties for basic python operations via edge features; (c) we do not reference the tokenized source code; (d) we omit the "last read" edges since these edges introduce unnecessary sequentialism; (e) otherwise we construct the graph similarly to OGB Code2 for comparability. For example, we aggregate child nodes containing only

	Position. Enc.	GNN	Test F1-Score	Val. F1-Score	
Sequen.	AST depth	✗	16.70±0.05	15.46±0.06	Prev.
		✓	19.37±0.09	17.73±0.07	
		✗	19.09±0.10	17.68±0.06	Ours
		✓	21.03±0.07	19.38±0.07	
Data-flow	AST depth	✓	21.61±0.12	19.79±0.11	
	Random walk	✗	19.34±0.08	**17.96±0.05**	
		✓	21.32±0.12	19.58±0.08	
	Magnetic Lap.	✗	**19.43±0.03**	17.83±0.05	
		✓	**22.22±0.10**	**20.44±0.06**	

Table 1: Results on the Open Graph Benchmark Code2 dataset. The first two rows correspond to prior work. All other approaches are our contribution. We report both architecture variants, with and without a GNN (see Fig. 4). We report the average and error of the mean over 10 reruns.

attributes into their parent's node attributes. In summary, we construct a data-flow-centric directed graph that is also able to handle the sharp bits like if-else, loops, and exceptions.

Results. In Table 1, we report the results on OGB Code2. Here we additionally compare to a transformer w/ GNN for query and key (see Fig. 4b), called Structure Aware Transformer (SAT) (Chen, O'Bray, and Borgwardt 2022). SAT (w/ GNN) was the prior state of the art prior to our work and we closely follow its data preprocessing.

SAT++. We improve the current state-of-the-art model with a number of small tricks (i.e., no new positional encoding yet). Our SAT++ improves the F1-score by 2.46% w/o GNN and by 1.73% with GNN. Besides smaller changes like replacing ReLU with GeLU activations, we most notably (1) add dropout on the sparsely populated node attributes and, (2) offset the softmax score to adjust for class imbalance of the special tokens for *unknown words* as well as *end of sequence*. We also replace the GCN with a three-layer GNN following Battaglia et al. (2018) (excluding a global state). The edge and node embeddings are updated sequentially and forward as well as backward messages are aggregated independently. Then, a Multi-Layer Perceptron (MLP) with two layers processes the concatenated embeddings (twice lower dimensionality as the transformer).

Data-flow. With our graph construction, we observe a consistent increase in predictive performance. For example, with the AST depth positional encodings and the SAT++ architecture (w/ GNN) the performance improves by almost 0.59% (relatively 3%). Moreover, we want to emphasize that due to the graph construction we additionally gain robustness w.r.t. certain reorderings of statements in the source code. We do not report results w/ AST depth but w/o GNN because this approach does not make use of the enhanced graph structure.

Transformer. Similarly to the sorting network task, we again observe that positional encodings can help improve the predictive performance (here applied additionally to the AST depth). W/o GNN, we find that random walks and the eigenvectors of the Magnetic Laplacian perform comparably.

Hybrid. The Magnetic Laplacian also helps in the presence of GNNs. *Our SAT++ with Magnetic Laplacian positional encodings marks the new state of the art on the Code2 dataset, outperforming SAT by 2.93% (relatively 15%).*

Surprisingly the Random Walk positional encodings even slightly decay performance. For the Code2 graphs, the GNN for query and key appears to be of great importance. We hypothesize that this is due to the sparsely populated node features. Only a few nodes are attributed and, additionally, the permitted vocabulary is restrictive. The local message passing might spread the information to neighboring nodes to adjust for this sparseness. Moreover, w/o GNN we do not make use of edge features.

6 Conclusion

Directed graphs can lower the effective input dimensionality, while symmetrization can hurt performance. Our positional encodings based on the Magnetic Laplacian help to address both effects and can improve the understanding of directed graphs. We see this as an important step towards true multi-purpose transformers for graphs universally handling undirected and directed graphs.

References

Bandeira, A. S.; Singer, A.; and Spielman, D. A. 2013. A Cheeger Inequality for the Graph Connection Laplacian.

Battaglia, P. W.; et al. 2018. Relational inductive biases, deep learning, and graph networks.

Bieber, D.; et al. 2022. A Library for Representing Python Programs as Graphs for Machine Learning.

Chen, D.; O'Bray, L.; and Borgwardt, K. 2022. Structure-Aware Transformer for Graph Repr. Learning. *ICML*.

Dwivedi, V. P.; and Bresson, X. 2021. A Generalization of Transformer Networks to Graphs. *DLG-AAAI 2021*.

Fanuel, M.; et al. 2016. Magnetic eigenmaps for community detection in directed networks.

Fanuel, M.; et al. 2018. Magnetic Eigenmaps for the visualization of directed networks. *Appl. Co. Harmon. Anal.*

Guo, D.; et al. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. *ICLR*.

Hu, W.; et al. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *NeurIPS*.

Knuth, D. E. 1973. *The art of computer programming, V. 3*.

Kool, W.; Hoof, H. v.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! *ICLR*.

Kreuzer, D.; et al. 2021. Rethinking Graph Transformers with Spectral Attention. *NeurIPS*.

Li, Y.; et al. 2022. Competition-Level Code Generation with AlphaCode.

Lim, D.; et al. 2022. Sign and Basis Invariant Networks for Spectral Graph Representation Learning.

Mialon, G.; Chen, D.; Selosse, M.; and Mairal, J. 2021. GraphiT: Encoding Graph Structure in Transformers.

Min, E.; et al. 2022. Transformer for Graphs: An Overview from Architecture Perspective.

Vaswani, A.; et al. 2017. Attention is all you need. *NeurIPS*.

Ying, C.; et al. 2021. Do Transformers Really Perform Bad for Graph Representation? *NeurIPS*.

Zhang, X.; et al. 2021. MagNet: A Neural Network for Directed Graphs. *NeurIPS*.