

MPM-LLM4DSE: Reaching the Pareto Frontier in HLS with Multimodal Learning and LLM-Driven Exploration

Lei Xu, Shanshan Wang, Chenglong Xiao*

Department of Computer Science, Shantou University, China

{24lxu,sswang,chl Xiao}@stu.edu.cn

Abstract—High-Level Synthesis (HLS) design space exploration (DSE) seeks Pareto-optimal designs within expansive pragma configuration spaces. To accelerate HLS DSE, graph neural networks (GNNs) are commonly employed as surrogates for HLS tools to predict quality of results (QoR) metrics, while multi-objective optimization algorithms expedite the exploration. However, GNN-based prediction methods may not fully capture the rich semantic features inherent in behavioral descriptions, and conventional multi-objective optimization algorithms often do not explicitly account for the domain-specific knowledge regarding how pragma directives influence QoR. To address these limitations, this paper proposes the MPM-LLM4DSE framework, which incorporates a multimodal prediction model (MPM) that simultaneously fuses features from behavioral descriptions and control and data flow graphs. Furthermore, the framework employs a large language model (LLM) as an optimizer, accompanied by a tailored prompt engineering methodology. This methodology incorporates pragma impact analysis on QoR to guide the LLM in generating high-quality configurations (LLM4DSE). Experimental results demonstrate that our multimodal predictive model significantly outperforms state-of-the-art work ProgSG by up to $10.25\times$. Furthermore, in DSE tasks, the proposed LLM4DSE achieves an average performance gain of 39.90% over prior methods, validating the effectiveness of our prompting methodology. Code and models are available at <https://github.com/wslcccc/MPM-LLM4DSE>.

Index Terms—Graph neural network, high-level synthesis, design space exploration, multimodal features, language models

I. INTRODUCTION

High-Level Synthesis (HLS) has emerged as a paradigm-shifting methodology in modern VLSI design. This approach empowers designers to utilize algorithmic specifications in high-level languages (e.g., C/C++) for hardware generation, while providing tunable synthesis parameters to optimize implementation quality. However, the substantial time required by HLS tools to generate quality-of-results (QoR) metrics remains a critical bottleneck particularly during hardware optimization where design spaces are often combinatorially vast. Within design space exploration (DSE), most algorithms struggle with both aspects due to their inability to interpret the impact of synthesis pragmas on final implementations. Thus, the pivotal objectives for HLS DSE are to rapid and accurate QoR prediction and establish an intelligent DSE paradigm.

As illustrated in Fig. 1, the HLS DSE flow begins with directive-annotated C/C++ behavioral descriptions, which undergo high-level synthesis to generate RTL implementations and extract corresponding QoR metrics. A multi-objective

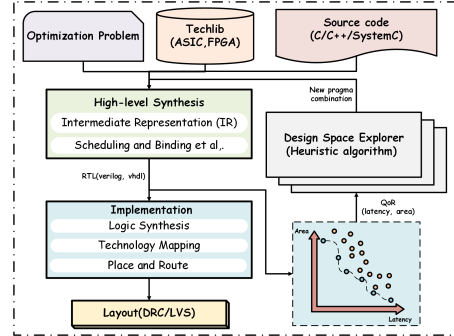


Fig. 1: An overview of HLS DSE.

optimization engine then explores new directive combinations, iteratively refining the design space to converge on Pareto-optimal configurations that balance QoR metrics. Notably, deep learning techniques are employed to accelerate this process by rapidly predicting QoR metrics, thus reducing the need for full synthesis at each iteration.

QoR prediction. The evolution of QoR prediction methodologies demonstrates a clear trajectory: starting from foundational graph neural network (GNN)-based feature extraction on control and data flow graphs (CDFGs), it has progressed toward increasingly sophisticated architectures and paradigms. Early works establish GNNs as viable surrogates for HLS QoR estimation (Wu et al. [1]), with subsequent innovations enhancing topological awareness through edge-centric aggregation (Lin et al. [2]) and hierarchical pooling (Kuang et al. [3]). Breakthrough methodologies then emerged that bypassed traditional EDA dependencies: Sohrabizadeh et al. achieved millisecond-level evaluation via surrogate modeling [4], Ferretti et al. matched industrial simulator accuracy without compiler internals [5], and Gao et al. pioneered direct C/C++ source processing with multi-granular embeddings [6]. Most recently, Qin et al. proposed a representation learning method combining the source sequence and the CDFG (ProgSG [7]).

DSE methods. Prior DSE methodologies have primarily built upon evolutionary algorithms and machine learning. Wang et al. utilized machine learning to optimize metaheuristic parameters, demonstrating significant performance gains over hand-tuned approaches [8]. Wu et al. established a reinforcement learning driven framework for Pareto-optimal resource allocation across competing objectives [9], while Yao et al. decomposed DSE problems via MOEA/D and EDA probabilistic modeling to minimize synthesis runs [10]. Xu et al. introduced the LLMMH framework, which integrates large language models (LLMs) as solution operators within metaheuristics to enhance DSE precision [11].

*Corresponding author: Chenglong Xiao (chl Xiao@stu.edu.cn). This work is partially sponsored by Guangdong Basic and Applied Basic Research Foundation (2022A1515110712, 2025A1515010272 and 2023A1515010077, the Scientific Research Project of Colleges and Universities in Guangdong Province of China under Grant No. 2021ZDZX1027).

While GNN-based QoR prediction has shown promise, it still has several limitations. CFGs only provide control and data flow information to GNNs and do not explicitly capture structural semantics or programmer annotations, which are critical for prediction accuracy. For instance, with directives such as `#pragma HLS UNROLL factor=4`, the CFG contains UNROLL nodes connected to corresponding code blocks. However, graph feature extractors struggle to capture both the precise semantic meaning and operational scope of UNROLL directives. While UNROLL node insertion supplements abstract CFG graph information, it fails to accurately represent the implicit intent behind pragma usage in the source code. Although ProgSG [7] introduced token-node alignment to mitigate this issue, its monolithic transformer architecture may not fully capture complex source code features and lacks robust mechanisms for fusing graph and text representations. For example, ProgSG’s method merely converts the UNROLL directive into a node and aligns it with corresponding tokens, failing to capture how the directive influences code statement blocks. Furthermore, while DSE algorithms aim to iteratively improve solutions, many existing methods often do not systematically account for the significant influence of pragma directives on final QoR metrics such as latency and resource utilization. To address the aforementioned challenges, we propose MPM-LLMDSE, an automated framework featuring:

- 1) **A Multimodal Dataset:** Constructs Graph-Text dataset combining CFG structural features with semantic embeddings extracted from pragma-augmented source code using a pre-trained language model (LM).
- 2) **A Multimodal Prediction Model (MPM):** Proposes a hybrid architecture leveraging LMs and GNNs to extract complementary features from source code and CFG respectively, dynamically fused via multi-head attention mechanisms.
- 3) **An LLM-Driven DSE Engine (LLM4DSE):** Introduces LLM as an optimizer for DSE, and a sophisticated prompt engineering methodology (PEODSE) that incorporates pragma impact analysis and high-quality examples to guide LLMs in generating high-quality design configurations.

II. PRELIMINARIES

A. Multimodal Graph Learning in HLS QoR Prediction

Existing GNN-based HLS QoR prediction methods, which rely on CFGs generated by compilation tools [12], [13], primarily utilize structural flow information but do not fully capture the rich semantic features available in source code. This issue motivates our investigation into more effective integration of source-level semantics for improved prediction accuracy. Tang et al. [14] demonstrate LLM embeddings’ intrinsic regression capability with MLP heads, while Joshi et al. [15] conceptualize Transformers as message-passing GNNs. Transformer-based LMs (e.g., BERT [16]) can leverage their inherent multi-head attention mechanisms to learn semantic and syntactic features from text [17]. Building upon these theoretical foundations, we posit that fusing graph representations and textual representations yields embeddings incorporating both semantic information

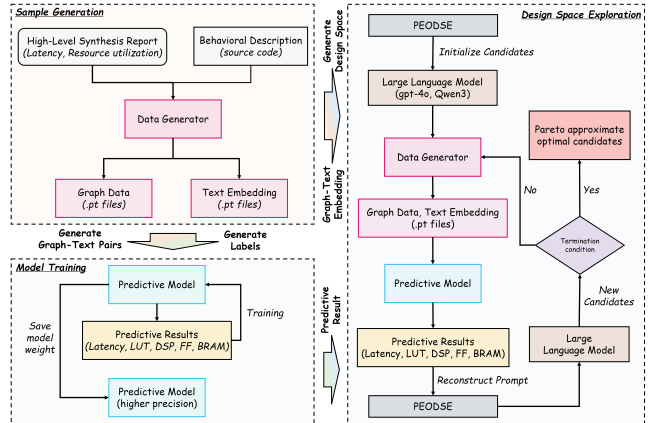


Fig. 2: The framework of MPM-LLM4DSE.

and graph structural information, thus leading to more accurate QoR prediction. However, graph representations and textual representations reside in distinct representation spaces, making their effective integration the critical challenge. Based on the preceding analysis, we conceptualize multimodal graph learning for QoR prediction as follows:

$$Target_i = MLP_i(Fuse(h_G, h_S)) \quad (1)$$

where i denotes the index of the QoR prediction objective, h_G and h_S represent the global structural representations extracted by GNNs and global semantic representations derived from LMs respectively, and $Fuse(\cdot)$ denotes the parametric fusion mechanism integrating h_G and h_S to yield joint representations.

B. Multimodal Dataset in HLS QoR Prediction

Traditional single-CFG datasets are incompatible with our proposed predictive model, thus we introduce a novel multimodal dataset specifically designed for QoR prediction. Given a CFG \mathcal{G} , its design configuration d_c , and behavioral description b_d , we first merge the configuration d_c and description b_d into composite text inputs. After tokenization, these inputs are passed through the LM. h_S is derived by averaging the summed hidden states of the CLS token from the final layer of the LM (CLS token inherently represents the global semantic representation of the input text). The process is formulated as:

$$h_S = \frac{1}{l} \sum_{k=0}^l CLS(LM_{hidden_k}(tokenizer(merge(d_c, b_d)))) \quad (2)$$

where $hidden_k$ is the output of the last hidden state of k th layers of the LM, $merge(\cdot)$ is used to merge behavioral descriptions and design configurations, $tokenizer(\cdot)$ is used to convert text data into tokens and $CLS(\cdot)$ gets the CLS token features from the hidden states of a LM. Combining the obtained feature representations h_S with corresponding CFG \mathcal{G} yields individual trainable data instances.

III. METHODOLOGY

Fig. 2 shows the MPM-LLM4DSE framework that integrates three cohesive modules: Sample Generation employs compilation tools and GNNs to extract graph representations from source codes, while language model is used to extract

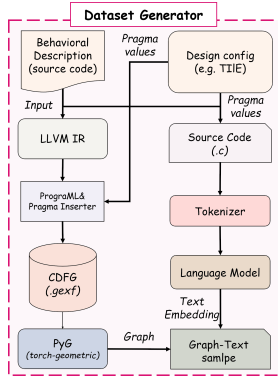


Fig. 3: The workflow of Dataset Generator.

TABLE I: Node features of graph data.

Features	Description	Values type
<i>node type</i>	index of node type	long
<i>instruction type</i>	index of instruction	long
<i>function type</i>	index of function	long
<i>block type</i>	index of block	long
<i>Latency</i>	amount of clock cycles in the node	int
<i>LUT</i>	amount of LUT used in the node	int
<i>DSP</i>	amount of DSP in the node	int
<i>FF</i>	amount of FF in the node	int

textual representations, thereby forming multimodal embedding representations. These representations are then combined with QoR metrics extracted from HLS reports to create the training samples. Model Training initializes model weights and optimizes them through training on generated multimodal datasets, persistently storing weights that minimize prediction error. Design Space Exploration orchestrates an iterative loop where task-specific prompts guide LLMs to generate design configurations until the given termination condition is met (e.g., convergence thresholds or iteration limits). Each iteration involves the Data Generator synthesizing these with source code into graph-text embedding representations, and the predictive model evaluating corresponding QoR metrics.

A. Data Generator

Fig. 3 outlines the processing flow of the Data Generator. The source code is first processed by LLVM to produce an intermediate representation (IR), which is then converted into a CDFG via ProGraML. Concurrently, in accordance with the GNN-DSE [4], pragma inserter constructs icmp nodes to enrich the CDFG with pragma-related information. On the other hand, as discussed in Equation 2, we obtain complete source code by integrating pragmas.

While LLMs rapidly evolve with parameter counts reaching hundreds of billions, most of LMs contain merely millions of parameters. However, for specialized downstream tasks, LMs trained on domain-specific corpora have been shown to outperform LLMs. Furthermore, owing to their faster deployment and reduced computational demands, such LMs represent a highly suitable choice for QoR prediction applications. In this work, we choose the pre-trained CodeBERT-c [18] as our language model because the behavioral descriptions are typically written in C or C++. CodeBERT-c is specifically trained on C code,

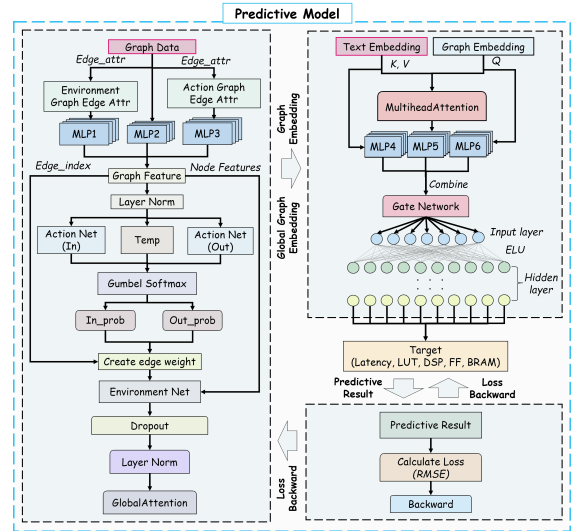


Fig. 4: The architecture of the proposed predictive model.

exhibiting stronger comprehension of the C language. Therefore, selecting CodeBERT-c enables better feature extraction from the source code. Using a tokenizer, we convert the source code into token representations, and employ CodeBERT to extract and save features from these token representations. By integrating CDFGs and text embeddings, we obtain trainable Graph-Text representations. The Graph-Text representations along with the QoR metrics in HLS reports or predictive results are used to form the training samples. Additionally, the node features of CDFGs are presented in Table I. Categorical features such as node type and instruction type are represented using one-hot encoding, while preprocessed numerical features directly utilized.

B. Multimodal Predictive Model

Fig. 4 illustrates the architecture of the proposed multimodal predictive model, which consists of three core components: graph feature extraction using GNNs, fusion of graph features with text embeddings, and the end-to-end training of the predictive model. Firstly, prior to the GNN processing, the Graph Data undergoes preliminary transformation in which the original edge feature matrix is mapped into two distinct edge feature matrices. These, along with the original graph structure, collectively form the enhanced Graph Feature representation used in subsequent stages.

Secondly, we design a Enhanced-CoGNN (ECoGNN) variant for QoR prediction inspired by the CoGNN [19] framework. Since the baseline CoGNN only processes undirected graph information flow, we extended its functionality. We redefine the set of node states Φ as follows:

$$\Phi = \{S, L_{in}, L_{out}, B, I\} \quad (3)$$

where S , B , and I denote node states for information aggregation: S receives neighbor information while broadcasting its own state, B exclusively broadcasts its own state without receiving, I neither broadcasts nor receives information; L_{in} receives exclusively from neighbors connected via incoming

edges, and L_{out} receives exclusively from neighbors connected via outgoing edges.

At the k -th ECoGNN layer, we input node features h_{k-1} into LayerNorm [20] to obtain the regularized feature h'_{k-1} . Since node actions follow a categorical distribution rendering the training process non-differentiable. We employ the Gumbel-Softmax estimator [21] to achieve differentiable training. According to comparative experiments in [11], Environment Net and Action Net are implemented using MEANGUNs [22] and SUMGNNs [22] respectively, with Temp (learnable Gumbel softmax temperature) primarily realized via MLPs. The weight of the edges $edge_weights$ is constructed for outgoing and incoming edges based on the index of the edges $edge_indexes$ and the probability of information inflow and outflow $prob_{in}$, $prob_{out}$, integrated with the \mathcal{G}'_f and fed into the Environment Net to yield h_i , which is then regularized via LayerNorm. By iterating the above process until node features are output from the final ECoGNN layer, we obtain final node-level representations h_f . To derive graph-level embeddings h_G , these would directly feed into an MLP for QoR prediction. However, using average pooling causes significant information loss. To address this, we leverage global node attention [23] to generate the final graph-level embedding h_G . With both textual embeddings from source code and graph-level embeddings available, we prioritize multimodal feature fusion. Direct summation introduces excessive feature noise. In image captioning, Sun et al. employed multi-head attention to align image regions with tokens [24]. As defined for QoR-oriented multimodal graph learning, our fusion aims to supplement semantic representations for nodes in graph data and augment structural information for subregions. Multi-head attention [25] effectively fulfills this objective by fusing graph-level and textual embeddings.

Then we propose using the graph embedding h_G as the query matrix Q and the text embedding as the key matrix K and value matrix V . Multi-head attention computation is performed and the results are aggregated to obtain fused features h_{fuse} . The process is formulated as:

$$head_w = Attention(QW_w^Q, KW_w^K, VW_w^V) \quad (4)$$

$$h_{fuse} = Concat(head_1, \dots, head_w)W^O \quad (5)$$

where W_w^Q , W_w^K and W_w^V are the learnable weight matrices of Q , K and V respectively, W^O is used to fuse the outputs of multiple heads. MLPs first align the dimensions of the graph embedding h_G and the fused features h_{fuse} . These aligned features are then fed into a gated network [23] to dynamically control the information flow between them.

Finally, the dynamically fused feature h'_{fuse} is fed into an MLP prediction head to yield predicted results. The root mean squared error (RMSE) loss is computed to update model parameters. While mean absolute percentage error (MAPE) also reflects prediction accuracy, for targets like latency with values exceeding hundreds of thousands, a 1% error translates to thousands of clock cycles. Thus, RMSE more accurately quantifies prediction error.

Fig. 5 depicts the process of multimodal feature fusion for QoR prediction. The left portion of Fig. 5 illustrates ECoGNN's

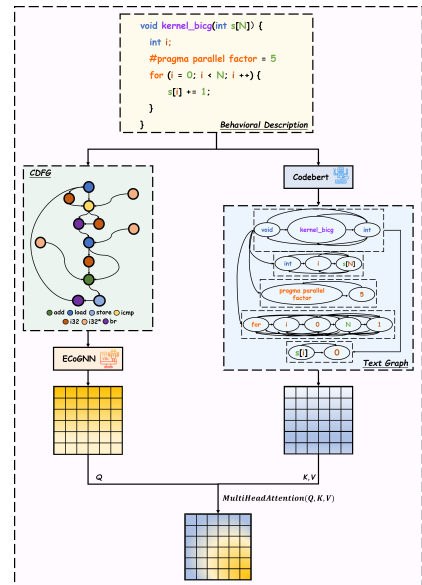


Fig. 5: Multimodal feature fusion for QoR prediction.

feature extraction process from CDFGs, with the resulting features forming the Query matrix. The right section demonstrates LM's feature extraction from source code, where the text graph [15] clarifies that this process inherently operates as a message-passing procedure analogous to GNNs. The obtained text embeddings serve as the Key and Value matrices, which are fused with the Query matrix through a multi-head attention mechanism to yield feature representations incorporating both semantic information and graph structural information.

C. Design Space Exploration

To fully harness the semantic capabilities of LLMs for intelligent design space exploration, we propose the LLM4DSE methodology. This approach consists of three core steps: leveraging LLMs as optimizers to initialize or generate new solutions, evaluating these solutions, and updating both the optimal solutions and prompts accordingly. As shown in Fig. 2, the iterative process commences with a task-specific prompting method PEODSE guiding LLMs to initialize solutions. These solutions are then processed by the Data Generator to produce graph and text embeddings, which are input into the Predictive Model for evaluation. The results of this evaluation update the optimal solution set and trigger the reconstruction of PEODSE, which in turn directs LLMs to generate new solutions. This cycle repeats until the iteration termination conditions (the maximum number of exploration design configurations) are met.

Why PEODSE? LLMs exhibit strong semantic comprehension capabilities, yet their performance on target tasks highly depends on carefully engineered prompts. While LLMMH [11] innovatively directs LLMs to perform solution generation for metaheuristics, the exponential growth in LLM parameters necessitates sophisticated prompt engineering to fully leverage domain-specific knowledge. The prompt design in LLMMH relied on minimal task descriptions, which limited the model's depth of task understanding. While established techniques such as Zero-shot [26], Few-shot [27], Optimization by PROMPTing

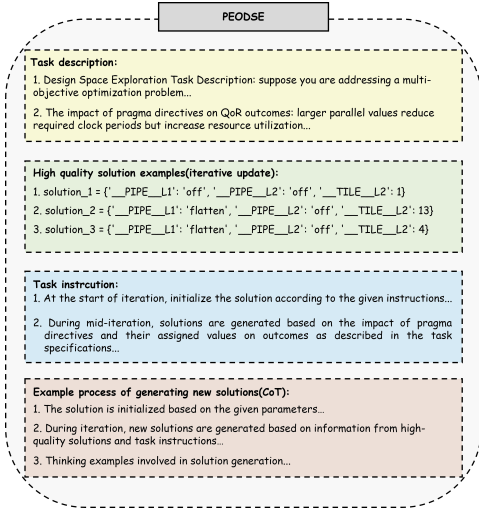


Fig. 6: Illustration of the proposed prompting engineering strategy for DSE.

(OPRO) [28], and Chain-of-Thought (CoT) [29] have shown promise in prompt engineering, we argue that enriching prompts with high-quality contextual information significantly enhances the quality of LLM-generated solutions. To this end, we propose PEODSE, a dedicated prompt formulation that provides comprehensive DSE task information, dynamically updated high-quality solutions, and reasoning traces for solution generation to reduce the probability of LLMs producing erroneous solutions.

As shown in Fig. 6, PEODSE comprises four components: task description, high-quality solution examples, task instruction, and solution generation exemplars. The task description introduces the task background and the impact of pragma directives with their values on QoR outcomes. High-quality solution examples, inspired by Optimization by PROMpting methodology, provide exemplary configurations and are dynamically updated during iterations. Task instruction delivers explicit guidance for LLMs to generate solutions. Within the task instruction, we incorporate the impact of pragma directives on QoR to enable more intelligent design space exploration (e.g., setting the pipeline directive to "off" reduces utilization such as LUTs while relatively increasing latency. Conversely, setting it to "flatten" yields the opposite effect.). Solution generation exemplars incorporate Chain-of Thought reasoning to demonstrate stepwise solution derivation processes.

IV. EXPERIMENTS

A. Experimental Setup

Based on the dataset from GNN-DSE [4], as detailed in Section III-A, we construct a multimodal graph learning dataset for QoR prediction, implementing a more distinct partitioning scheme than GNN-DSE. As depicted in Table II, we select five target kernels from MachSuite [30] and ten target kernels from Polyhedral benchmarks [31] as our training set. 70% of the training set is allocated for training, 15% for testing, and 15% for validation. All GNN models employ a hidden layer dimension of 128, optimized via Adam with a batch size of 64, learning rate of 0.001, and 500+ iterations. Experiments

TABLE II: The dataset in the training set consists of a total of 15 benchmarks, with 4,353 graph-text samples in total. Dataset Size denotes the number of graph-text samples employed per benchmark for training or inference purposes.

Kernel	Description	Dataset Size
<i>adi</i>	Alternating direction implicit solver	322
<i>aes</i>	A common block cipher	45
<i>atax</i>	Matrix transpose and vector multiplication	227
<i>bicg</i>	BiCG sub kernel of bicgstab linear solver	347
<i>doitgen</i>	multi-resolution analysis kernel	74
<i>fdtd-2d</i>	2-D finite different time domain kernel	289
<i>gemm-blocked</i>	A blocked version of matrix multiplication	243
<i>gemm-ncubed</i>	Dense matrix multiplication	362
<i>gemver</i>	Vector multiplication and matrix addition	464
<i>gemm-p</i>	Matrix-multiply	303
<i>gesummv</i>	Scalar, vector and matrix multiplication	159
<i>mvt</i>	matrix vector product and transpose	476
<i>spmv-crs</i>	Sparse matrix-vector multiplication(variable-length)	73
<i>spmv-ellpack</i>	Sparse matrix-vector multiplication(fixed-size)	114
<i>2mm</i>	2 matrix multiplications	388
<i>3mm</i>	3 matrix multiplications	467

TABLE III: Kernels used for inference and design space exploration.

Kernel	Description	Dataset Size	# Design configs
<i>heat-3d</i>	Heat equation over 3D data domain	225	71511
<i>jacobi-1d</i>	1-D Jacobi stencil computation	222	2871
<i>jacobi-2d</i>	2-D Jacobi stencil computation	524	7609187
<i>nw</i>	A dynamic programming algorithm	386	6615
<i>seidel-2d</i>	2-D Seidel stencil computation	56	10919
<i>stencil</i>	A two-dimensional stencil computation	524	7591

execute on the AMD Ultrascale+ MPSoC ZCU104 platform. Benchmarks synthesize using Vitis-HLS 2022.1 and Vivado 2022.1 to collect ground-truth Latency and resource utilization (LUT, DSP, FF, BRAM) metrics, which serve as training labels.

B. Evaluation of HLS QoR Prediction Accuracy

We first evaluate HLS QoR prediction performance of the proposed MPM. Model training employs the dataset detailed in Table II. To demonstrate generalization capability, inference is performed on completely unseen kernels, with dataset specifics presented in Table III. Our comparative analysis includes state-of-the-art (SOTA) approaches: GNN-DSE [4], HGBO [3], IronMan-Pro [9], PROGS [7]. Experimental results demonstrate that MPM achieves the lowest prediction errors across all targets: 0.3870 (Latency), 0.0004 (LUT), 0.0004 (DSP), 0.0015 (FF), 0.0005 (BRAM). Compared with ProgSG, our approach significantly outperforms it by up to 10.25 \times . MPM's superior performance originates from its enhanced multimodal feature fusion capability and utilization of text feature extractors with stronger semantic comprehension, which effectively establish correspondences between textual features and source code characteristics.

We also performed ablation study by comparing GNN-only and LM-only with MPM. The results reveal that standalone LM outperforms ECoGNN, highlighting the untapped value of code semantics for QoR prediction. By fusing graph features from CDFGs (via ECoGNN) with textual features (via LM) under the multimodal graph learning framework, MPM significantly outperforms GNN-only approaches.

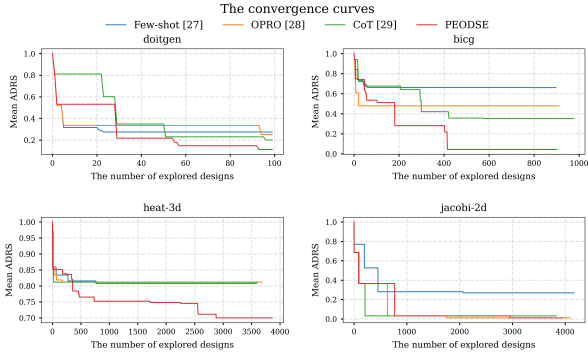


Fig. 7: Convergence curves achieved by different prompting strategies.

TABLE IV: RMSE loss of MPM and SOTA models on unseen applications.

Model	Latency	LUT	DSP	FF	BRAM	All
GNN-DSE [4]	0.7759	0.0025	0.0023	0.0078	0.0023	0.7909
HGBO [3]	0.5754	0.0127	0.0041	0.0064	0.0069	0.6082
IronMan-Pro [9]	0.4201	0.0012	0.0015	0.0016	0.0007	0.4251
PROGSG [7]	0.4061	0.0041	0.0018	0.0081	0.0025	0.4226
ECoGNN-only	0.4019	0.0016	0.0015	0.0016	0.0015	0.4084
LM-only	0.3920	0.0011	0.0011	0.0015	0.0009	0.3965
MPM	0.3870	0.0004	0.0004	0.0015	0.0005	0.3898

C. Evaluation of Design Space Exploration

To demonstrate the efficacy of our proposed PEODSE method over alternative prompting approaches in DSE tasks, we evaluate various prompting strategies using the LLM4DSE framework. The quality of approximate Pareto-optimal sets is quantified using the Average Distance from the Referenced Set (ADRS) metric [32]:

$$ADRS(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\lambda \in \Gamma} \min_{\mu \in \Omega} f(\lambda, \mu) \quad (6)$$

where Γ represents the reference Pareto-optimal set, Ω denotes the approximate Pareto-optimal set, and the function f computes the distance between λ and μ . A lower ADRS value indicates a higher accuracy of the approximate Pareto-optimal set relative to the reference. We employ GPT-3.5-turbo [33] as the LLM for comparative experiments on prompting strategies, utilizing OpenAI’s API ¹. The convergence curves achieved by different prompting methods are presented in Fig. 7. The results demonstrate that PEODSE outperforms other prompting methods. This advantage arises from PEODSE’s ability to integrate the strengths of existing methods, coupled with the domain-specific knowledge of how pragma directives impact QoR. As a result, LLM gain a deeper understanding of DSE tasks and consistently find high-quality design configurations.

Using our proposed MPM-LLM4DSE framework with PEODSE prompting, we perform comparative experiments comparing it against several established methods: multi-objective genetic algorithm (NSGA-II) [34], simulated annealing (SA) [35], ant colony optimization (ACO) [36], and LLMMH [11]. The experiments target kernels that are unseen for the training

¹<https://openai.com/index/openai-api/>

TABLE V: ADRS results and runtime on unseen applications.

Benchmark	NSGA-II [34]	SA [35]	ACO [36]	LLMMH [11]	LLM4DSE(GPT-4o)	LLM4DSE(Qwen3)
heat-3d	0.0756	0.0672	0.0784	0.0645	0.0602	0.0486
jacobi-1d	0.0320	0.0253	0.0255	0.0298	0.0212	0.0228
jacobi-2d	0.0111	0.0014	0.0201	0.0029	0.0086	0.0035
seidel-2d	0.0517	0.0551	0.0606	0.0492	0.0391	0.0406
stencil	0.0946	0.0742	0.0893	0.0742	0.0636	0.0666
nw	0.0028	0.1197	0.1812	0.0121	0.0018	0.0010
Average ADRS	0.0447	0.0572	0.0758	0.0388	0.0324	0.0305
Overall Runtime (s)	1081	664	1216	10941	7222	9847
LLM4DSE(Qwen3) Improv. over NSGA-II					31.77%	
LLM4DSE(Qwen3) Improv. over SA					46.68%	
LLM4DSE(Qwen3) Improv. over ACO					59.76%	
LLM4DSE(Qwen3) Improv. over LLMMH					21.39%	

TABLE VI: Comparative analysis of ADRS values between GNN-DSE and LLM4DSE on large benchmarks under identical time constraints.

Benchmark	# Design configs	Time(s)	LLM4DSE(Qwen3)	GNN-DSE [4]	Improv.(%)
heat-3d	71511	3068	0.0486	0.0689	30.78
jacobi-2d	7609187	4462	0.0035	0.0074	52.70
seidel-2d	10919	581	0.0666	0.0779	14.51

presented in Table III, leveraging high performance LLM (GPT-4o [37], Qwen3-235B-A22B-Thinking-2507 [38]), with results summarized in Table V. To ensure a fair comparison, all algorithms are set to explore identical number of designs. The experimental results demonstrate that both LLM4DSE combinations achieve superior exploration accuracy compared to traditional metaheuristics and LLMMH. LLM4DSE (Qwen3) yields the lowest ADRS value, showing average improvements of 46.07% over conventional metaheuristic algorithms and 21.39% over LLMMH. Additionally, LLM4DSE reduces runtime relative to LLMMH due to optimized prompt design that minimizes the latency of LLM reasoning. The current implementation of LLM4DSE is based on API calls. By localizing large language models, communication latency can be greatly reduced, which in turn may significantly shorten the runtime of LLM4DSE.

To further evaluate the effectiveness of LLM4DSE, we compare it with the exact DSE algorithm of GNN-DSE [4] on benchmarks with large design spaces containing up to 7 million design configurations. For each benchmark, we record LLM4DSE’s runtime, then execute GNN-DSE for the same duration and report its ADRS. Table VI shows that under identical time constraints, LLM4DSE identifies higher-quality design configurations, exhibiting a 32.66% reduction in ADRS over GNN-DSE, demonstrating its superior performance in large design spaces.

V. CONCLUSION

This paper presents a novel LM-GNN framework for HLS QoR prediction, offering an efficient and accurate alternative to conventional HLS tools. Our analysis reveals that LMs outperform GNN architectures in QoR prediction. Furthermore, the proposed PEODSE prompting methodology significantly enhances LLM4DSE’s task comprehension capabilities, leading to improved DSE outcomes. This work establishes a new paradigm for HLS DSE, highlighting the untapped potential of LMs and LLMs in hardware design. Future work will focus on employing smaller, fine-tuned and task-specific models for local execution to alleviate the computational burden and exploring its application in cross-platform synthesis.

REFERENCES

- [1] N. Wu *et al.*, “Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning,” in *GLSVLSI*. ACM, 2021, p. 39–44.
- [2] Z. Lin *et al.*, “Powergear: Early-stage power estimation in fpga hls via heterogeneous edge-centric gnns,” in *DATE*, 2022, pp. 1341–1346.
- [3] H. Kuang *et al.*, “Hgbo-dse: Hierarchical gnn and bayesian optimization based hls design space exploration,” in *ICFPT*, 2023, pp. 106–114.
- [4] A. Sohrabizadeh *et al.*, “Automated accelerator optimization aided by graph neural networks,” in *DAC*. ACM, 2022, p. 55–60.
- [5] L. Ferretti *et al.*, “Graph neural networks for high-level synthesis design space exploration,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 2, 2022.
- [6] M. Gao *et al.*, “Hierarchical source-to-post-route qor prediction in high-level synthesis with gnns,” in *DATE*, 2024, pp. 1–6.
- [7] Z. Qin *et al.*, “Cross-modality program representation learning for electronic design automation with high-level synthesis,” in *MLCAD*. ACM, 2024.
- [8] Z. Wang *et al.*, “Machine learning to set meta-heuristic specific parameters for high-level synthesis design space exploration,” in *DAC*. IEEE, 2020.
- [9] N. Wu *et al.*, “Ironman-pro: Multiobjective design space exploration in hls via reinforcement learning and graph neural network-based modeling,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 3, pp. 900–913, 2023.
- [10] Y. Yao *et al.*, “Decomposition based estimation of distribution algorithm for high-level synthesis design space exploration,” *Integration*, vol. 100, p. 102292, 2025.
- [11] L. Xu *et al.*, “Intelligent4dse: Optimizing high-level synthesis design space exploration with graph neural networks and large language models,” *arXiv preprint arXiv:2504.19649*, 2025.
- [12] C. Lattner *et al.*, “Llvm: A compilation framework for lifelong program analysis & transformation,” in *International symposium on code generation and optimization*. IEEE, 2004, pp. 75–86.
- [13] C. Cummins *et al.*, “Programl: A graph-based program representation for data flow analysis and compiler optimizations,” in *ICML*, 2021, pp. 2244–2253.
- [14] E. Tang *et al.*, “Understanding LLM embeddings for regression,” *Transactions on Machine Learning Research*, 2025.
- [15] C. K. Joshi, “Transformers are graph neural networks,” *arXiv preprint arXiv:2506.22084*, 2025.
- [16] J. Devlin *et al.*, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019, pp. 4171–4186.
- [17] K. Clark *et al.*, “What does bert look at? an analysis of BERT’s attention,” in *ACL Workshop BlackboxNLP*. ACL, 2019, pp. 276–286.
- [18] S. Zhou *et al.*, “CodeBERTScore: Evaluating code generation with pretrained models of code,” in *EMNLP*. ACL, 2023, pp. 13 921–13 937.
- [19] B. Finkelshtein *et al.*, “Cooperative graph neural networks,” in *ICML*, 2024.
- [20] J. L. Ba *et al.*, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.
- [21] E. Jang *et al.*, “Categorical reparameterization with gumbel-softmax,” in *ICLR*, 2017.
- [22] W. L. Hamilton, *Graph representation learning*. Morgan & Claypool Publishers, 2020.
- [23] Y. Li *et al.*, “Gated graph sequence neural networks,” in *ICIR*, 2016.
- [24] H. Bao *et al.*, “Vlmo: Unified vision-language pre-training with mixture-of-modality-experts,” in *NeurIPS*, vol. 35, 2022, pp. 32 897–32 912.
- [25] A. Vaswani *et al.*, “Attention is all you need,” in *NeurIPS*, 2017, p. 6000–6010.
- [26] A. Radford *et al.*, “Language models are unsupervised multitask learners,” 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:160025533>
- [27] T. B. Brown *et al.*, “Language models are few-shot learners,” in *NeurIPS*, 2020.
- [28] C. Yang *et al.*, “Large language models as optimizers,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [29] J. Wei *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” in *NeurIPS*, 2022.
- [30] B. Reagen *et al.*, “Machsuite: Benchmarks for accelerator design and customized architectures,” in *IISWC*. IEEE, 2014, pp. 110–119.
- [31] T. Yuki *et al.*, “Polybench, 4.2.1,” 2010, <https://github.com/MatthiasJReisinger/PolyBenchC-4.2.1>.
- [32] B. C. Schafer *et al.*, “High-level synthesis design space exploration: Past, present, and future,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2628–2639, 2019.
- [33] L. Ouyang *et al.*, “Training language models to follow instructions with human feedback,” in *NeurIPS*, 2022.
- [34] B. C. Schafer, “Parallel high-level synthesis design space exploration for behavioral ips of exact latencies,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 4, 2017.
- [35] B. C. Schafer *et al.*, “Adaptive simulated annealer for high level synthesis design space exploration,” in *International Symposium on VLSI Design, Automation and Test*, 2009, pp. 106–109.
- [36] B. Carrion Schafer, “Probabilistic multiknob high-level synthesis design space exploration acceleration,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 394–406, 2016.
- [37] OpenAI *et al.*, “Gpt-4o system card,” *arXiv preprint arXiv:2410.21276*, 2024.
- [38] A. Yang *et al.*, “Qwen3 technical report,” *arXiv preprint arXiv:2505.09388*, 2025.