SMART: A Surrogate Model for Predicting Application Runtime in Dragonfly Systems

Xin Wang*,[1] Pietro Lodi Rizzini*,[1] Sourav Medya,[1] Zhiling Lan[1,2]

[1]University of Illinois Chicago
[2]Argonne National Laboratory
{xwang823, plodir2, medya, zlan}@uic.edu

Abstract

The Dragonfly network, with its high-radix and low-diameter structure, is a leading interconnect in high-performance computing. A major challenge is workload interference on shared network links. Parallel discrete event simulation (PDES) is commonly used to analyze workload interference. However, high-fidelity PDES is computationally expensive, making it impractical for large-scale or real-time scenarios. Hybrid simulation that incorporates data-driven surrogate models offers a promising alternative, especially for forecasting application runtime, a task complicated by the dynamic behavior of network traffic. We present SMART, a surrogate model that combines graph neural networks (GNNs) and large language models (LLMs) to capture both spatial and temporal patterns from port level router data. SMART outperforms existing statistical and machine learning baselines, enabling accurate runtime prediction and supporting efficient hybrid simulation of Dragonfly networks.

Code — https://github.com/SPEAR-UIC/SMART
Datasets — https://zenodo.org/records/16667461
Extended version — https://arxiv.org/abs/2511.11111

Introduction

High-performance computing (HPC) systems play a vital role in numerous scientific domains, including climate modeling and drug discovery (Zhu and et al. 2014), where complex simulations and data-intensive computations are required. In these systems, the interconnect network serves as the central nervous system, facilitating data exchange among computer nodes and hence determining the system's overall performance. Among the various network topologies designed to meet these demands, the Dragonfly topology stands out as a high-radix, low-diameter solution that balances cost and performance (Kim and et al. 2008). Distinct from fat-tree networks, Dragonfly networks reduce infrastructure costs while maintaining high throughput and scalability. This effectiveness is reflected in their widespread adoption in the fast supercomputers, as evidenced by the latest Top500 list (11/2025), where six of the top ten supercomputers utilize Dragonfly interconnects (top500.org 2025).

In Dragonfly systems, a critical challenge is workload interference, which arises as multiple applications run concurrently on the same machine, producing highly dynamic and complex network traffic patterns. Parallel discrete event simulation (PDES) is widely used to model workload interference and network flows with high fidelity, i.e., at the flit-level granularity (Fujimoto 1990). CODES is a widely recognized PDES-based network simulator known for its ability to accurately model Dragonfly network behavior in detail(Mubarak et al. 2017; Carothers and et al. 2002; Wang et al. 2020). However, the computational complexity of high-fidelity PDES grows intractably, requiring the processing of billions of flit-level events across thousands of routers. As a result, hybrid simulation integrating data-driven surrogate models into PDES has become increasingly important for accelerating network modeling while preserving accuracy. One particular area of interest is constructing surrogate models to predict application runtime in Dragonfly systems.

Developing *data-driven surrogate models* for PDES is extremely challenging due to the highly dynamic nature of network traffic (e.g., at the millisecond scale) and the complex performance fluctuations inherent in large-scale systems. These challenges are exacerbated when multiple parallel applications run concurrently, competing for shared network resources, with iteration times that shift dynamically in response to changing network loads. Additionally, surrogate models must strike a delicate balance: they must achieve high accuracy while imposing minimal runtime overhead to be practical for real-time use. Such surrogate models hold significant potential to enhance networking systems by enabling improved decision-making in areas such as routing, congestion management, and resource allocation.

In this work, we explore advanced machine learning for surrogate modeling of Dragonfly interconnects, aiming to advance PDES and, in turn, contribute meaningful advancements to Dragonfly network research. The major contributions are summarized as follows.

- **Objective:** Our goal is to develop an accurate yet lightweight surrogate model for predicting application iteration times in Dragonfly. Such a model can not only accelerate networking simulations, but also contributes to improving networking decisions in Dragonfly systems.
- **Novel Approach:** We develop **SMART** (**S**urrogate **M**odel for Predicting **A**pplication **R**un**T**ime). Our design

*These authors contributed equally.

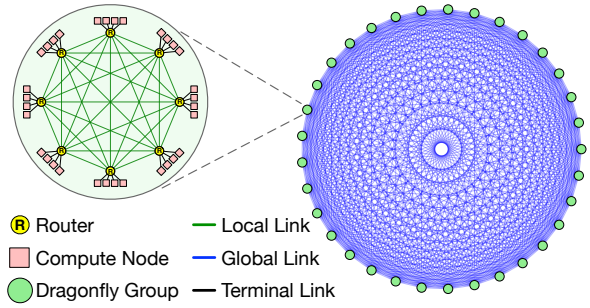

Figure 1: Topology of a 1,056-node 1D Dragonfly network.

leverages domain-specific knowledge of Dragonfly interconnects by incorporating both *topological structure* and *temporal dynamics* of network traffic to address the application runtime forecasting problem. Unlike the conventional algorithms, SMART integrates graph neural networks (GNNs) and large language models (LLMs) to holistically model the complex behavior of Dragonfly systems. Specifically, GNNs are employed to capture the hierarchical and spatial connectivity inherent in the Dragonfly topology, while LLMs are introduced to enhance temporal modeling through long-range pattern recognition and contextual prompting, a capability unavailable in traditional sequence models. To the best of our knowledge, this is the first effort of building a surrogate model for large-scale networking using LLMs and GNNs.

- **Experimental Results:** We conduct extensive experiments using two datasets generated from a 1,056-node Dragonfly system, where multiple representative HPC workloads are executed concurrently under various job placement strategies. Our method SMART significantly outperforms all the baselines in predictive accuracy. Moreover, SMART achieves an inference time of just 0.515 seconds, offering at least an order-of-magnitude speedup over the original simulation time.

Background & Related Work

Dragonfly Network Topology. The Dragonfly topology is adopted in exascale HPC systems, including TOP500 systems (top500.org 2025). It offers high bandwidth and low diameter while balancing scalability and cost. As shown in Figure 1, Dragonfly (Kim and et al. 2008) is organized into groups of routers connected via local links, with compute nodes attached via terminal ports. Groups are interconnected in an all-to-all pattern using global links, enabling low-hop communication across the system. We focus on the 1D Dragonfly variant, where routers within each group are fully connected. This design underlies Slingshot networks used in production systems such as Frontier and Aurora (ORNL 2022; Stevens and et al. 2019), making it a relevant target.

Hyrbid Network Simulation. Parallel discrete event simulation (PDES) is widely used for high fidelity modeling of large scale network systems. CODES, built on the ROSS engine, enables flit level simulation and has been applied to various HPC interconnect studies (Mubarak et al. 2017; Yang et al. 2016a,b; Wolfe et al. 2017).

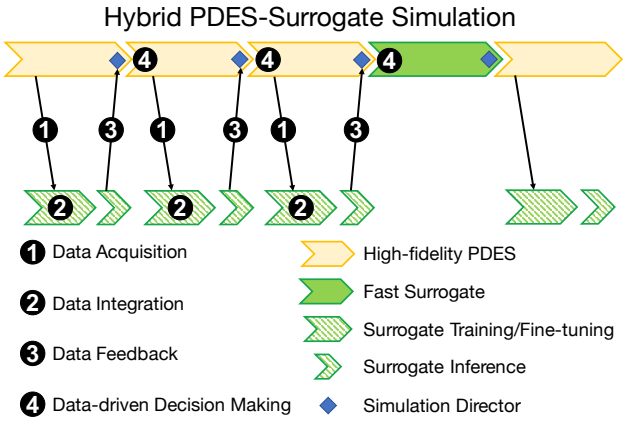

Figure 2: Hybrid PDES-surrogate simulation (Rizzini 2024).

Due to the high computational cost of PDES, hybrid approaches combine it with machine learning surrogate models to improve scalability (Cruz-Camacho et al. 2024). Figure 2, shows this process with four stages: (1) simulation data is collected, (2) used to train or fine tune the surrogate model, (3) the surrogate is validated through real time PDES output, and (4) a control loop decides whether to continue with PDES or switch to the faster surrogate. This adaptive feedback mechanism enables accurate yet efficient simulation.

Application Runtime Forecasting. Forecasting application runtime in large scale systems is a long standing challenge due to dynamic network conditions and application variability. Traditional time series models such as ARIMA, RNNs, and LSTMs capture temporal trends but often overlook spatial dependencies. Recent efforts have introduced GNNs to network modeling, focusing on metrics like delay and jitter. A step toward runtime prediction was taken by applying DCRNN to small Dragonfly systems, though its scalability is limited (Rizzini 2024). Our work advances this direction by integrating GNNs with large language models (LLMs) to jointly model spatial and temporal patterns in high fidelity HPC simulations. Unlike prior approaches designed for datacenter networks, SMART is tailored to support research on Dragonfly networks, a high-performance computing interconnect distinguished by its unique hierarchical design and adaptive routing capabilities.

Our Methodology

Networking Data

High-fidelity simulations generate two main datasets: (1) *application data*, which captures workload characteristics, and (2) *router data*, which records port-level features every 250 µs. These are used to construct a *temporal graph* representation.

The temporal graph is represented as a sequence $G_1, G_2, \ldots, G_T$, where each graph G_i corresponds to iteration i. The node set V and edge set E remain fixed, while node features change over time. In the graphs, *nodes* repre-

sent the router ports of the system, with some ports also representing the computing devices directly connected to them in the Dragonfly topology. *Edges* are defined in two ways: (1) fully connecting nodes accounting for ports of the same router and (2) connecting ports connected by global or local links. This results in a complete graph structure that captures the connectivity of a dragonfly system.

Node features. Each node is annotated with features that describe the network state at the corresponding port during each iteration. Since network snapshots are taken every 250 µs, we aggregate the snapshots over the iteration period by statistics like minimum, maximum, average or a quantile.

Active nodes are Dragonfly computing units executing workloads whose iteration times are being predicted. For these nodes, aggregation bounds are defined by the start and end timestamps of the iteration. For non-active nodes, the aggregation interval $[lb, ub)$ is determined by: (1) if one of the terminal ports is active, the iteration timestamps of that node define the bounds, (2) if all terminal ports are active, the bounds are set by the earliest start and the latest end timestamps across the iterations, (3) if none of the terminal ports is active, the bounds are derived from the nearest active node's timestamps.

Problem Definition

Let the Dragonfly network consist of n_r routers, N_c computer nodes, and application has N_p processes distributed across the computer nodes for execution. During execution, the N_p processes collaborate to complete T iterations. For each process $p \in \{1, 2, \ldots, N_p\}$ at a specific iteration $t \in \{1, 2, \ldots, T\}$, we define $y_{p,t}$ as the time taken for its application iteration and $x_{p,t}$ as the network characteristics of the router connected to the compute node hosting the rank p. To simplify the notation, we drop the subscript p in the following descriptions. We formally define the problem below.

Problem Statement. Given a sequence of iteration times y and network features x for process rank p over a look-back window, we define $\mathcal{B}$ as the input:

$$\mathcal{B} = \{y_{t-(L_y-1)}, y_{t-(L_y-2)}, \ldots, y_t; \\ x_{t-(L_x-1)}, x_{t-(L_x-2)}, \ldots, x_t\}$$

consisting of look-back windows of length L_y for y and L_x for x. The goal is to predict the future application iteration times at $t+1$, denoted as y_{t+1}.

Our Model: SMART

Our model, SMART, has three components, GNN, Transformer, and LLM, integrated for spatio-temporal modeling of workload contention and temporal variation in Dragonfly systems, which conventional graph or time series models cannot capture. Figure 3 shows the architecture.

GNN Encoder Our first goal is to capture the topological aspect of the dragonfly network with a graph neural network (GNN)-based encoder. The GNN encoder takes a sequence of temporal graphs $\{G_{t-T_{inGNN}}, G_{t-T_{inGNN}+1}, \ldots, G_{t-1}\}$ within a look-back window of length T_{inGNN} where each graph $G_t = (V, E, X^{(t)})$ represents the network state at iteration t. Here, V is the set of nodes (router ports), E is the set

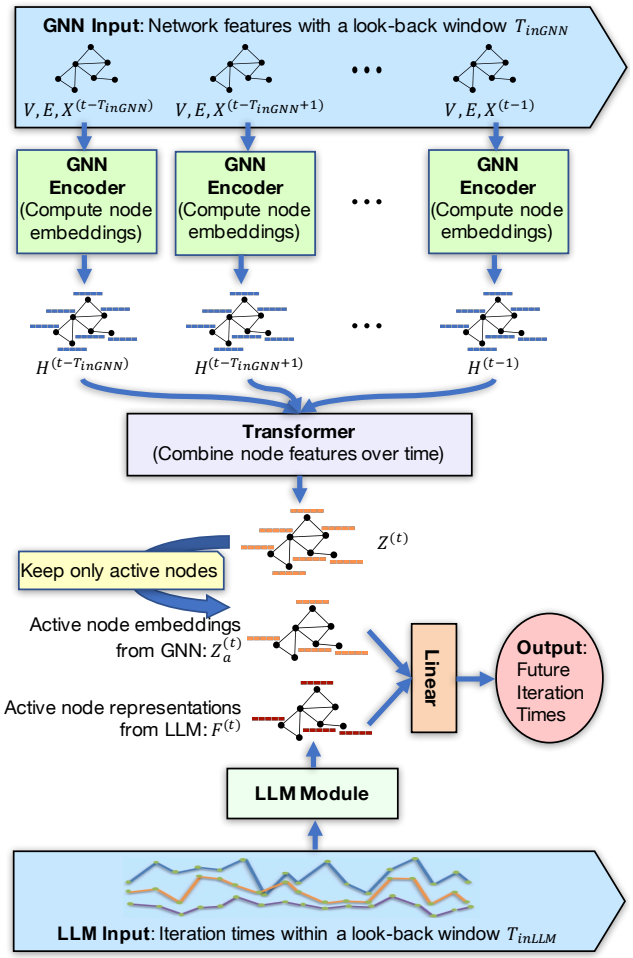

Figure 3: The architecture of our proposed model, SMART. The GNN Encoder (top) generates the node embeddings while the Transformer helps to produce node representations over time. The lower part shows the LLM-based component. The outputs of each component is concatenated node-wise and fed into a Linear layer for the final prediction.

of edges (connections between ports), and $X^{(t)} \in \mathbb{R}^{|V| \times d_f}$ is the node feature matrix at iteration t, with d_f being the number of features per node. The GNN Encoder is a Graph Convolutional Network (GCN) (Kipf and Welling 2017) which is applied to each graph G_t to capture individual spatial dependencies. The propagation rule for layer $l+1$ in GCN is given by:

$$H^{(l+1,t)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l,t)}W^{(l)}\right)$$

where l is the layer, t is the iteration number, $\tilde{A} = A + I_N$ is the adjacency matrix A with self-loops, $\tilde{D}$ is the diagonal degree matrix of $\tilde{A}$, $H^{(l)}$ represents the node feature matrix at layer l, with $H^{(0,t)} = X^{(t)}$, $W^{(l)}$ is the trainable weight matrix, $\sigma(\cdot)$ denotes the RELU activation function. The output of the GNN Encoder is a sequence of node embeddings that capture the spatial dependencies in the net-

work at each input iteration: $\{H^{(1)}, H^{(2)}, \ldots, H^{(T_{inGNN})}\}$, where $H^{(t)} \in \mathbb{R}^{|V| \times d_h}$ is the node embedding matrix at iteration t, and d_h is the dimensionality of the embeddings.

Temporal Transformer to Encode Temporal Dependencies Our next goal is to capture the temporal dependencies in a unified way. SMART utilizes a transformer (Vaswani et al. 2023) module to process the temporal graph embeddings $\{H^{(1)}, H^{(2)}, \ldots, H^{(T_{inGNN})}\}$ generated by the GNN encoder. Unlike traditional recurrent or convolutional neural networks, transformers are designed for sequence handling via self-attention. Here we employ transformer to capture the temporal dependencies in the graph embeddings from each iteration. The input to the transformer encoder is a sequence of node embeddings over time, enriched with positional encoding. The transformer module processes this input and applies an attention mechanism.

Time-dependent Attention. The core operation of the transformer module in SMART is the attention mechanism, which computes the importance of different parts of the input sequence. The input embedding of size (T_{inGNN}, d_m), where T_{inGNN} is the length of the input sequence and $d_m (= d_h \times |V|)$ is its dimensionality, is linearly transformed into three matrices: Query (Q), Key (K), and Value (V). These matrices are computed as:

$$Q = HW^Q, \quad K = HW^K, \quad V = HW^V$$

where H in our setting is as follows:

$$H = [H^{(t-T_{inGNN})}, H^{(t-T_{inGNN}+1)}, \ldots, H^{(t-1)}]$$

and $H \in \mathbb{R}^{T_{inGNN} \times |V| \times d_h}$. Moreover, W^Q, W^K, W^V are trainable weight matrices. Each attention head calculates scaled dot-product attention as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where d_k is the dimensionality of K. Multi-head attention is concatenation of the outputs of multiple attention heads:

$$\text{multihead}(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Here, $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and W_i^Q, W_i^K, W_i^V, and W_i^O are learnable weight matrices. This mechanism allows the transformer to focus on different parts of the sequence simultaneously to capture diverse dependencies. The output of the temporal transformer is a sequence of node embeddings that encode both spatial—by the GNN encoder—and temporal dependencies within the input features look-back window.

Output of the Transformer Module. The final output from the encoder is passed to the decoder, together with the last temporal node embeddings. These embeddings represent the most recent state of the nodes and provide critical information for predicting the next temporal embedding. In the decoder, the target sequence (last temporal node embeddings) is shifted to the right, ensuring that the model does not access the embedding it is trying to predict. The shifted target and the output of the encoder are then fed into the second and third layers, which correspond to successive encoder blocks, where K and V are derived from the encoder outputs, and Q is derived from the target. The output is finally passed to a dense layer in order to produce the temporal node embeddings. The final output from the transformer is $Z^{(t)} \in \mathbb{R}^{|V| \times d_z}$.

LLM-powered Forecasting We utilize a large language model (LLM)-based approach to further enhance our model, SMART. Unlike traditional sequence models, LLMs in SMART enable contextual prompting, incorporating task-specific and domain knowledge alongside time-series data. This helps capture long-range dependencies and improves runtime prediction across diverse workloads. To the best of our knowledge this is the first work that uses an LLM in the surrogate model to forecast application runtime. In SMART, this component uses the historical application iteration time data in as a sequence $\{y_{t-T_{inLLM}}, y_{t-T_{inLLM}+1}, \ldots, y_t\}$ within a pre-defined look-back window T_{inLLM}.

We closely follow the mechanism of Time-LLM (Jin et al. 2024). The steps of this module is as follows. The input sequence is first patched to normalize the input time series. Next we embed them via a *patch embedding* module into a dimension that is compatible with the LLM via a simple linear layer. The patches are then transformed via a multi-head attention layer to obtain the embeddings which are in the same space as the pre-trained LLM. Finally, the LLMs generate embeddings $E^{(t)} \in \mathbb{R}^{|V_a| \times d_{llm}}$ to represent the temporal patterns in the iteration times. Here, V_a represents the subset of V corresponding to the active nodes and d_{llm} is the number of hidden channels in the underlying LLM model.

Prompting. We use *Prompt-as-Prefix (PaP)* technique for prompting (Jin et al. 2024). In this technique, the patch embeddings are prefixed with prompts that contain context to the input, including task instructions, domain knowledge, and descriptive statistics about the time series. For example, statistics such as minimum, maximum, median values, and trends can be used for this purpose. We use the following prompt template given below, where $workload\_name$ is the name of the application workload, T_{inLLM} is the length of the historical input window to the LLM component.

Prompt for Forecasting

The dataset contains application iteration times for the [$workload\_name$] workload.
Task description: Forecast the next step given the previous [T_{inLLM}] steps information.
Input statistics: min value [...], max value [...], median value [...]

The input—which is composed of the prefix prompt and embedded patches—fed into the LLM. The obtained LLM output $E^{(t)} \in \mathbb{R}^{|V_a| \times d_{llm}}$ is flattened and linearly projected into an hidden dimension that matches the temporal node embeddings size from the transformer module, resulting in a final output $F^{(t)} \in \mathbb{R}^{|V_a| \times d_z}$.

Final Integration In the final step, we integrate GNN embeddings along with the LLM output effectively. The temporal embeddings $Z^{(t)} \in \mathbb{R}^{|V| \times d_z}$ from the Transformer are filtered by applying a node mask to ensure that only active nodes—such as those with target values—are considered in the merging process. The result of this operation is $Z_a^{(t)} \in \mathbb{R}^{|V_a| \times d_z}$. For each active node, the reduced-dimensional hidden state from the LLM is concatenated with the corresponding node embedding from the GNN. This creates a combined feature vector that encodes both spatial and temporal information powered by the LLM. The combined feature vector is passed through a fully connected layer to produce the final predictions ($\hat{y}_{t+1}$) for each node.Rather than manually tuning the balance between spatial and temporal modeling, SMART learns this integration end-to-end. The final prediction layer combines GNN-derived spatial features with LLM-derived temporal context, allowing the model to automatically determine their relative importance.

Online Tuning Strategy We develop an online tuning strategy to address the evolving nature of network traffic and workload behavior in Dragonfly systems. This approach is designed for hybrid simulation settings, where ground-truth data from PDES is intermittently available. The model is fine-tuned during inference only when such feedback is present, rather than assumed to be continuous. With a feedback loop, SMART adjusts its parameters to align with the latest network dynamics, thus maintaining prediction accuracy over time. This strategy is particularly beneficial in environments where workloads are heterogeneous and exhibit temporal variability. Our approach has two phases. The first one is the offline learning over a training dataset consisting of 30% of the available data. The second phase is the inference phase. Here we consider the ground truth data to be available for every F_t iterations. This implies that if the iteration index t is multiple of F_t, we use the data from $t-F_t$ to t to update the model weights via the usual back-propagation. In the following, $F_t = \infty$ means that the online tuning strategy is turned off, i.e. the inference phase does not incorporate any update of the model weights.

Experiments

We demonstrate the effectiveness of SMART in predicting application runtime on a 1,056-node Dragonfly system (Figure 1) with concurrently running applications competing for shared network resources. The code, datasets and extended results are available as noted after the abstract.

Experimental Setup

Data Generation. We evaluate SMART on two datasets generated using CODES simulations of a 1,056-node Dragonfly system. The dataset D_1 includes the MILC (MILC Collaboration 2011) and LAMMPS (Thompson et al. 2022) applications, together with the uniform random (UR) traffic. The dataset D_2 expands D_1 by adding a third workload, NN, which uses a 3D stencil pattern common in scientific simulations. For each dataset, we simulate two job placement strategies: *random*, where the compute nodes are scattered throughout the system, and *contiguous*, where the nodes are assigned sequentially. This yields four workload configurations: **D1-Rand**, **D1-Cont**, **D2-Rand**, and **D2-Cont**. Simulation details, workload breakdown, and hardware configurations are provided in the extended version of this paper.

Baselines. We use the following baselines: **(i) Last** is a heuristic that forecasts the next iteration time as the last historical available one; **(ii) Mean** forecasts based on the past W average iteration times; **(iii) LSTM** (Hochreiter and et al."urgen 1997)) predicts the next iteration time based on the historical iteration time values within a look-back window varied according to T_{inGNN}. **(iv) DCRNN-based:** This recent method (Rizzini 2024) uses a popular GNN-based architecture named as Diffusion Convolutional Recurrent Neural Network (DCRNN). See the extended version for details.

Performance Measures. To evaluate the performance of the models, we consider two key measures: (1) forecasting quality and (2) inference time. We use the standard Mean Absolute Percentage Error (MAPE) as the metric to assess quality, which is computed as: MAPE $= \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i-\hat{y}_i}{y_i}\right| * 100$ where y_i represents the actual iteration time, and $\hat{y}_i$ represents the predicted iteration time.

Other Settings. For each application, dataset and job placement strategy, we train and test the models tailored to predict the specific application iteration times. The hyperparameters used in the experiments are summarized in the extended version of this paper. We run the experiments for different combinations of $T_{inLLM} = 2, 4, 8$ and $T_{inGNN} = 2, 4, 8$ and tuning frequencies $F_T = \infty, 8$, where $F_T = \infty$ means that the model is never tuned during the testing, while $F_T = 8$ means that every 8 test iterations, the model weights are updated to allow the model to capture recent changes in the network patterns.

Results on Forecasting Quality

This section evaluates the forecasting quality of SMART alongside baseline models for predicting iteration time. Table 1 and Table 4 present the MAPE values for experiments conducted on datasets D_1 and D_2, respectively. The columns "MILC", "LAMMPS" and "NN" refer to models trained considering the nodes corresponding to the specific application as active. The columns "Cont" and "Rand" indicate whether the dataset follows a contiguous or random placement strategy. The cells between "T_{inLLM}" and "T_{inGNN}" refer to the results for the proposed SMART model, the other cells refer to the results for the baselines.

The results show that the best performance is obtained for $T_{inLLM} = 8, T_{inGNN} = 2$ in all the cases. This suggests the LLM effectively captures long-term temporal patterns while the GNN focuses on recent spatial dynamics. In addition, the online tuning strategy presented for the DCRNN model in (Rizzini 2024) appears to be beneficial. Incorporating an online tuning strategy ($F_t = 8$) consistently reduces forecasting errors compared to a static model ($F_t = \infty$). By updating the model weights every 8 iterations, SMART adapts to recent changes in network behavior, resulting in lower MAPE values in both data sets and placement types.

SMART			T_{inGNN}											
			2				4				8			
			Cont		Rand		Cont		Rand		Cont		Rand	
T_{inLLM}	Tuning frequency F_t		MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS
2	∞		3.98	2.64	3.9	2.54	4.30	2.64	4.17	2.59	4.64	2.88	4.30	2.67
	8		3.70	2.51	3.81	2.47	3.91	2.62	4.04	2.53	4.36	2.80	4.12	2.60
4	∞		3.87	1.90	3.72	1.81	4.22	1.94	3.91	1.85	4.28	2.04	3.96	1.90
	8		3.75	1.86	3.38	1.78	4.05	**1.92**	3.52	**1.82**	4.18	**1.97**	3.59	**1.88**
8	∞		3.50	1.89	3.40	1.87	4.04	2.06	3.67	1.89	3.90	2.05	3.71	1.95
	8		**3.19**	**1.78**	**3.12**	**1.84**	**3.76**	1.96	**3.34**	1.86	**3.74**	2.03	**3.44**	1.91
Baselines														
LSTM	∞		6.15	7.29	6.11	7.32	5.90	7.20	5.89	7.16	5.90	7.20	5.88	6.54
	8		6.09	7.25	6.05	7.28	5.82	7.15	5.81	7.11	5.79	6.48	5.83	6.53
DCRNN	∞		6.27	7.14	6.04	7.13	6.07	7.36	6.07	7.35	6.11	6.53	5.99	6.86
	8		6.35	7.10	5.98	7.10	5.97	7.34	5.97	7.33	6.00	6.49	5.88	6.82
LAST			7.86	8.14	7.83	8.21	7.86	8.14	7.83	8.21	7.86	8.14	7.83	8.21
MEAN			7.80	8.24	9.31	7.72	10.37	12.85	10.96	13.14	12.86	13.11	13.15	13.53

Table 1: MAPE comparison of SMART versus other baselines for different historical input windows T_{inGNN}, T_{inLLM} and retrain frequencies F_T over the simulation in D_1. Bold indicates the best result, underlined indicates the second best (column-wise). SMART consistently outperforms the baselines.

Model	Avg. Inference Time
SMART	0.5150
LLM-only	0.4158
GNN-only	0.0633
MEAN	0.00001
LAST	$<$ 0.00001
LSTM	0.0398
DCRNN	0.0459

Table 2: *Average inference time* (in seconds). Benchmarked on an 80-core ARM system with 2×A100 GPUs. All models, including SMART, achieve inference times significantly lower than the simulation per-iteration times (Table 3).

Dataset	Application	Contiguous	Random
D_1	MILC	79.36	132.34
	LAMMPS	56.47	243.48
D_2	MILC	65.46	112.42
	LAMMPS	47.06	216.58
	NN	8.09	14.98

Table 3: *Simulation time* per iteration (seconds) for each application in D_1 and D_2 under different job placements. All surrogate models (Table 2) are orders of magnitude faster, enabling integration with PDES.

Discussion. Achieving 100% accuracy in practice is unrealistic, particularly in highly dynamic networking environments of large-scale systems. In a related study, researchers demonstrated that a hybrid PDES using a MEAN-based surrogate model could achieve comparable simulation performance (Cruz-Camacho et al. 2024).

Our results show that SMART significantly outperforms MEAN across various scenarios. Prediction accuracy varies across different applications, with LAMMPS exhibiting higher precision than MILC. This discrepancy arises from their distinct sensitivities to network latency: LAMMPS is moderately sensitive, while MILC, with frequent global communication, is highly latency-sensitive. As a result, their sensitivity to application communication prediction also differs, affecting the accuracy of surrogate models. *Overall, our model outperforms in prediction accuracy by effectively incorporating domain-specific knowledge of both application characteristics and network architecture.*

Results on Inference Time

Table 2 shows the average inference times for all models, demonstrating that they are orders of magnitude faster than traditional PDES simulations. Inference times range from under 0.00001 seconds (LAST) to 0.5150 seconds (SMART). In contrast, Table 3 shows that PDES iteration times span from 8.09 to 243.48 seconds, depending on the application and job placement. Even the slowest surrogate model (SMART) requires less than 1% of the shortest PDES iteration time and under 0.2% of the longest, making real-time use feasible. These results confirm the surrogate models are lightweight and suitable for integration into time-sensitive simulation workflows.

Discussion. A practical surrogate model must execute efficiently to benefit large-scale simulations. While traditional

SMART		T_{inGNN}																	
		2						4						8					
		Cont			Rand			Cont			Rand			Cont			Rand		
T_{inLLM}	Tuning frequency F_t	MILC	LAMMPS	NN	MILC	LAMMPS	NN	MILC	LAMMPS	NN	MILC	LAMMPS	NN	MILC	LAMMPS	NN	MILC	LAMMPS	NN
2	∞	4.29	2.81	3.30	4.04	2.63	3.05	4.70	3.04	3.49	4.24	2.71	3.29	4.92	3.14	3.69	4.37	2.82	3.39
	8	4.12	2.72	3.21	3.98	2.58	2.97	4.39	2.95	3.30	4.18	2.63	3.24	4.75	3.02	3.71	4.28	2.71	3.33
4	∞	4.03	1.99	3.21	3.92	1.92	2.88	4.64	**2.13**	**3.19**	4.08	1.97	2.97	4.63	2.20	3.40	4.12	2.05	3.06
	8	4.11	1.98	2.89	3.83	1.90	2.85	4.59	2.23	3.26	3.96	1.96	2.91	4.60	2.28	3.34	4.08	1.99	2.97
8	∞	3.91	1.93	3.07	3.51	1.89	2.85	4.17	2.26	3.22	3.79	1.91	2.93	4.19	2.20	3.23	3.87	1.97	2.99
	8	**3.51**	**1.88**	**2.87**	**3.39**	**1.86**	**2.79**	**3.84**	2.20	3.30	**3.63**	**1.88**	**2.82**	**4.17**	**2.14**	**3.22**	**3.74**	**1.93**	**2.90**
Baselines																			
LSTM	∞	6.23	8.14	6.96	6.23	7.45	6.92	6.97	8.28	7.66	5.72	7.11	6.81	6.32	7.95	7.81	5.59	6.46	7.21
	8	6.28	8.02	7.38	6.17	7.41	6.82	6.85	8.80	8.12	5.65	7.06	6.76	6.37	8.02	7.55	5.51	6.40	7.15
DCRNN	∞	6.54	7.54	7.22	6.13	7.25	6.83	6.78	8.47	8.13	5.85	7.21	6.87	6.65	8.43	7.87	5.85	6.42	7.17
	8	6.79	7.32	7.49	6.07	7.21	6.79	7.16	8.06	7.65	5.75	7.20	6.80	6.57	8.31	7.50	5.74	6.38	7.10
LAST		8.73	9.11	8.97	8.84	9.03	8.68	8.73	9.11	8.97	8.84	9.03	8.68	8.73	9.11	8.97	8.84	9.03	8.68
MEAN		8.20	9.00	8.63	7.75	8.18	7.71	8.89	9.63	8.68	11.32	13.17	9.83	12.51	15.24	11.26	13.46	13.70	9.76

Table 4: MAPE comparison for different historical input windows T_{inGNN}, T_{inLLM} and retrain frequencies F_T over the simulation in D_2. Bold indicates the best result, underlined indicates the second best (column-wise). SMART consistently outperforms the baselines; the best result is often obtained with longer LLM input window ($T_{inLLM} = 8$).

GNN-only	$T_{inGNN} = 2$				$T_{inGNN} = 4$				$T_{inGNN} = 8$			
	Cont		Rand		Cont		Rand		Cont		Rand	
F_t	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS	MILC	LAMMPS
∞	3.68	2.14	3.81	4.78	4.15	2.31	3.98	4.74	4.30	2.36	4.96	4.76
8	3.46	2.05	3.58	4.59	3.86	2.26	3.70	4.65	3.95	2.30	4.56	4.64

Table 5: MAPE results for GNN-only under different retrain frequencies F_T for the simulation in D_1.

	Cont		Rand	
	MILC	LAMMPS	MILC	LAMMPS
$T_{inLLM} = 2$	4.23	2.77	4.24	2.77
$T_{inLLM} = 4$	4.36	1.98	4.33	1.90
$T_{inLLM} = 8$	4.02	1.94	3.97	1.91

Table 6: MAPE results for LLM-only for different T_{inLLM} for the simulation in D_1.

simulations can take hours or days to complete, SMART performs inference in just 0.5150 seconds per step, making hybrid simulation both efficient and feasible.

Ablation Study

We evaluate the contributions of the two main components of SMART: the GNN encoder and the Time-LLM module.

First, the GNN-only variant, which removes the LLM component and relies solely on the GNN encoder and temporal transformer, shows significantly degraded performance on D_1 (Table 5), indicating that LLM plays a critical role in capturing temporal patterns. Accuracy also improves when using a shorter online tuning window ($F_T = 8$).

Next, the LLM-only variant excludes the GNN and uses only temporal input to predict future iteration times based on a look-back window T_{inLLM}. This model also underperforms SMART on D_1 (Table 6), underscoring the importance of spatial features. Longer input windows generally lead to better accuracy by allowing the model to leverage more historical data. Results on D_2 for both variants are provided in the extended version.

Finally, we assess alternative temporal models by replacing Time-LLM with Autoformer, DLinear and PatchTST. SMART consistently achieves lower prediction errors across workloads, as detailed in the extended version of this paper.

Conclusion

We presented SMART, a domain-aware surrogate model designed to accelerate parallel discrete event simulation for Dragonfly interconnects. By integrating GNNs to model the hierarchical topology of Dragonfly networks and LLMs to capture temporal traffic dynamics, SMART effectively learns both the spatial structure and temporal behavior intrinsic to Dragonfly systems. Evaluated on 1,056-node Dragonfly simulations, SMART consistently outperforms baseline methods in prediction accuracy and delivers sub-second inference, reducing simulation time from hours to seconds.

Although this study centers on Dragonfly systems, the modular architecture of SMART allows for straightforward adaptation to other network topologies. For instance, applying SMART to alternative architectures like Fat Tree requires only retraining with new graph-structured inputs, without altering the core model. In future work, we plan to extend the model's forecasting horizon and explore its generalization across network types.

Acknowledgments

This work was supported in part by the U.S. Department of Energy under Contract No. DE-SC0024271 and by the U.S. National Science Foundation under Grants OAC-2402901 and CCF-2515009. The authors thank Kevin Brown and Rob Ross at Argonne National Laboratory and Chris Carothers at RPI for their helpful feedback and discussions.

References

Carothers, C.; and et al. 2002. ROSS: A high-performance, low-memory, modular Time Warp system. *JPDC*, 62: 1648–1669.

Cruz-Camacho, E.; Brown, K.; Wang, X.; Xu, X.; Shu, K.; Lan, Z.; Ross, B.; and Carothers, C. 2024. Hybrid PDES Simulation of HPC Networks Using Zombie Packets. *ACM Trans. Model. Comput. Simul.*

Fujimoto, R. M. 1990. Parallel discrete event simulation. *Commun. ACM*, 33(10): 30–53.

Hochreiter, S.; and et al."urgen. 1997. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780.

Jin, M.; Wang, S.; Ma, L.; Chu, Z.; Zhang, J. Y.; Shi, X.; Chen, P.-Y.; Liang, Y.; Li, Y.-F.; Pan, S.; and Wen, Q. 2024. Time-LLM: Time Series Forecasting by Reprogramming Large Language Models. arXiv:2310.01728.

Kim, J.; and et al. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *2008 ISCA*, 77–88.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907.

MILC Collaboration. 2011. MILC Benchmark. https://web.physics.utah.edu/~detar/milc/. Accessed: 2024-11-13.

Mubarak, M.; Carothers, C. D.; Ross, R. B.; and Carns, P. 2017. Enabling parallel simulation of large-scale HPC network systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1): 87–100.

ORNL. 2022. Frontier Supercomputer. https://www.olcf.ornl.gov/frontier/. Accessed: 2025-06-27.

Rizzini, P. L. 2024. *Predictive Modeling of Application Runtime in Dragonfly Systems*. Master's thesis, University of Illinois Chicago.

Stevens, R.; and et al. 2019. Aurora: Argonne's next-generation exascale supercomputer. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States).

Thompson, A. P.; Aktulga, H. M.; Berger, R.; Bolintineanu, D. S.; Brown, W. M.; Crozier, P. S.; In't Veld, P. J.; Kohlmeyer, A.; Moore, S. G.; Nguyen, T. D.; et al. 2022. LAMMPS-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer physics communications*, 271: 108171.

top500.org. 2025. Top500 list. https://www.top500.org/lists/top500/2025/11/. Accessed: 2025-12-15.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2023. Attention Is All You Need. arXiv:1706.03762.

Wang, X.; Mubarak, M.; Kang, Y.; Ross, R. B.; and Lan, Z. 2020. Union: An Automatic Workload Manager for Accelerating Network Simulation. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 821–830.

Wolfe, N.; Mubarak, M.; Jain, N.; Domke, J.; Bhatele, A.; Carothers, C. D.; and Ross, R. B. 2017. Preliminary performance analysis of multi-rail fat-tree networks. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 258–261. IEEE.

Yang, X.; Jenkins, J.; Mubarak, M.; Ross, R. B.; and Lan, Z. 2016a. Watch out for the bully! job interference study on dragonfly network. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 750–760. IEEE.

Yang, X.; Jenkins, J.; Mubarak, M.; Wang, X.; Ross, R. B.; and Lan, Z. 2016b. Study of intra-and interjob interference on torus networks. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, 239–246. IEEE.

Zhu, D.; and et al. 2014. Developing A High Performance Computing (Hpc) Based Hydrological Modelling Framework To Support Extreme Weather Impact Studies.