Multivariate de Bruijn Graphs: A Symbolic Graph Framework for Time Series Forecasting

Mert Onur Cakiroglu [1] **Idil Bilge Altun** [1] **Mehmet Dalkilic** [1] **Elham Buxton** [2] **Hasan Kurban** [3]

Abstract

Time series forecasting remains a challenging task for foundation models due to temporal heterogeneity, high dimensionality, and the lack of inherent symbolic structure. In this work, we propose DRAGON (Discrete Representation and Augmented Graph encoding Over de BruijN Graphs), a novel encoder that introduces Multivariate de Bruijn Graphs (MdBGs) to bridge the gap between symbolic representations and neural modeling. DRAGON discretizes continuous input sequences and maps them onto a fixed graph structure, enabling dynamic context recovery via graph-based attention. Integrated as an auxiliary module within a dual-branch architecture, DRAGON augments conventional CNN-based encoders with symbolic, structure-aware representations. All code developed for this study is available at: https://github.com/KurbanIntelligenceLab/MultdBG-Time-Series-Library

1. Introduction

Foundation models have made great strides in NLP (Zhao et al., 2023) and computer vision (Awais et al., 2023) by utilizing non-numeric, large-scale data; however, the same level of generalization has not emerged in the domain of time series where challenges exist *e.g.*, temporal heterogeneity, multivariate dependencies, and distant relationships.

Existing time series forecasting models rely heavily on the inputs being numeric, especially continuous (Ekambaram et al., 2024) (Nie et al., 2023) while missing out on recurring motifs that are central to many real-world temporal processes. Moreover, the common practice of creating fixed-size sliding window fragments from long sequences to facilitate supervised training limits the model's access to the global context outside the current fragment. To address these gaps, we introduce a new architecture that incorporates a symbolic graph-based structure into time series modeling through de Bruijn Graphs (dBG) (De Bruijn, 1946). dBGs are widely used in computational biology in applications such as genome assembly and k-mer–based (also known as k-tuple) sequence modeling to efficiently and compactly represent long sequences from overlapping fragments (Compeau et al., 2011). Recently, dBGs have also been explored in the context of time series modeling for capturing symbolic temporal patterns and enhancing forecasting performance (Cakiroglu et al., 2024b;a). Modeling time series data as dBG allows transforming continuous temporal data into a compact symbolic representation that captures recurring motifs and long-range dependencies across sequence fragments. We introduce a novel method for encoding multivariate time series as Multivariate de Bruijn Graphs (MdBGs) to capture patterns both within and across dimensions, allowing models to access cross sequence global context while processing each sliding window locally.

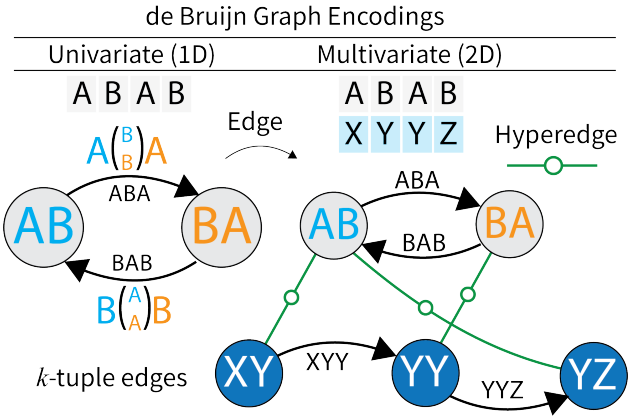

Figure 1. Illustration of de Bruijn Graphs for univariate and multivariate dimensions. (LEFT) The univariate dBG encodes k-tuples with directed and weighted edges in a single dimension. (RIGHT) The multivariate dBG extends this structure by incorporating edges (hyper-tuples) that connect k-tuples across multiple dimensions at the same time step, capturing inter-variable dependencies.

[1]Department of Computer Science, Indiana University Bloomington, IN, USA [2]Department of Computer Science, University of Illinois Springfield, IL, USA [3]College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar. Correspondence to: Hasan Kurban <hkurban@hbku.edu.qa>.

Proceedings of the 1st ICML Workshop on Foundation Models for Structured Data, Vancouver, Canada. 2025.

Our contributions are threefold: (i) MdBG, a multivariate extension of de Bruijn graphs, captures intra-, inter-dimensional, and long-range dependencies; (ii) DRAGON, an encoder built on MdBGs, generates task-specific time series embeddings and integrates into any model; and (iii) DRAGON outperforms state-of-the-art methods on forecasting benchmarks, including TimesNet (Wu et al., 2022).

2. Methods

2.1. Multivariate de Bruijn Graphs (MdBGs)

2.1.1. Data Processing

Let $\mathcal{D}^{\text{raw}} = \{\mathbf{X}_1^{\text{raw}}, \mathbf{X}_2^{\text{raw}}, \ldots, \mathbf{X}_D^{\text{raw}}\}$ be a time series data with D dimensions. Each raw sequence is defined as: $\mathbf{X}_i^{\text{raw}} = [x_1^i, x_2^i, \ldots, x_S^i]$, where $x_t^i \in \mathbb{R}$ and S is the sequence length. Since dBGs operate on categorical data, the raw input $\mathbf{X}_i^{\text{raw}}$ is first discretized using a discretization function $\text{DISC}_i(\mathbf{X}_i^{\text{raw}}, \alpha_i) \rightarrow \mathbf{X}_i^{\text{disc}}$. The set of bin sizes for all dimensions is denoted by $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, where α_i represents the bin size for dimension i. Similarly, the set of discretization functions is denoted by $\mathbf{F} = \{\text{DISC}_1, \text{DISC}_2, \ldots, \text{DISC}_D\}$. This formulation allows discretization to be customized independently for each dimension. The resulting discretized dataset, denoted as $\mathcal{D}$, consists of sequences defined as follows: $\mathcal{D}^{\text{disc}} = \{\mathbf{X}_1^{\text{disc}}, \mathbf{X}_2^{\text{disc}}, \ldots, \mathbf{X}_D^{\text{disc}}\}$, where $\mathbf{X}_i^{\text{disc}} = [c_1^i, c_2^i, \ldots, c_S^i], \quad c_t^i \in \{1, 2, \ldots, \alpha_i\}$. Each discretized element c_t^i represents a categorical value bounded by the discretization level α_i.

2.1.2. Graph Construction

A single MdBG is constructed from all sequences in the training set prior to model training. This allows the entire training set to be compactly represented as a graph and provides global context during model training. MdBG is a structured graph representation of both intra- and inter-dimensional dependencies of multivariate sequence data. It is formally defined as $G(V, E)$ where V is the set of nodes (fixed-length strings over some alphabet) and E is the set of directed edges (fixed-length strings over the same alphabet) where some overlapping substring of fixed size between two nodes. E is labeled with a non-negative number.

More formally, a dBG of order k is a directed graph where each node represents a unique univariate $(k-1)$-tuple (a subsequence of length $k-1$), and each edge corresponds to a k-tuple (length-k subsequence) observed in the input sequence. Specifically, the set of k-tuples $\mathbf{e}_t^i = (x_t^i, x_{t+1}^i, \ldots, x_{t+k-1}^i)$ induces a directed edge from the prefix node to the suffix node, $\mathbf{v}_{\text{prefix}} = (x_t^i, \ldots, x_{t+k-2}^i) \rightarrow \mathbf{v}_{\text{suffix}} = (x_{t+1}^i, \ldots, x_{t+k-1}^i)$ for $t = 1, 2, \ldots, (S-k+1)$. Each $\mathbf{e}_t^i$ defines a directed edge between two $(k-1)$-tuple nodes, capturing temporal transitions in the sequence. Repeated transitions increment the edge weights, encoding frequency. The construction of a dBG from univariate data is illustrated in Figure 1 (Left).

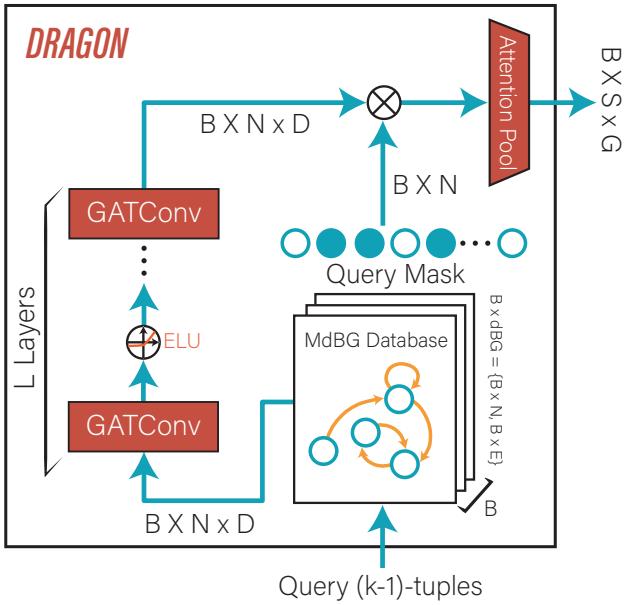

Figure 2. The DRAGON architecture employs a MdBG as a fixed graph database built from the entire training set. MdBG structure remains constant across batches and is encoded through L layers of Graph Attention Convolution (GATConv) with ELU activation. A node-level hard mask determines active nodes and is used to select encodings for the subgraph corresponding to the$(k-1)$-tuples in the input sequence. Only the node features are updated through masking. To obtain the final output, an attention pooling mechanism is applied that aggregates the graph-level features and reshapes them to match the input sequence length. B denotes the Batch size, D is the number of input dimensions, N is the number of MdBG nodes and E is the edge feature dimension.

Each node $\mathbf{v} \in V$ can correspond to a set of continuous k-tuples extracted from the original raw dataset $\mathcal{D}^{\text{raw}}$. This set is denoted as the feature space of node $\mathbf{v}$:

$$\mathcal{F}_{\mathbf{v}} = \{x_1^k, x_2^k, \ldots, x_m^k\},$$

where each $x_i^k \in \mathbb{R}^k$ and $1 \leq m \leq S-k+2$. This feature set $\mathcal{F}_{\mathbf{v}}$ captures the continuous embeddings associated with the discrete representation $\mathbf{v}$.

After constructing individual dBGs for each dimension, a set of D disconnected subgraphs $\textbf{MdBG} = \{G_1, G_2, \ldots, G_D\}$ is obtained, where each graph layer $G_i = (V_i, E_i)$ contains a set of nodes V_i representing all observed univariate $(k-1)$-tuples from dimension i, and directed edges E_i corresponding to observed k-tuples.

To capture inter-dimensional structure, hyper-tuple edges

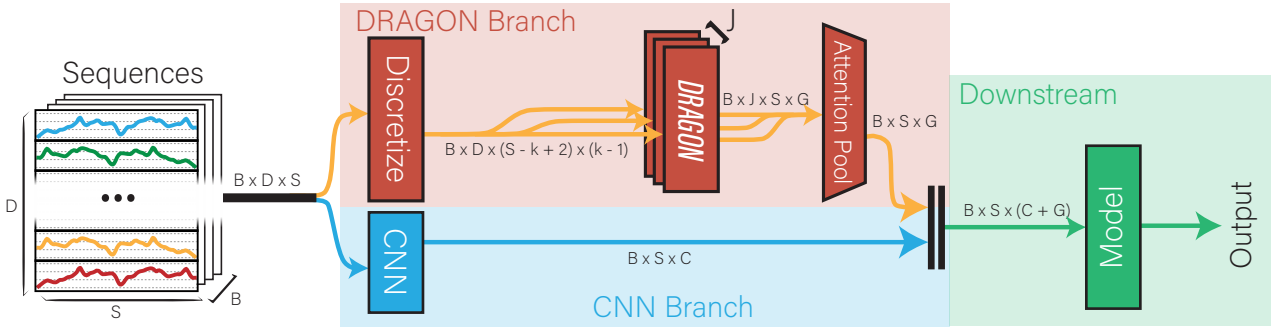

Figure 3. Overview of the proposed DRAGON architecture for multivariate time series modeling. The input sequence of shape $\mathbb{R}^{B\times D\times S}$ where B is the batch size, is processed through two parallel branches. In the top branch, the sequence is discretized and passed into the DRAGON encoder, which constructs MdBGs and generates graph-based embeddings of shape $\mathbb{R}^{B\times J\times S\times G}$ where J is the number of the DRAGON encoders. These embeddings are then aggregated via an attention pooling mechanism to produce the output of the shape $\mathbb{R}^{B\times S\times G}$. In the bottom branch, the original continuous sequence is fed into a 1D CNN encoder, yielding features of shape $\mathbb{R}^{B\times S\times C}$. The outputs from both branches are concatenated to form a joined representation of shape $\mathbb{R}^{B\times S\times (C+G)}$, which is then sent to a downstream model for prediction.

that connect nodes across different dimensions are defined. Given two nodes $\mathbf{v} \in V_i$ and $\mathbf{u} \in \mathcal{V}_j$, a bi-directional edge is established if their feature sets $\mathcal{F}_{\mathbf{v}_i}$ and $\mathcal{F}_{\mathbf{u}_j}$ contain raw k-tuples that occur at the same time step. This construction enables the graph to represent temporal dependencies across dimensions. As a result, for each multi-dimensional point $x \in \mathbb{R}^D$, a hyper-tuple clique of size D is formed by linking all co-occurring $(k-1)$-tuples across the D dimensions. This process is illustrated in Figure 1 (RIGHT). Note that the definition of MdBG is an extension of the original dBG; when $D = 1$, the MdBG is structurally equivalent to the standard dBG structure. An algorithm for constructing the MdBG is presented in Algorithm 1 in the Appendix.

Once MdBG is constructed from the training set, Graph Diffusion Convolution (GDC) is applied to further refine the structural properties of the MdBG and enhance information propagation across the constructed multivariate graph. GDC modifies the adjacency matrix by incorporating global diffusion patterns, effectively capturing long-range dependencies and smoothing over noisy or sparse connections. Specifically, we utilize Personalized PageRank (PPR)-based diffusion, which has demonstrated improved performance in node representation learning tasks by preserving both local and global graph structure (Gasteiger et al., 2022). This transformation improves the robustness and expressiveness of the learned node embeddings, which are later used in downstream tasks.

2.2. Training

During training, the DRAGON module encodes a subgraph of MdBG corresponding to the current input sequence. For each input sequence, query $(k-1)$-tuples are generated using a sliding window mechanism and subsequently discretized using the dimension-specific functions in the set $\mathbf{F} = \{\text{DISC}_1, \ldots, \text{DISC}_D\}$. These functions map continuous input sequences into categorical representations, yielding discretized tuples consistent with the MdBG vocabulary.

During training, all generated tuples are assumed to exist within the MdBG, the encoding for the nodes that correspond to the tuples in the input sequence can be simply retrieved. However, at test time, a query tuple $\hat{\mathbf{q}}$ may not appear in the MdBG node set $\mathcal{V}$, so the subgraph corresponding to the input sequence is approximated by selecting the most similar node based on L1 norm (Manhattan distance) from $\hat{\mathbf{q}}$. More specifically, masking vector $\mathbf{m}^{\mathbf{v}}$ over all nodes $\mathbf{v} \in \mathcal{V}$ is defined as follows:

$$\mathbf{m}^{\mathbf{v}} = \begin{cases} 1 & \text{if } \mathbf{v} = \hat{\mathbf{q}} \in \mathcal{V} \\ 1 & \text{if } \mathbf{v} = \arg\min_{u\in\mathcal{V}} \|\hat{\mathbf{q}} - u\|_1 \quad \text{and } \hat{\mathbf{q}} \notin \mathcal{V} \\ 0 & \text{otherwise} \end{cases}$$

This masking mechanism ensures that either the exact or the closest matching node is encoded for each $(k-1)$-tuple in the input sequence. The overall encoder architecture is illustrated in Figure 2.

The DRAGON module can be seamlessly integrated into most time series encoders as an auxiliary component, enhancing encoding by providing global context on the structure and recurring motifs present in the training set. It operates alongside the main architecture in a dual-branch design, with both branches later concatenated to form a fused representation. The overall process is illustrated in Figure 3.

Table 1. Average MSE and MAE per model across four datasets (ETTh1, ETTh2, ETTm1, ETTm2). Lowest values in each column are highlighted in bold. The DRAGON module achieves the lowest error rates in the majority of settings, demonstrating its effectiveness in recovering long-range dependencies from severely limited input contexts. These results are obtained using a fixed input length of 12 and illustrate the generalization capabilities of DRAGON in multivariate time series benchmarks.

	ETTh1		ETTh2		ETTm1		ETTm2	
Model	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
Transformer (Vaswani et al., 2017)	0.909	0.733	2.54	1.219	0.889	0.672	1.404	0.844
TSMixer (Lin et al., 2024)	1.002	0.756	2.262	1.236	0.916	0.690	0.907	0.723
Crossformer (Ekambaram et al., 2024)	0.691	0.609	1.982	1.060	**0.718**	0.588	1.005	0.675
FiLM (Zhou et al., 2022)	0.712	0.550	0.450	0.433	1.031	0.626	0.350	0.375
Nonstationary (Liu et al., 2022)	0.742	0.586	0.576	0.506	0.764	0.557	0.418	0.399
PatchTST (Nie et al., 2023)	0.642	0.525	0.448	0.432	1.014	0.624	0.347	0.373
TimeMixer (Wang et al., 2024)	0.633	0.519	0.447	0.434	0.942	0.601	0.343	0.370
Autoformer (Wu et al., 2021)	0.656	0.556	0.458	0.450	0.854	0.603	0.335	0.376
TimeXer (Lin et al., 2024)	0.613	0.512	**0.444**	**0.428**	0.904	0.589	0.336	0.362
TimesNet (Wu et al., 2022)	0.639	0.525	0.457	0.438	0.850	0.578	0.332	0.358
DRAGON (Ours)	**0.604**	**0.510**	0.446	0.430	0.742	**0.542**	**0.320**	**0.348**

(1) DRAGON Branch: The current batch is processed through one or more DRAGON encoders. Multiple DRAGON modules can be employed to capture graph representations with different configurations (e.g., varying k values or alphabet sizes A). The outputs from these modules are combined using an attention pooling mechanism, resulting in a unified graph-based feature representation of shape $\mathbb{R}^{B\times S\times G}$, where G denotes the embedding dimension of the DRAGON branch.

(2) Main Branch: The original continuous input sequence is passed through a main branch, which can be any standard time series encoder, such as a Transformer or CNN. In our design, we opt for a 1D CNN due to its simplicity and computational efficiency. This allows us to maintain a lightweight main branch, emphasizing the global contextual information captured by the DRAGON module. The CNN-based main branch yields an output of shape $\mathbb{R}^{B\times S\times C}$, where C is the number of channels.

The outputs from both branches are concatenated along the feature dimension, resulting in a fused encoding of shape $\mathbb{R}^{B\times S\times (C+G)}$. This combined representation is subsequently fed into the downstream decoder.

3. Results

To evaluate the effectiveness of the DRAGON module, a series of experiments is conducted using a narrow input context of length 12—an intentionally constrained setting designed to highlight the module's ability to recover temporal dependencies that span beyond the immediate input window. TimesNet is used as the base downstream model, with and without the DRAGON module, to assess the impact of our approach. The performance of the model is compared across four widely-used multivariate time series forecasting benchmarks: ETTh1, ETTh2, ETTm1, and ETTm2. The results are benchmarked against several state-of-the-art (SoTA) models as well as the baseline TimesNet model. A summary of the key results is presented in Table 1, and the complete results can be found in Appendix Table 4.

Our experiments show that the DRAGON module consistently improves performance, particularly in scenarios characterized by severely limited input context and extended forecasting horizon. By leveraging graph-based representations of historical patterns, DRAGON dynamically recovers missing temporal dependencies during inference. In addition to outperforming the TimesNet baseline, the DRAGON-enhanced variants often achieve SoTA performance when combined with TimesNet backbone, demonstrating its versatility and generalizability across forecasting architectures.

4. Conclusion

In this work, we introduced DRAGON, a novel encoder architecture that incorporates MdBGs into time series modeling. By discretizing continuous inputs and mapping them onto symbolic graph structures, DRAGON enables dynamic recovery of missing temporal context, especially in constrained input scenarios. Our results across four standard benchmarks show that DRAGON substantially improves performance when paired with a strong downstream model like TimesNet. The consistent gains highlight the effectiveness of symbolic graph representations in complementing neural architectures for time series analysis. DRAGON offers a new direction for time series modeling (one that bridges discrete structure and continuous representation) and sets the stage for more symbolic-aware foundation models in temporal domains.

References

Awais, M., Naseer, M., Khan, S., Anwer, R. M., Cholakkal, H., Shah, M., Yang, M.-H., and Khan, F. S. Foundational models defining a new era in vision: A survey and outlook. arXiv preprint arXiv:2307.13721, 2023.

Cakiroglu, M. O., Kurban, H., Aljihmani, L., Qaraqe, K., Petrovski, G., and Dalkilic, M. M. A reinforcement learning approach to effective forecasting of pediatric hypoglycemia in diabetes i patients using an extended de bruijn graph. Scientific Reports, 14(1): 31251, Dec 2024a. ISSN 2045-2322. doi: 10.1038/s41598-024-82649-4. URL https://doi.org/10.1038/s41598-024-82649-4.

Cakiroglu, M. O., Kurban, H., Buxton, E. K., and Dalkilic, M. A novel discrete time series representation with de bruijn graphs for enhanced forecasting using timesnet (extended abstract). In 2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA), pp. 1–3, 2024b. doi: 10.1109/DSAA61799.2024.10722826.

Compeau, P. E., Pevzner, P. A., and Tesler, G. Why are de bruijn graphs useful for genome assembly? Nature biotechnology, 29(11):987, 2011.

De Bruijn, N. G. A combinatorial problem. Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam, 49(7):758–764, 1946.

Ekambaram, V., Raju, G., Chatterjee, M., Kim, C., Ananthanarayanan, G., Bhatia, K., Ranganathan, P., and Rajpurkar, P. Tsmixer: An all-mlp architecture for time series forecasting. arXiv preprint arXiv:2306.14052, 2024.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric, 2019. URL https://arxiv.org/abs/1903.02428.

Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning, 2022. URL https://arxiv.org/abs/1911.05485.

Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J. (eds.), Proceedings of the 7th Python in Science Conference, pp. 11 – 15, Pasadena, CA USA, 2008.

Lin, K., Qiu, Y., Deng, Y., and Rush, A. M. A simple baseline for time series forecasting with llms, 2024. Available at https://arxiv.org/abs/2402.19072.

Liu, Y., Wu, H., Wang, J., and Long, M. Non-stationary transformers: Exploring the stationarity in time series forecasting. In Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=ucNDIDRNjjv.

Nie, Y., Li, H., Chen, Y., Lin, T.-Y., Wu, Y., Wang, Y., Xu, W., and Tang, J. A time series is worth 64 words: Long-term forecasting with transformers. arXiv preprint arXiv:2211.14730, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Advances in Neural Information Processing Systems, pp. 5998–6008, 2017.

Wang, S., Wu, H., Shi, X., Hu, T., Luo, H., Ma, L., Zhang, J. Y., and Zhou, J. Timemixer: Decomposable multiscale mixing for time series forecasting. In International Conference on Learning Representations (ICLR), 2024. URL https://openreview.net/forum?id=TIMEMIXER2024. Published at ICLR 2024.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Advances in neural information processing systems, 34:22419–22430, 2021.

Wu, H., Hu, T., Liu, Y., Zhou, H., Wang, J., and Long, M. Timesnet: Temporal 2d-variation modeling for general time series analysis. arXiv preprint arXiv:2210.02186, 2022.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.

Zhou, T., Ma, Z., xue wang, Wen, Q., Sun, L., Yao, T., Yin, W., and Jin, R. FiLM: Frequency improved legendre memory model for long-term time series forecasting. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), Advances in Neural Information Processing Systems, 2022. URL https://openreview.net/forum?id=zTQdHSQUQWc.

A. MdBG Feature Selection

In the Multivariate de Bruijn Graph (MdBG), nodes do not have a fixed feature set due to the discretized nature of the graph. Specifically, multiple raw $(k-1)$-tuples can map to the same node, resulting in feature sets of varying sizes, i.e., $1 \leq |\mathcal{F}_{\mathbf{v}}| \leq S - k + 2$. Some nodes may contain a large number of associated raw tuples in their feature set $\mathcal{F}_{\mathbf{v}}$. To handle this variability, a fixed number of $(k-1)$-tuples (denoted by f) from each node at every iteration is randomly sampled (with replacement). This strategy ensures computational tractability and encourages the model to explore diverse node features during training. $f = 16$ is set for our experimentation.

B. MdBG Construction Algorithm

The MdBG construction algorithm with the time complexity of $\mathcal{O}(m^2 \cdot T)$ is shown in Algorithm 1. The input consists of multivariate continous time series data $\mathcal{D}^{\text{raw}} = \{\mathbf{X}_1^{\text{raw}}, \ldots, \mathbf{X}_m^{\text{raw}}\}$, where each $\mathbf{X}_i^{\text{raw}}$ denotes a univariate sequence. k denotes the k-tuple length and the $\mathcal{F} = \{\text{DISC}_1, \ldots, \text{DISC}_m\}$ represents the discretization functions for each dimension which can be shared or unique. First, an empty graph is initialized $G = (\mathcal{V}, \mathcal{E})$ (line 2). For each input dimension i, the data in that dimension is discretized DISC_i (line 3-6). A sliding window of size k is applied to the discretized data to generate k-tuples R_i^t and K_i^t (lines 9-10). Extracted k-tuple is used to generate a prefix u_i and a suffix node v_i and its feature set F_{u_i} or F_{v_i} is initialized (lines 11–20). These feature sets are always updated to track which raw sequences contributed to each node (lines 21–22). If the edge (u_i, v_i) does not exist in $\mathcal{E}$, then it is initialized with the edge weight $w = 1$, otherwise w is incremented each time it is visited to reflect the frequency (lines 23–28). At the initial time step $t = 0$, all prefix nodes u_i across dimensions are connected using a bidirectional hyper-tuple edge to capture the inter-dimensional dependencies (lines 30–33). For all subsequent timesteps, hyper-tuple edges are formed between suffix nodes v_i as well (lines 35–37). Finally, the constructed MdBG (G) is returned (line 39).

Table 2. Dataset Partition Sizes

Dataset	**Train** (S)	**Validation**	**Test**	**Dimensions** (D)
ETTh1 & ETTh2	8,533	2,785	2,785	7
ETTm1 & ETTm2	34,453	11,425	11,425	7

Table 3. MdBG sizes constructed from training data for each α (20, 25, 30).

Dataset	**Nodes**	**Edges**	α
ETTh1	4,137 / 6,044 / 8,104	207,445 / 263,494 / 304,185	20 / 25 / 30
ETTm1	3,835 / 5,678 / 7,918	306,653 / 454,542 / 604,859	20 / 25 / 30
ETTm2	1,502 / 2,215 / 3,044	102,003 / 162,246 / 241,866	20 / 25 / 30
ETTh2	1,708 / 2,520 / 3,540	93,531 / 140,562 / 180,012	20 / 25 / 30

Algorithm 1 MdBG Construction Algorithm

Input : $\mathcal{D}^{\text{raw}} = \{\mathbf{X}_1^{\text{raw}}, \mathbf{X}_2^{\text{raw}}, \ldots, \mathbf{X}_m^{\text{raw}}\}$: Aligned multivariate time series,
$k \in \mathbb{N}$: Desired k-tuple length,
$\mathbf{F} = \{\text{DISC}_1, \ldots, \text{DISC}_m\}$: Discretizers per dimension
Output : $G = (\mathcal{V}, \mathcal{E})$: Constructed Multivariate de Bruijn Graph

Function `MdBGConstruction` $(\mathcal{D}^{\text{raw}}, k, \mathbf{F})$ **:**
  Initialize empty graph: $G \leftarrow (\mathcal{V} = \varnothing, \mathcal{E} = \varnothing)$
  for $i \leftarrow 1$ **in** m **do**
    $\text{DISC}_i \leftarrow \mathbf{F}[i]$
    $\mathbf{X}_i^{\text{disc}} \leftarrow \text{DISC}_i(\mathbf{X}_i^{\text{raw}})$
  end
  for $t \leftarrow 0$ **in** $T - k$ **do**
    for $i \leftarrow 1$ **in** m **do**
      $K_i^t \leftarrow \mathbf{X}_i^{\text{disc}}[t : t + k]$
      $R_i^t \leftarrow \mathbf{X}_i^{\text{raw}}[t : t + k]$
      $u_i \leftarrow (i, K_i^t[0 : k{-}1])$
      $v_i \leftarrow (i, K_i^t[1 : k])$
      if $u_i \notin \mathcal{V}$ **then**
        Add u_i to $\mathcal{V}$
        $\mathcal{F}_{u_i} = \varnothing$
      end
      if $v_i \notin \mathcal{V}$ **then**
        Add v_i to $\mathcal{V}$
        $\mathcal{F}_{v_i} = \varnothing$
      end
      $\mathcal{F}_{u_i} = \mathcal{F}_{u_i} \cup \{R_i^t[0 : k{-}1]\}$
      $\mathcal{F}_{v_i} = \mathcal{F}_{v_i} \cup \{R_i^t[1 : k]\}$
      if $(u_i, v_i) \notin \mathcal{E}$ **then**
        Add directed edge: $u_i \xrightarrow{w=1} v_i$
      end
      else
        Increment edge weight: $w \leftarrow w + 1$
      end
    end
    if $t = 0$ **then**
      forall distinct pairs (u_i, u_j) from prefix nodes **do**
        Add bidirectional hyperedge: $u_i \leftrightarrow u_j$ with type `hyper`
      end
    end
    forall distinct pairs (v_i, v_j) from suffix nodes **do**
      Add bidirectional hyperedge: $v_i \leftrightarrow v_j$ with type `hyper`
    end
  end
  return G
end

C. Experimental Details

C.1. Evaluation Metrics

To evaluate the performance of the constructed model, two standard regression metrics are used: mean squared error (MSE) and mean absolute error (MAE).

MSE is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MAE is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i and $\hat{y}_i$ denote the ground-truth and the predicted value, and n represents the total number of predictions.

C.2. Extended Experimental Results

In Table 4, MSE and MAE of state-of-the-art (SoTA) forecasting models across on four benchmark datasets with the prediction length $PL \in \{96, 192, 336, 720\}$ are presented. The DRAGON model architecture has shown improved performance over the original TimesNet, in most configuration settings.

All experiments use an input sequence length of 12. We employ three DRAGON modules with order $k = 4$ and discretization alphabet sizes $\alpha = 20$, 25, and 30, respectively for all layers/dimensions. Similarly, a uniform discretization function is used, which assigns equal-width bins across all features and dimensions for every layer. Each DRAGON module incorporates two layers of graph attention, and Graph Diffusion Convolution (GDC) is applied with a top-k value of 32. For the downstream TimesNet model, the "optimal" hyperparameters recommended for each benchmark dataset and forecasting horizon is adopted. The graph embedding dimension G is consistently set equal to the model channel size C. Our experimental pipeline, along with the hyperparameter configurations for all models, follows the TSLib framework (Wu et al., 2022), available at: `https://github.com/thuml/Time-Series-Library`.

The partition sizes for training, validation, and test splits across all datasets are summarized in Table 2. The resulting MdBG graph sizes (in terms of number of nodes and edges) for each alphabet size α are reported in Table 3.

The graph construction algorithm is implemented using the NetworkX library (Hagberg et al., 2008), and the resulting graphs are subsequently converted to data objects compatible with PyTorch Geometric (Fey & Lenssen, 2019). All experiments are conducted on a single NVIDIA A100 GPU with 42 GB of memory.

C.3. Future Work

We plan to extend the DRAGON framework in several promising directions. The first step is to improve its scalability in terms of graph construction and masking strategies to better accommodate long-term forecasting and high dimensional time series data. Subsequently, we aim to conduct more extensive experiments across diverse benchmark datasets from various domains and multiple downstream models to assess generalization. Beyond forecasting, we intend to transform DRAGON into a multimodal architecture by incorporating support for additional downstream tasks, including anomaly detection, classification, and imputation. Ultimately, our long-term goal is to evolve DRAGON into a foundation model by training it on large-scale, heterogeneous time series data in a self-supervised manner.

Table 4. MSE and MAE values of different models across different forecast horizons tested on four datasets. Best-performing values are highlighted in bold, and the second best is underlined.

		ETTh1		ETTh2		ETTm1		ETTm2	
model	**PL**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
Transformer	**96**	0.775	0.667	1.147	0.801	0.762	0.584	0.484	0.535
TSMixer		0.867	0.678	1.013	0.823	0.805	0.61	0.33	0.438
Crossformer		0.594	0.554	0.81	0.63	**0.618**	0.516	0.335	0.404
FiLM		0.669	0.521	0.349	0.37	0.971	0.591	0.234	0.309
Nonstationary		0.589	0.502	0.361	0.384	0.682	0.513	0.219	0.291
PatchTST		0.589	0.489	0.345	0.369	0.946	0.587	0.231	0.306
TimeMixer		0.583	0.484	0.34	0.367	0.886	0.568	0.225	0.301
Autoformer		0.635	0.531	0.353	0.383	0.778	0.565	0.212	0.304
TimeXer		**0.543**	**0.470**	**0.337**	**0.362**	0.833	0.551	0.22	0.295
TimesNet		0.59	0.495	0.345	0.366	0.679	0.506	0.201	0.276
DRAGON (Ours)		0.552	0.479	0.34	0.363	0.664	**0.499**	**0.2**	**0.275**
Transformer	**192**	0.819	0.675	3.095	1.312	0.772	0.623	1.052	0.77
TSMixer		0.975	0.741	2.419	1.303	0.87	0.66	0.49	0.549
Crossformer		0.631	0.578	1.667	0.899	**0.693**	0.576	0.438	0.474
FiLM		0.711	0.543	0.446	0.422	1.015	0.616	0.305	0.352
Nonstationary		0.665	0.547	0.504	0.469	0.752	0.548	0.292	0.33
PatchTST		0.637	0.516	0.442	0.421	0.995	0.613	0.303	0.35
TimeMixer		0.621	0.509	0.438	0.42	0.925	0.591	0.297	0.346
Autoformer		0.655	0.55	0.451	0.438	0.807	0.582	0.288	0.35
TimeXer		**0.593**	**0.499**	**0.437**	**0.417**	0.899	0.582	0.292	0.338
TimesNet		0.626	0.514	0.449	0.425	0.727	0.533	0.289	0.337
DRAGON (Ours)		0.601	0.505	0.44	0.417	0.701	**0.523**	**0.274**	**0.321**
Transformer	**336**	1.048	0.804	2.854	1.335	0.911	0.699	1.166	0.816
TSMixer		1.067	0.792	2.765	1.403	0.957	0.721	0.784	0.717
Crossformer		0.737	0.627	2.837	1.364	**0.774**	0.628	0.889	0.688
FiLM		0.74	0.563	0.502	0.464	1.054	0.638	0.377	0.392
Nonstationary		0.808	0.621	0.749	0.593	0.799	0.572	0.481	0.434
PatchTST		0.67	0.538	0.501	0.462	1.047	0.639	0.374	0.39
TimeMixer		0.659	0.532	**0.498**	0.465	0.963	0.613	0.372	0.389
Autoformer		0.657	0.564	0.515	0.483	0.9	0.627	0.366	0.395
TimeXer		0.646	0.527	0.498	**0.459**	0.919	0.599	0.363	0.379
TimesNet		0.657	0.533	0.512	0.472	0.892	0.599	**0.349**	**0.366**
DRAGON (Ours)		**0.636**	**0.525**	0.503	0.464	0.786	**0.564**	0.35	0.369
Transformer	**720**	0.995	0.786	3.062	1.429	1.11	0.782	2.914	1.256
TSMixer		1.1	0.813	2.849	1.414	1.03	0.768	2.023	1.188
Crossformer		0.801	0.678	2.614	1.345	**0.785**	0.633	2.357	1.133
FiLM		0.729	0.575	0.504	0.476	1.083	0.659	0.483	0.447
Nonstationary		0.905	0.675	0.69	0.578	0.822	0.595	0.68	0.54
PatchTST		0.67	0.556	0.502	0.475	1.069	0.658	0.481	0.446
TimeMixer		0.668	0.55	0.513	0.483	0.993	0.633	0.477	0.445
Autoformer		0.676	0.58	0.515	0.494	0.932	0.638	0.476	0.455
TimeXer		0.67	0.552	0.506	0.476	0.967	0.624	0.471	0.437
TimesNet		0.681	0.559	0.523	0.489	1.1	0.675	0.488	0.451
DRAGON (Ours)		**0.627**	**0.533**	**0.5**	**0.474**	0.819	**0.583**	**0.455**	**0.428**