Leveraging LLMs to Close Simulation-to-Reality Gap in Wireless Mesh Network Configuration

Jean Tonday Rodriguez*, Dipan Sadekeen*, Mo Sha, Mohammad Ashiqur Rahman
Knight Foundation School of Computing and Information Sciences
Florida International University
{jtond004,dsade003,msha,marahman}@fiu.edu

***Abstract*—Recent years have witnessed the rapid adoption of wireless mesh networks (WMNs) across various fields, including industrial automation, smart energy, and smart cities. Although WMNs usually perform well thanks to years of research, configuring these networks remains a challenging task. Simulations provide distinct advantages over physical deployment for network configuration, as they can be run more quickly, with much less overhead, and allow testing different configurations under controlled, identical conditions. However, recent studies have shown that the network configuration chosen in simulations may not perform well in real-world deployments due to the simulation-to-reality gap. Unfortunately, all existing solutions require collecting data from the physical deployment to effectively narrow the simulation-to-reality gap, a costly and labor-intensive process. In this paper, we present LLM-OPT, the first solution that produces high-quality network configuration models for WMNs without requiring data from physical deployments. LLM-OPT employs feature masking and textual serialization techniques to preprocess the simulation data and leverages LLM fine-tuning to generate network configuration models. Experimental results show that the network configuration model generated by LLM-OPT achieves 82% prediction accuracy and significantly outperforms all existing solutions (up to 34.2%).**

***Index Terms*—Wireless Mesh Networks, Network Configuration, Simulation-to-Reality Gap, Large Language Models**

I. Introduction

Recent years have witnessed the rapid adoption of wireless mesh networks (WMNs) across various fields, including industrial automation, smart energy, and smart cities. One notable example is the increasing use of IEEE 802.15.4-based WMNs in process industries over the past two decades, largely due to their ability to reduce operational costs [1]. Battery-powered wireless modules can be easily and cost-effectively integrated into existing industrial sensors and actuators, eliminating the need for costly wiring for both communication and power. Several standards organizations, including FieldComm [2], ISA [3], and CSA [4], are actively promoting the deployment of industrial WMNs in the real world. For example, Emerson Process Management has deployed more than 54,835 WMNs that run the WirelessHART standard worldwide to monitor and control industrial processes [5].

Although WMNs usually perform well thanks to years of research, configuring these networks remains a challenging task. Setting up a WMN is a complex process that involves theoretical computations, simulations, field testing, and more. Simulations provide distinct advantages over physical deployment for network configuration because they can be conducted more quickly, with less overhead, and allow testing different configurations under controlled, identical conditions. However, recent studies have shown that the network configuration chosen via simulations may not perform well in real-world deployment due to the simulation-to-reality gap in network configuration [6]. Such observations motivate recent research efforts to develop new solutions to close the simulation-to-reality gap [6], [7], [8], [9]. Unfortunately, all existing solutions require collecting data from the physical deployment to narrow the gap and ensure effective transfer learning. This data collection process can be very costly and labor-intensive in many industrial facilities [6], significantly compromising the benefit of using simulations to configure WMNs. Therefore, there is a critical need for new solutions that can close the simulation-to-reality gap without requiring data collection from physical deployments.

In recent years, significant efforts have been made in the machine learning community to explore how to ensure effective cross-modality training and transfer knowledge learned from the source domain to the target domain with minimal target data. For instance, large language models (LLMs) characterized by their extensive pre-training and contextual analysis abilities have successfully produced good prediction models for data-scarce domains in the context of natural language processing [10], [11] and numerical problem solving [12], [13], which motivates us to explore their usage as a predictive engine for WMN configuration. Unlike existing approaches such as DANN [6], WMN-CDA [9], and MARIA [8], which rely on deep learning–based domain adaptation techniques, the use of large language models for cross-domain generalization in WMN configuration remains largely unexplored. In this paper, we first present an empirical study that investigates the feasibility of feature masking and text serialization techniques to close the simulation-to-reality gap in WMN configuration. We then introduce LLM-OPT, a novel network configuration solution that leverages LLM fine-tuning to produce good network configuration models. To our knowledge, LLM-OPT is the first solution to produce high-quality network configuration models for WMNs without requiring data from physical deployments. The benefits of LLM-OPT do not come solely from

*These authors contributed equally to this work.

pure data serialization, but also from leveraging a pretrained transformer. By leveraging LLMs' pretrained representations and our masking approach, we promote reliance on domain-transferable features. Our main contributions are as follows:

- We conduct an empirical study to explore the effects of feature masking and text serialization techniques on closing the simulation-to-reality gap in WMN configuration.
- We develop LLM-OPT, the first solution to produce high-quality network configuration models for WMNs without requiring data from physical deployments.
- We implement LLM-OPT and compare its performance against state-of-the-art baselines. Experimental results show that the network configuration model generated by LLM-OPT achieves 82% prediction accuracy and significantly outperforms all existing solutions (up to 34.2%).

Our paper is organized as follows. Section II introduces our empirical study. Section III presents the design of LLM-OPT. Section IV evaluates LLM-OPT. Section V reviews the related work. Section VI concludes this paper.

II. Empirical Study

In this section, we first introduce the datasets used in our empirical study and the problem formulation. We then present the limitations of existing solutions and the insights gathered from our study.

A. Dataset and Problem Formulation

We use two publicly accessible datasets created by Shi et al. [6], [14] for our study. The first dataset, target domain data (D^t), consists of performance measurements (i.e., end-to-end network latency L, battery life B, and end-to-end reliability R) collected from a 50-device network running the WirelessHART standard across 88 network configurations. The 88 network configurations are created by combining the values of three tunable network parameters: the packet reception ratio (PRR) threshold for link selection, P, the number of channels used in the network, C, and the number of transmission attempts scheduled for each packet, A. The other dataset, namely source domain dataset D^s, consists of network performance (L, B, and R) simulated by three simulators (NS-3 [15], Cooja [16], and TOSSIM [17]) under each of the 88 network configurations.

The primary objective of network configuration is to select the parameters (P, C, A) that allow the network to meet the performance requirements (L, B, R) specified by the application. Therefore, the network configuration problem can be formulated as a machine learning problem where the goal is to learns a nonlinear mapping $F_\theta(\cdot) : X \rightarrow y$, where $X = concatenation(L, B, R)$ denote the given network performance requirements and $y = concatenation(P, C, A)$ denote the configuration that allows the network to meet the network requirements. θ denotes the model parameters that are learned from training in a data-driven manner. Given that the network configuration values (y) can be discretized without losing generality, we further restrict F_θ as a discriminative model to solve a classification problem: an application can set its performance requirements (X), and the network configuration classification model (F_θ) will predict the network configuration (y) to satisfy the application requirements.

B. Limitations of Existing Solutions

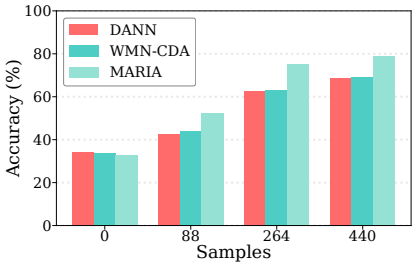

Fig. 1: Performance of DANN, MARIA, and WMN-CDA when trained on different numbers of samples from D^t.

Because creating D^t is very costly in real-world deployments [6], recent research has focused on how to train a good network configuration model F_θ with D^s and a minimum number of samples from D^t. In our empirical study, we first examine the performance of three state-of-the-art methods (DANN [6], WMN-CDA [9], and MARIA [8]) with different numbers of data samples from D^t during training. DANN leverages a teacher-student neural network to generalize D^s to D^t. WMN-CDA uses contrastive learning to produce a network configuration model. MARIA employs multi-source domain adaptation to narrow the simulation-to-reality gap.

Figure 1 shows the prediction performance of the network configuration models generated by DANN, WMN-CDA, and MARIA when using different numbers of data samples from D^t for training. As Figure 1 shows, without using any data from D^t for training, the prediction accuracies of the network configuration models generated by DANN, WMN-CDA, and MARIA are 34.2%, 33.6%, and 32.4%, respectively. The results show that none of the three methods can generate high-quality network configuration models without using data collected from the physical deployment, due to the simulation-to-reality gap. The accuracy increases as more samples from D^t are used for training. The accuracies reach 68.6% (DANN), 68.9% (WMN-CDA), and 78.5% (MARIA) when 440 samples from D^t are used to train the network configuration models. However, the data collection process can be very costly and labor-intensive in many industrial facilities, significantly compromising the benefit of using simulations to configure WMNs. Therefore, there is a critical need for new solutions that can produce high-quality network configuration models without collecting data from physical deployments.

C. Advantages of Feature Masking

We then study the cause of the simulation-to-reality gap in network configuration by quantifying the domain variance of different features. Table I lists the Kolmogorov-Smirnov Statistic (KS), the Wasserstein Distance (W-1), and the Standardized Mean Difference (SMD) of three features: L, B, R in X. KS

TABLE I: Domain variance on numerical features.

Feature	KS	W-1	SMD
L	0.482	113.746	0.863
B	0.017	0.015	0.000
R	0.081	0.025	0.176

TABLE II: Domain variance on textual representations.

Feature	KS	W-1	SMD
L Emb	0.208	0.066	0.386
B Emb	0.026	0.007	0.019
R Emb	0.065	0.023	0.095

and SMD are used to measure the distribution statistics in a normalized manner, highlighting which features show a critical gap in both the mean and the overall distribution. W-1 is used to measure the magnitude shift between the source domain D^s and the target domain D^t in addition to the distribution cost. Specifically, KS, W-1, and SMD are defined as:

$$\text{SMD}(D_f^t, D_f^s) = \frac{\mu_{D_f^t} - \mu_{D_f^s}}{\sqrt{\frac{1}{2}(\sigma_{D_f^t}^2 + \sigma_{D_f^s}^2)}} \tag{1}$$

$$\text{KS}(D_f^t, D_f^s) = \sup_{x \in \mathbb{R}} \left| F^t(x) - G^s(x) \right| \tag{2}$$

$$\text{W-1}(D_f^t, D_f^s) = \int_{-\infty}^{\infty} \left| F^t(x) - G^s(x) \right| \, dx \tag{3}$$

In Eq. 1 $\mu_{D_f^t}$ and $\sigma_{D_f^t}$ represent the mean and standard deviation of dataset D^t on feature f. Similarly, $\mu_{D_f^s}$ and $\sigma_{D_f^s}$ are the mean and standard deviation of dataset D^s under feature f. Referring to Eq. 2 and 3, F^t and G^s are the distribution functions of D^t and D^s under each feature f, and D_f^t, D_f^s are the feature vectors for the source and target domains, while sup is the supremum calculation.

As Table I lists, the feature L has the largest variance among all three metrics compared to the features B and R. For example, L's KS value is 0.482, while the KS values of B and R are 0.017 and 0.081, respectively. Similarly, the feature L has the largest W-1 (113.75) and SMD (0.863) values. The results show that the simulators struggle to portray the latency values accurately. All simulated readings tend to be smaller compared to those in D^t, as demonstrated by the large W-1 and SMD values. The results show that it is very challenging to train a good network configuration model without using D^t due to the large variance. The results also suggest that reducing feature variance is beneficial when training the network configuration model.

To understand the effect of masking, we measure the average Integrated Gradient Attribution (IGA) scores of input features in D^s when evaluated on D^t. IGA is a gradient-based interpretability method that quantifies the contribution of an input feature to a model's prediction by integrating gradients along a straight-line path from a baseline input to the actual input [18]. First, we serialize the input data into text using the template `"Latency: 178.07, Battery: 14.94, Reliability: 0.96"`, before calculating the IGA scores. Formally, given a prediction model $F_\theta(\cdot)$, an input embedding $\mathbf{e}$, and a baseline embedding $\mathbf{e}'$, the attribution of the i-th input dimension is computed as

$$\text{IGA}_i(\mathbf{e}) = (e_i - e_i') \int_0^1 \frac{\partial F(\mathbf{e}' + \alpha(\mathbf{e} - \mathbf{e}'))}{\partial e_i} \, d\alpha. \tag{4}$$

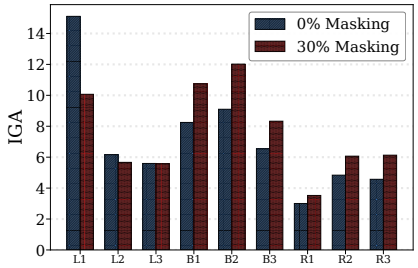

Fig. 2: IGA scores of different features with or without feature masking.

In our setting, IGA is computed at the token level over the serialized embeddings produced by a transformer model. For each data sample in D^t, token-level attributions are calculated with respect to the predicted configuration class, aggregated according to their corresponding serialized features, and then averaged across all samples in D^t to estimate the model's expected feature reliance. The blue bars in Figure 2 show the IGA scores for a model trained without feature masking (0%), while the red bars show the scores for a model trained with 30% random masking applied to domain-variant features.

As Figure 2 shows, without feature masking, tokens associated with feature L receive substantially higher attribution scores (15.109) than those associated with features B (8.248) and R (3.0), indicating that the model relies heavily on L's information learned from D^s. After applying 30% feature masking, the attribution magnitude of L-related tokens is significantly reduced (10.069), while the attributions of B (10.74) and R (3.525) remain relatively stable. This shift indicates that feature masking suppresses over-reliance on domain-variant features and guides the model to depend evenly on stable features.

D. Advantage of Textual Serialization

We further investigate the effectiveness of textual serialization in increasing prediction accuracy. We convert the numerical measurements on L, B, and R into strings using the technique described in II-C. We then add padding and split the text into its corresponding feature components, creating a sub-string for L, B, and R. Each sub-string is applied through the BERT embedding model [19], to extract 768 numerical semantic embeddings for L, B, and R each. Afterward, we recalculate the KS, W-1, and SMD values of the embeddings across the different feature sub-strings. Table II lists the KS, W-1, and SMD values on three features after the conversion. As Table II lists, both KS and SMD decrease after we convert the features from numerical values to strings. For example, the feature L in a numerical format has a KS value of 0.482 and an SMD value of 0.863, while the same feature yields a KS of 0.208 and an SMD of 0.386 after we apply the textual serialization and embedding. We attribute this finding to the embedding vector's greater dimensionality, which creates a more robust representation. The results demonstrate the effectiveness of using textual embeddings to reduce the

Algorithm 1: Domain-Variant Residual Analysis

Input: Dataset X with features $\{f_1 = L, f_2 = B, f_3 = R\}$
Pre-trained models M
Noise standard deviations N
Variant feature threshold τ
Output: Domain variant features set f_{dv}

1 Initialize $f_{dv} = \varnothing$;
2 **foreach** *feature* $f \in X.features$ **do**
3 Initialize $R_\mu = \varnothing$;
4 **foreach** *Standard deviation* $\sigma \in N$ **do**
5 Generate Gaussian noise vector $\epsilon_\sigma \sim \mathcal{N}(0, \sigma)$;
6 Perturb feature column $\tilde{X}[f] = X[f] + \epsilon_\sigma$;
7 Initialize $R_\forall = \varnothing$;
8 **foreach** *model* $m \in M$ *where* f *is a member of its input* **do**
9 Apply regression on noise $\tilde{f}'_m = m(\tilde{X})$;
10 Apply regression on clean $f'_m = m(X)$;
11 $R_\forall \leftarrow \Delta\mathcal{L} = |\mathcal{L}(\tilde{f}'_m, f_m) - \mathcal{L}(f'_m, f_m)|$;
12 $R_\mu \leftarrow \text{Avg}(R\forall)$;
13 $S_f = |\text{Slope}(R_\mu)|$;
14 **if** $S_f < \tau$ **then**
15 $f_{dv} \leftarrow f$;
16 **return** f_{dv};

discrepancy between D^s and D^t, highlighting the potential of using only D^s to produce high-quality network configuration models without physically collected data.

III. Design of LLM-OPT

LLM-OPT aims to use only simulation data to create good network configuration models. Figure 3 shows the overview of LLM-OPT. LLM-OPT first identifies features with large variance in the simulation dataset and then applies feature masking (see Section III-A) and text serialization (see Section III-B) to preprocess the data. Finally, it uses LLM fine-tuning to produce the network configuration model (see Section III-C).

A. Feature Masking

As discussed in Section II-C, the simulation-to-reality gap in WMN configuration largely depends on the variance of different features. Therefore, it is important to identify features with high variance (denoted as domain-variant features) and mitigate their negative effects during training. To achieve this goal, LLM-OPT first trains a prediction model that predicts each feature in the dataset from the others. It then add Gaussian noise to the features one at a time and evaluates the loss. If the loss is similar across different noise levels for a given feature, that feature is considered domain-variant. Otherwise, the feature is domain-invariant.

Algorithm 1 presents the detailed process. The inputs of Algorithm 1 include (1) the simulation data D^s with multiple features f_i (e.g., $f_1 = L$, $f_2 = B$, and $f_3 = R$), defined as X, (2) a set of pretrained linear regression models M that aims to predict each feature f_i from the remaining features, defined as $m(\cdot) : X[\sim f_m] \rightarrow X[f_m]$ for $m \in M$, (3) the standard deviation set N consisting of a range of user-specified standard deviations σ, and (4) a user-defined threshold τ. The Algorithm's output is the set of domain-variant features, denoted f_{dv}. Algorithm 1 first initializes f_{dv} to an empty set (Line 1) and then examines each feature in a loop iteration (Lines 2–15). Specifically, it first initializes the mean residual array for all models ($R\mu$), where f is a member of its input (Line 3) and then loops over each noise σ in N to construct R_μ using the mean residuals from each model not trained on f (Lines 4–11). To produce effective results, values in N must be evenly spaced to improve consistency when measuring the slope. Next, Algorithm 1 generates a Gaussian noise vector for column $X[f]$ by building a normal vector centered on mean 0 with standard deviation σ (ϵ_σ) (Line 5), and a higher σ disrupts the links between features, further degrading the predictive performance for domain-invariant features. Algorithm 1 then adds the noise vector ϵ_σ to the feature column $X[f]$ to generate a noisy dataset $\tilde{X}$, with only the f column modified (Line 6) and initializes the per-model residual array $R_\forall$ (Line 7). Lines 8–11 present the core sensitivity measurement. Each model $m \in M$ is trained on clean data ($\sigma = 0$) to capture potential inter-feature relationships. Algorithm 1 queries the model m with $\tilde{X}$ to obtain the predicted feature from that model $\tilde{f}'_m$, compared to the ground truth f_m, where $f_m \neq f$ (Line 9). It then obtains the expected prediction f'_m by giving the model X, the original training set without added noise (Line 10). Next, Algorithm 1 obtains the residual and stores it into $R_\forall$ (Line 11); the residual $\Delta\mathcal{L}$ is defined as the difference of the loss from the perturbed prediction, $\mathcal{L}(\tilde{f}'_m, f_m)$, and the loss from the expected prediction, $\mathcal{L}(f'_m, f_m)$. We use the Mean Squared Error (MSE) as the loss function $\mathcal{L}$, which is defined as:

$$\mathcal{L}(f'_m, f_m) = \frac{1}{n}\sum_{k=1}^{n}(f'_m - f_m)^2 \tag{5}$$

where n represents the total number of D^s samples. For features with correct inter-feature relationships, adding noise should increase the residuals. It takes the average residual among all the models where f is a member of its input for each σ, storing them in R_μ (Line 12). Finally, Algorithm 1 estimates the slope using a linear regression of the values in R_μ, yielding the absolute value of the slope (S_f), which measures the residual drift as noise is added to the data (Line 13). Because domain-variant features are less reactive to data changes, we expect that adding noise to the dataset will result in a smaller slope. Next, it checks for this change by analyzing whether S_f is below a user-defined threshold τ; if so, the feature is expected to be domain-variant, as changes in distribution do not affect the predictive quality of other features using it (Lines 14–15). τ is defined by the user, given how closely aligned the domains are, and through numerical analysis.

As demonstrated in II-C, masking domain-variant features reduces their impact during learning, reducing the gap. To achieve the same effect, LLM-OPT applies random masking

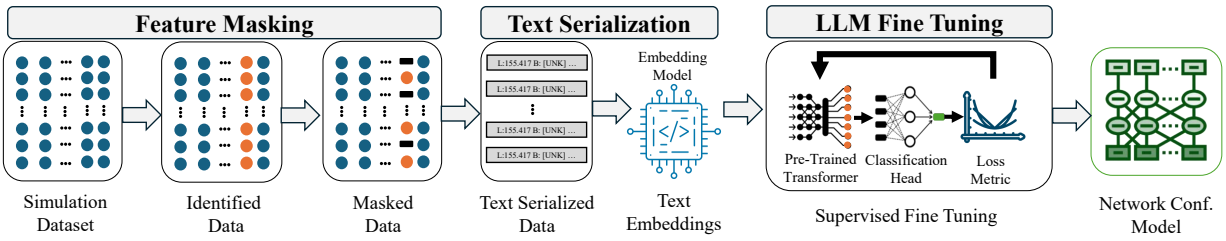

Fig. 3: Overview of LLM-OPT.

to all features in f_{dv}. Specifically, LLM-OPT randomly selects $\alpha * n$ samples from each of the data columns $X[f]$ and replaces them with `[UNK]` tokens during the following text serialization process, where n is the total number of samples and α is the masking ratio.

B. Text Serialization

As presented in Section II-D, it is beneficial to leverage textual embeddings to reduce the discrepancy between the simulation and physical data. Several text serialization methods have been developed in recent years for numerical classification using LLMs. For example, Hegselmann et al. [12] developed an automated serialization approach that converts tables to text using LLM-generated descriptions. However, existing methods are designed for highly variable and dynamic tasks, such as cross-task table analysis, whereas our problem requires greater consistency within a single situation. Dynamically generated templates are better suited to generalist models; in fine-tuning for a highly specific task, they introduce instability into the input. To address such an issue, we develop a template-based serialization method that uses a fixed template to ensure consistently generated text.

Our serialization technique uses a fixed, predefined prompt that is minimal in descriptive terms. Additional description in the fixed template results in the addition of universal tokens, increasing cost without any additional benefit. We define our template as $T =$ "Latency: {L}, Battery: {B}, Reliability: {R}", L, B and R are the respective features of X. We do not apply normalization to any feature, ensuring that the model captures the correct value ranges. We then replace the masked values with `[UNK]` tokens (see Section III-A).

C. LLM Fine-Tuning

As Section II-D highlights, textual embeddings through serialization help reduce distribution shifts in the datasets. However, to effectively leverage textual embeddings for the classification task defined in Section II-A, we need to train a model that can analyze the textual data. To this end, we use pre-trained LLMs and apply a fine-tuning pipeline to align the classification task objective to the model pre-training. Pre-trained LLMs are powerful models trained on large amounts of textual data and can effectively generalize to the embedding space, while fine-tuning helps align the model to specific objectives. Given the large output space of the data (88 labels), LLM-OPT employs a custom classification head, which is incompatible with the commonly used sequence-to-sequence models. LLM-OPT is built using the BERT pretrained model using an encoder architecture, but our solution is designed to be compatible with a range of model types, including encoder-only, decoder-only, and permutation language models (PLMs). Section IV-D shows LLM-OPT's performance when working with different models.

LLM-OPT employs LLM fine-tuning during training. It first applies an embedding process that uses a model to convert serialized text into numerical semantic representations. It then fine-tunes the BERT model to leverage its textual processing capabilities for the classification task. The process includes four steps. First, it applies a modified textual input via an encoding and embedding model to obtain semantic embeddings that capture relationships across all features. It then applies forward inference of the embedding on the BERT model. BERT consists of layers of encoder blocks, each containing a self-attention step followed by a linear layer. The BERT encoders' final output layer returns the logit values, which represent the raw predicted values and can be converted to probabilities. The logits are then fed to a softmax head during the second step to obtain the probabilities for each of 88 configuration classes as detailed in equation 6:

$$p(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{6}$$

where K is the length of the input vector z, and i and j are indexes in z. Lastly, it back propagates on the training loss, $\mathcal{L}_{Tr}$ (Categorical Cross Entropy) to carry out the multi-class classification problem as shown in Equation 7:

$$\mathcal{L}_{Tr} = -\frac{1}{\beta} \sum_{j=1}^{\beta} \sum_{i=1}^{\mathcal{K}} y_{i,j} \cdot log(\hat{y}_{i,j}) \tag{7}$$

where β is the total number of samples in the batch, $\mathcal{K}$ is the total number of classes available, $\hat{y}$ is the probability array from the output of the classification head, and y is a one-hot encoded array with a probability of 1 for the correct label, and 0 for everything else. Applying this formulation enables LLM-OPT to fine-tune the LLM by updating the weights based on embeddings derived from the original data.

IV. Evaluation

We conduct a series of experiments to evaluate LLM-OPT's ability to produce high-quality network configuration models.

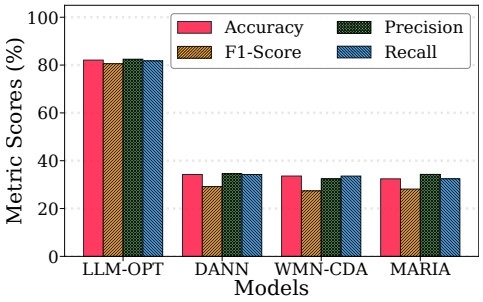

Fig. 4: Performance of LLM-OPT and three baselines without using data from physical deployments for training.

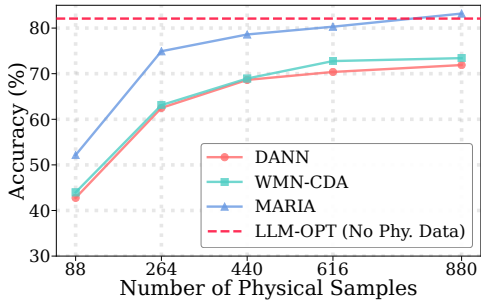

Fig. 5: Prediction accuracy of our baselines when using different numbers of samples collected from the physical deployment for training. The red dashed line denotes the 82.1% prediction accuracy achieved by LLM-OPT without using data collected during physical deployment.

In this section, we first present the implementation of LLM-OPT and our three non-LLM baselines. We then compare LLM-OPT's performance to the baselines. We repeat our experiments with different masking percentages and examine their effects on LLM-OPT's performance. We then conduct an ablation study to investigate the performance contribution of each component of LLM-OPT. Finally, we study the effect of different model architectures on LLM-OPT's performance.

A. Implementation

We implement LLM-OPT's domain-variant residual analysis algorithm (Algorithm 1) in Python using the Scikit-Learn library, empirically set τ to 0.1, the masking percentage to 30%, and the default seed to 42. Both the masking ratio and the τ are empirically selected from experimental runs. For τ, we identify that even minute changes are enough to identify domain invariant features from our dataset quickly. Similarly, the masking ratio is fairly consistent after 20%, and we find a global maximum at 30%. We use the PyTorch and HuggingFace Transformers libraries to implement LLM-OPT's fine-tuning pipeline, set the learning rate to $2e^{-5}$, and use a BERT (Encoder) as the base model to train the network configuration model. Furthermore, a linear scheduling algorithm with 0.01 weight decay is used over 30 epochs with a batch size of 32 to control the training process of LLM-OPT.

We adopt the implementations of WMN-CDA [9], MARIA [8], and DANN [6] provided by their authors and use their default settings. If the baselines have noise added to them or require less data from D^t, they are effectively reduced via dataset-level abstraction. We leverage two publicly accessible datasets created by Shi et al. [6], [14] for our experiments. All experiments are performed on a server with 2 NVIDIA RTX 4090 cards, totaling 48GB of VRAM and 256 GB of RAM, using CUDA for acceleration.

B. Performance of LLM-OPT

We first evaluate the prediction performance of the network configuration model produced by LLM-OPT and compare it with that generated by our baselines. As the red bars in Figure 4 show, the network configuration model trained by LLM-OPT without using data from physical deployments achieves an 82.1% prediction accuracy. In contrast, the accuracies of the models generated by DANN, WMN-CDA, and MARIA are 34.2%, 33.6%, and 32.4%, respectively. Similarly, the F1 Score of the network configuration model produced by LLM-OPT is 82.0%, significantly higher than the ones under DANN (34.2%), WMN-CDA (33.6%), and MARIA (32.4%). We observe similar trends in terms of precision and recall. The results show that LLM-OPT is the only solution that can generate good network configuration models without using data from physical deployments. At the same time, all baselines fail due to the simulation-to-reality gap.

Figure 5 shows the prediction accuracy of our baselines when using different numbers of data samples collected from the physical deployment. As Figure 5 shows, the prediction accuracies of all baselines improve when increasing numbers of data samples collected from the physical deployment are used for training. For example, the accuracy of the network configuration model generated by MARIA increases from 32.4% to 52.2% as the number of physical data samples increases from 0 to 88. The accuracy further rises to 63.1% when 264 samples are used for training. MARIA requires 880 samples collected from the physical deployment to match the performance of the LLM-OPT model without using any physical data. Compared to LLM-OPT and MARIA, DANN and WMN-CDA achieve inferior performance even when trained on 880 physical data samples. Given that collecting data from physical deployments can be costly and labor-intensive in many industrial facilities, the results highlight LLM-OPT's ability to produce good network configuration models using only simulation data.

C. Effect of Feature Masking

To study the effect of feature masking on LLM-OPT's performance, we mask varying percentages of the data samples in f_{dv} and repeat the experiments. Figure 6 shows the prediction accuracies of LLM-OPT and our baselines when different percentages of data in f_{dv} are masked. As shown in the figure, the prediction accuracy of LLM-OPT increases from 63.3% to 80.3% when we mask 10% of the data. The accuracy increases to 81.7% and then to 82.1% when 20% and 30% of the data are masked, respectively. No further increase in accuracy is observed as more data is masked. The results

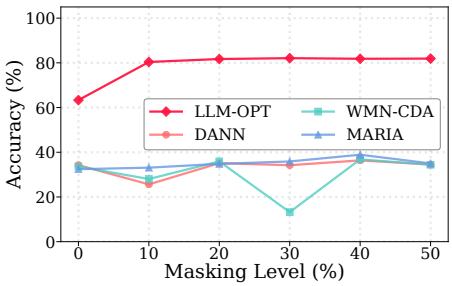

Fig. 6: Prediction accuracy of LLM-OPT and baselines when different percentages of data are masked.

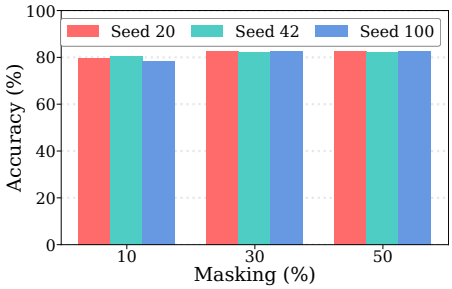

Fig. 7: Prediction accuracy when using three different seeds to select data for masking.

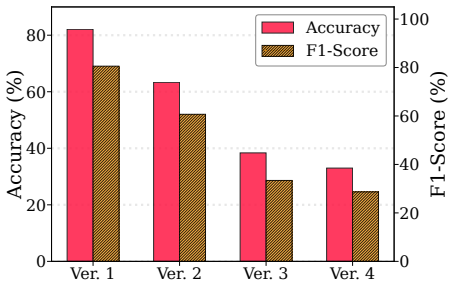

Fig. 8: Study on the effect of feature masking and text serialization on LLM-OPT.

show the robustness of LLM-OPT: masking 10%-50% of the data in f_{dv} consistently yields good performance. On the other hand, the masking percentage does not significantly affect the performance of our baselines. For example, MARIA achieves accuracies from 32.4% to 38.9% across different masking levels.

We further examine the stability of LLM-OPT by using different random seeds to determine which samples in f_{dv} to mask. Figure 7 displays the prediction accuracy using three different seeds (20, 42, and 100). As the figure demonstrates, the differences in prediction accuracy are negligible when different seeds are used for masking. For example, the accuracy ranges from 78.1% to 80.4% when 10% of the data in f_{dv} is masked. Similarly, the accuracy ranges from 82.1% to 82.6% when 30% of the data in f_{dv} is masked. The results demonstrate the strong stability of LLM-OPT across different random masking strategies.

D. Ablation Study

We also perform an ablation study to investigate the contribution of each of LLM-OPT's techniques (masking and text serialization) to its overall performance. To carry out this study, we implement four versions of LLM-OPT: (Ver. 1) the full version with both feature masking and text serialization, (Ver. 2) the version with text serialization but feature masking disabled, (Ver. 3) the version with feature masking but text serialization disabled[1], and (Ver. 4) the version with neither BERT text pre-training nor feature masking. Figure 8 shows the prediction accuracy and the F1-score of our four versions of LLM-OPT. As Figure 8 shows, the accuracy decreases from 82.1% to 63.3% when we disable feature masking in LLM-OPT. After disabling text serialization, the accuracy drops to 38.4%. The accuracy is 33.0% when we disable both feature masking and text serialization. The results show that feature masking and text serialization play an important role in LLM-OPT, and their synergy promotes high prediction accuracy.

E. Effect of Different Model Architectures

LLM-OPT is designed to be compatible with different model architectures, which we evaluate in this study. Table III lists the prediction accuracy and F1-score of LLM-OPT with BERT (encoder-only), TinyLLAMA (decoder-only) [20], and the XLNET (PLM) [21]. As shown in the table, accuracies range from 63% to 82% (BERT), 67% to 80% (TinyLlama), and 55% to 80% (XLNet) across different levels of data masking. The results show consistency across three models, with the best performance observed with the BERT encoder architecture. BERT demonstrates the best performance thanks to its excellent training architecture and bidirectional nature for text analytics. More importantly, the results demonstrate the robustness of LLM-OPT across different model architectures.

[1]To disable the text serialization, we replace the BERT text pre-training in LLM-OPT with the transformer model trained with numerical data.

TABLE III: Performance of LLM-OPT with different model architectures.

Models	Metrics	0%	10%	20%	30%	40%	50%
BERT	Accuracy	0.63	0.80	0.82	0.82	0.82	0.82
	F1-Score	0.61	0.78	0.80	0.81	0.80	0.80
TinyLLAMA	Accuracy	0.67	0.75	0.78	0.78	0.81	0.80
	F1-Score	0.66	0.73	0.77	0.76	0.80	0.79
XLNET	Accuracy	0.55	0.72	0.79	0.80	0.79	0.80
	F1-Score	0.51	0.71	0.77	0.79	0.77	0.78

V. Related Works

In recent years, there has been increasing interest in using simulations to configure WMNs because they can be done more quickly, with much less overhead, and allow for testing different configurations under controlled, identical conditions. For instance, Shi et al. developed a framework that leverages a wireless control simulator to configure industrial WMNs based on the application's Quality of Service (QoS) demands [22]. Liu et al. integrated the process control system model into a unified discrete-event simulator and used it to identify good configurations for industrial WMNs and process control systems [23]. However, recent studies have shown that the network configuration chosen in simulations may not perform well in real-world deployment due to the simulation-to-reality gap in network configuration [6]. Such observations motivate recent research efforts to develop new solutions to close the simulation-to-reality gap [6], [7], [8], [9]. For example, Shi et al. developed a method that leverages a teacher-student neural network to transfer network configuration knowledge learned from simulations to real-world WMN deployments [6]. Ma

et al. proposed to use contrastive learning in WMN configurations [9]. Cheng et al. used the simulation data generated by multiple simulators and employed multi-source domain adaptation to close the simulation-to-reality gap. Unfortunately, all existing solutions require collecting data from the physical deployment to narrow the gap and ensure effective transfer learning. This data collection process can be costly and labor-intensive in many industrial facilities [6], significantly compromising the benefit of using simulations to configure WMNs. In contrast to previous work, this paper presents the first solution that produces high-quality network configuration models for WMNs without requiring data from physical deployments.

Recently, significant efforts have been made to leverage LLMs in wireless networking. For example, Shao et al [24] and Qui et al. [25] proposed to use LLMs to assist in the design of new wireless networks. Tan et al. developed a framework that employs LLMs and reinforcement learning to configure medium access control (MAC) protocols based on runtime conditions [26]. Ronando et al. introduced a few-shot optimization method that uses LLMs to select sensor data for fatigue detection [27]. Golam et al. proposed using LLMs to assist in the design of digital twins for industrial WMNs, thereby improving security and operability [28]. Du et al. designed a distributed automation firmware using LLMs as machine-to-machine agents for communication and optimization of the wireless system [29]. In contrast to existing work, this paper explores the efficiency of using LLMs to configure WMNs only with simulation data.

VI. Conclusions

In this paper, we present LLM-OPT, the first solution that produces high-quality network configuration models without requiring data collected from the physical deployment. Unlike existing methods that rely on data collected from physical deployments to narrow the simulation-to-reality gap in WMN configuration, LLM-OPT first employs feature masking and textual serialization techniques to preprocess simulation data, then leverages LLM fine-tuning to generate network configuration and models. Experiments show that LLM-OPT outperforms existing solutions, achieving 82% prediction accuracy.

Acknowledgment

This work was supported in part by the National Science Foundation under grants CNS-2150010, ECCS-2242700, and IIS-2529283. This work was also supported in part by the Department of Energy (DOE) under grants DE-NA0004016 and DE-CR0000046.

References

[1] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems," *Proceedings of the IEEE, Special Issue on Industrial Cyber Physical Systems*, vol. 104, no. 5, pp. 1013–1024, 2016.

[2] "FieldComm Group," FieldComm Group. [Online]. Available: https://www.fieldcommgroup.org/

[3] "International Society of Automation (ISA)," ISA. [Online]. Available: https://www.isa.org/

[4] "Connectivity Standards Alliance," Connectivity Standards Alliance. [Online]. Available: https://csa-iot.org/

[5] "Industrial Wireless Technology," Emerson. [Online]. Available: https://www.emerson.com/en-us/automation/measurement-instrumentation/industrial-wireless-technology

[6] J. Shi, M. Sha, and X. Peng, "Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation," in *NSDI*, 2021.

[7] X. Cheng and M. Sha, "Configuring industrial wireless mesh networks via multi-source domain adaptation," in *IWQoS*, 2023.

[8] X. Cheng, M. Sha, and D. Chen, "Configuring Industrial Wireless Mesh Networks via Multi-Source Domain Adaptation," in *EWSN*, 2024.

[9] A. Ma and M. Sha, "WMN-CDA: Contrastive Domain Adaptation for Wireless Mesh Network Configuration," in *SAC*, 2025.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *NeurIPS*, 2020.

[11] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," in *NeurIPS*, 2022.

[12] S. Hegselmann, A. Buendía, H. Lang, M. Agrawal, X. Jiang, and D. Sontag, "TabLLM: Few-Shot Classification of Tabular Data with Large Language Models," *AISTATS*, 2023.

[13] X. Wen, H. Zhang, S. Zheng, W. Xu, and J. Bian, "From Supervised to Generative: A Novel Paradigm for Tabular Deep Learning with Large Language Models," in *KDD*, 2024.

[14] J. Shi, A. Ma, X. Cheng, M. Sha, and X. Peng, "WirelessHART Dataset," 2021. [Online]. Available: https://github.com/aitianma/WSNConfDomainAdaptation

[15] N.-. Consortium, "NS-3 Network Simulator," https://www.nsnam.org/.

[16] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level Sensor Network Simulation with Cooja," in *LCN*, 2006.

[17] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *SenSys*, 2003.

[18] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International Conference on Machine Learning (ICML)*, 2017.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *NAACL*, 2019.

[20] P. Zhang, G. Zeng, T. Wang, and W. Lu, "TinyLlama: An Open-Source Small Language Model," *arXiv preprint arXiv:2401.02385*, 2024.

[21] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *NeurIPS*, 2019.

[22] J. Shi and M. Sha, "Parameter Self-Configuration and Self-Adaptation in Industrial Wireless Sensor-Actuator Networks," in *INFOCOM*, 2019.

[23] Y. Liu, R. Candell, K. Lee, and N. Moayeri, "A Simulation Framework for Industrial Wireless Networks and Process Control Systems," in *WFCS*, 2016.

[24] J. Shao, J. Tong, Q. Wu, W. Guo, Z. Li, Z. Lin, and J. Zhang, "Wireless-llm: Empowering large language models towards wireless intelligence," *Journal of Communications and Information Networks*, vol. 9, no. 2, pp. 99–112, 2024.

[25] K. Qiu, S. Bakirtzis, I. Wassell, H. Song, J. Zhang, and K. Wang, "Large Language Model-Based Wireless Network Design," *IEEE Wireless Communications Letters*, vol. 13, no. 12, pp. 3340–3344, 2024.

[26] R. Tan, R. Li, and Z. Zhao, "LLM4MAC: An LLM-Driven Reinforcement Learning Framework for MAC Protocol Emergence," in *SPAWC*, 2025.

[27] E. Ronando and S. Inoue, "Few-Shot Optimization for Sensor Data Using Large Language Models: A Case Study on Fatigue Detection," *Sensors*, vol. 25, no. 11, p. 3324, 2025.

[28] M. Golam, M. M. Alam, M. R. Subhan, D.-S. Kim, and J.-M. Lee, "BLIND-TWIN: Blockchain-Assisted LLM-Based Cds for Digital Twin-Enhanced Industrial AIoT," in *ICC*, 2025.

[29] Y. Du, Q. Yang, L. Wang, J. Lin, H. Cui, and S. C. Liew, "LLMind 2.0: Distributed IoT Automation with Natural Language M2M Communication and Lightweight LLM Agents," *arXiv preprint arXiv:2508.13920*, 2025.