Invited Paper: LLM-Enhanced GPU-Optimized Physical Design at Scale

Yi-Chen Lu, Hao-Hsiang Hsiao, and Haoxing Ren
NVIDIA Research
{yilu, haoxingr}@nvidia.com

***Abstract*—Modern Physical Design (PD) flows face a dual challenge: proprietary, heterogeneous design data and the rapid evolution of process nodes, both of which block models from transferring to new chips. To overcome these hurdles, we demonstrate a unified, data-driven framework that distills critical netlist optimization moves, including gate sizing, buffer insertion, and cell relocation, into "optimization primitives" learned by Large Language Models (LLMs). Particularly, we develop a high-quality, synthetic optimization data generation pipeline with commercial tools at scale, while using a GPU-accelerated differentiable Static Timing Analysis (STA) engine to create fast feedback loop, enabling end-to-end gradient propagation to guide model learning. By training on both real and synthetic data across multiple technology generations, our approach captures fundamental PD optimization patterns that transfer seamlessly to unseen designs, overcoming the constraints of fragmented design representations and proprietary data in industrial PD flows.**

I. Introduction

Machine Learning- (ML-) powered Physical Design (PD) at advanced nodes is limited less by compute than by data and accurate feedback. Models are often asked to act across heterogeneous flows with evolving design rules, yet the key features that are essential for the success of end-of-flow PPA optimization are sparse, long-tailed, and expensive to obtain. To alleviate this issue, in this work, we propose to reframe the black-boxed PD optimization process as learning and composing a small set of *optimization primitives*, where we demonstrate the applications of gate sizing, buffer insertion, cell relocation procedures in a unified manner. Particularly, due to the scarcity of high-quality optimization data, we develop targeted, PPA-configurable synthetic optimization data generation flow to generate a diverse set of data with industry-standard commercial tools for effective model training, which leverages a GPU-accelerated, differentiable, and signoff-accurate Static Timing Analysis (STA) engine, INSTA [1], for instant and accurate feedback. The result of our effort is a closed loop in which data generation and curation are as critical as the model class, and gradients couple decisions to PPA.

Concurrently, the rise of Large Language Models (LLMs) introduces a principled way to recast discrete PD moves into a token-native action space, which is an abstraction that has proved highly effective across real-world domains [2]. In PD, a wave of LLM-enhanced PPA engines has emerged, focusing on the two most leveraged optimization moves, gate sizing and buffering, which are invoked repeatedly from synthesis to signoff. For gate sizing, LEGO-Size [3] introduces a novel idea to encode each library cell as a set of learnable "tokens" (function, drive strength, and V_t), sequencing timing paths with graph context, and finally casting the gate sizing prediction task as a sequence-to-sequence modeling task. This approach has proven to be highly effective for timing optimization, which enabling policies that transfer across different designs. As for buffering, a recent work BUFFALO [4] adopts a similar principle of netlist tokenization. Particularly, it converts a net to a sequence via a custom traversal algorithm to facilitate robust sequence-to-sequence learning. However, despite this progress, a persistent challenge of applying LLMs to PD still remains: high-quality, representative training data is scarce, and LLMs demand both breadth and coverage to generalize reliably to unseen designs.

To address this challenge, we design a realistic, high-fidelity data generation pipeline leveraging an industry-leading commercial PD tool to produce PPA-targeted datasets. The core idea is that modern PD tools expose a rich set of optimization knobs that can be configured for different PPA objectives, which has been studied extensively by many works focusing on parameter tuning [5], [6], [7]. Therefore, by sweeping these configurations, we can generate diverse netlist variants representing a wide range of realistic design scenarios, allowing us to observe, for given features extracted from Static Timing Analysis (STA), power analysis and other extraction, how a black-box commercial optimizer would behave. This is especially critical in the supervised learning setting, since it enables us to directly train models to mimic the decision patterns of these mature tools, which has proven to be highly effective to warm up models' parameters for subsequent training such as Reinforcement Learning (RL), which has been validated by [8] for Concurrent Clock and Data (CCD) optimization. Notably, these selected tool configurations themselves serve as an explicit representation of intent of PPA targets, which are essential for models to take as "conditioning" input as even with identical timing and power features, altering the PPA targets can lead to markedly different optimization trajectories.

Recently, GPU-accelerated PD tools have gained significant traction, exploiting the massive parallelism of modern GPUs to accelerate core PD tasks such as placement [9], routing [10], [11], and STA [12], [1]. These tools are pivotal for next-generation PD not only for their dramatic runtime improvements, but also for their scalability to the ever-growing design sizes driven by Moore's Law with custom differentiable CUDA kernels enabled by extending modern ML infrastructure such as PyTorch[13]. Crucially, the gradient-based optimization nature allows them to be integrated seamlessly with ML models to perform better-than-tool optimization rather than purely mimicking from a static dataset. In this paper, given the fact that timing optimization sits at the heart of PD since every optimization move such as gate sizing and buffering requires timing validation, and every design implementation needs to reach timing closure, we demonstrate how to leverage the industrial-grade, open-sourced, GPU-accelerated STA engine, INSTA [1], to perform key PD optimizations that extend beyond the reach of existing commercial flows. In particular, we show that how INSTA can be coupled with LLM-based decision models to achieve timing-driven optimization with both high fidelity and unprecedented speed.

The rest of the paper is organized as follows.

II. Background and Motivation

Two of the most critical primitives in PD for PPA optimization and constraint fixing are gate sizing and buffer insertion. We identify these as high-impact targets for LLM-Enhanced GPU-Optimized (LEGO)

methods because traditional algorithms, while engineered for pseudo-linear complexity, rely heavily on heuristic rules that often fail to scale with modern design complexity. Particularly, we realize that with the relentless growth in design size and complexity driven by Moore's Law, commercial tools face increasing difficulty in executing these primitives efficiently. Conventional heuristic-based approaches frequently yield sub-optimal PPA outcomes, diminishing the benefits of advanced process scaling. This scalability challenge is exacerbated at leading-edge technology nodes (e.g., $3nm$), where hundreds of library cells may exist for a single logic function, creating a vast combinatorial search space. The explosion in solution space makes it increasingly difficult for rule-based optimizers to explore and converge to globally effective solutions within practical runtime.

In this work, we propose leveraging the LEGO-based framework to address these limitations. By combining LLM-driven decision-making transforms with GPU-accelerated, differentiable, signoff-accurate analysis, LEGO-based PD optimization can systematically explore large design spaces, adapt to varying PPA objectives, and deliver solutions that scale to the demands of advanced technology nodes while maintaining high solution quality.

A. Prior ML-Enhanced Gate Sizing Work

A number of recent efforts have sought to enhance the efficiency and quality of gate sizing within commercial PD flows using machine learning. Supervised prediction approaches, such as ECO-GNN [14], DAGSizer [15], and TransSizer [16], aim to improve design productivity by providing fast, learned sizing recommendations. While these methods can achieve high accuracy on training designs, they are fundamentally constrained by their reliance on static datasets, leading to significant generalization drops on unseen designs (e.g., TransSizer drops from 89% to 61%). As a result, they remain unable to consistently outperform the very tools they are meant to augment. Reinforcement learning methods, exemplified by RL-Sizer [17], reframe gate sizing as a Markov Decision Process (MDP) and can surpass a leading PD tool in PPA metrics, but their high runtime overhead renders them impractical for production-scale designs. Beyond discrete learning approaches, differentiable optimization frameworks such as AGD [18] integrate predictive timing models (e.g., TimingGCN [19]) to approximate arrival times and slack, enabling gradient computation of global timing objectives like TNS with respect to gate sizes. While conceptually attractive, these methods inherit the generalization limits of their learned timing surrogates and suffer from prohibitive runtime costs. For instance, AGD is reported to be 60× slower than a commercial tool even on a legacy $130nm$ node with fewer than a hundred cells, making them unsuitable for modern nodes where libraries can contain tens of thousands of cells per function type. Collectively, these limitations underscore the need for a new class of gate sizing frameworks that (i) generalize across unseen designs, (ii) operate with signoff-level timing fidelity, and (iii) scale efficiently to the vast solution spaces of advanced technology nodes. This motivates our work, which combines LLM-driven decision making, GPU-accelerated differentiable STA, and targeted synthetic data to address these challenges in a unified, production-ready framework.

B. Prior ML-Enhanced Buffering Methods

Traditional buffering techniques can be broadly categorized into conventional heuristic-driven approaches and more recent ML-based methods. Traditional algorithms, such as the van Ginneken style [20], [21], typically follow a multi-stage pipeline: constructing a Steiner tree for the net, partitioning the interconnect into segments, and then performing buffer sizing and placement through dynamic programming. While effective in certain regimes, this staged decomposition accumulates errors between steps and limits achievable PPA quality, particularly in advanced nodes with complex parasitics. Recent ML-based methods, exemplified by BufFormer [22], attempt to overcome these shortcomings by leveraging transformer architectures to directly learn buffering strategies from data. However, these approaches still rely on fragmented net representations, often focus on imitation learning rather than direct PPA optimization, and are typically validated only on isolated nets without full-chip integration.

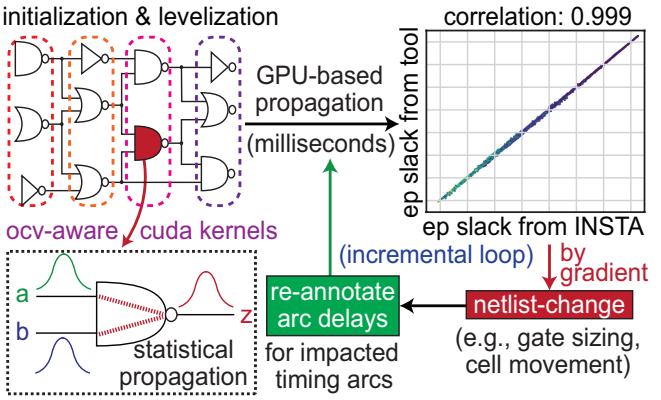

Fig. 1: Overview of INSTA which begins with a one-time initialization from a reference timing engine and performs fast, differentiable, and tool-accurate STA propagation. Notably, INSTA enables gradient computation of global timing metrics (e.g., TNS) with respect to leaf variables (e.g., gate sizes and cell locations) to drive critical PD optimization from [1].

III. INSTA: Fueling PD Optimization with Differentiable, Tool-Accurate, Statistical STA

An efficient STA engine is fundamental to guiding effective PPA optimization and is a prerequisite for deploying ML-powered PD applications in production. With the widespread availability of GPU computing resources and mature CUDA development frameworks, GPU-acceleration has emerged as a promising path to scalable PD optimization. However, prior GPU-accelerated STA (GPU-STA) approaches have struggled to gain industrial adoption, largely because they attempt to replace commercial signoff tools with standalone timing engines—an approach that cannot faithfully replicate the proprietary delay models and intricate signoff rules used in industry.

In this work, we demonstrate a fundamentally different approach with INSTA [1], the first differentiable, statistical GPU-STA engine that achieves both unprecedented accuracy and scalability by performing a one-time initialization from any reference signoff tool. This design philosophy unlocks two transformative capabilities for PD: (1) rapid, high-fidelity timing analysis for incremental netlist updates, and (2) Gradient-based, truly global timing optimization at scale.

As illustrated in Fig. 2, INSTA begins with a one-time synchronization to a reference STA engine, after which it performs OCV-aware timing propagation via a custom forward CUDA kernel and gradient backpropagation via a custom backward kernel. Experimental results show that INSTA achieves near-perfect correlation (0.9999) with an industry-leading signoff tool in endpoint slack values across multiple real-world high-performance designs in signoff mode. Achieving this correlation is far from trivial: INSTA implements precise handling of distribution-based timing propagation, timing exceptions, and

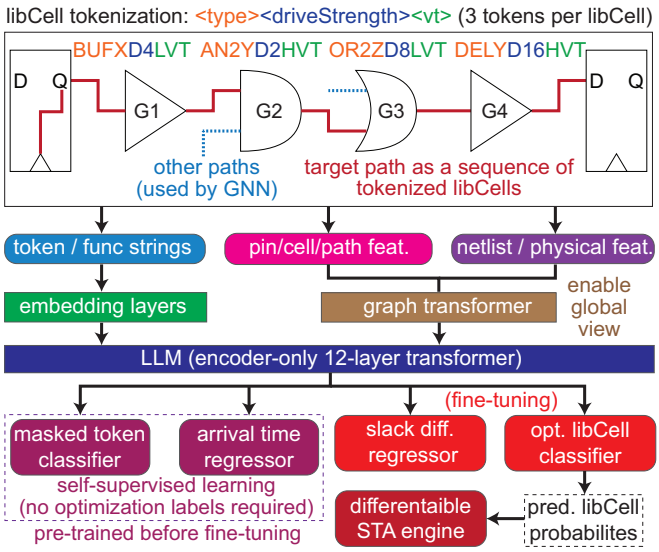

Fig. 2: High-level overview of LEGO-Size. Timing paths are considered as sequences of tokenized library cells, and the gate sizing prediction task is solved through a language modeling approach. To achieve better PPA beyond the commercial signoff tool, a differentiable STA engine is developed to refine LLM-predicted probabilities from [3].

common-path pessimism removal (CPPR), while correctly managing rise/fall conditions and unateness constraints.

Beyond replicating signoff accuracy, INSTA exposes gradients of global timing metrics—including worst negative slack (WNS) and total negative slack (TNS)—with respect to a wide range of leaf optimization variables. This enables fast, precise identification of timing bottlenecks at the level of cells, nets, or individual stages, making it possible to target optimization where it matters most. In this work, we demonstrate INSTA's capabilities in key PD optimization domains of gate sizing and buffering.

IV. LEGO-Size: Gate Sizing at Advanced Nodes

Unlike prior ML-enhanced gate sizing methods (Section II-A) that rely purely on supervised learning to imitate a target engine—an approach that inherently limits generalization and cannot surpass the tool being mimicked—LEGO-Size is designed as a scalable, generalizable framework capable of delivering instant, better-than-tool signoff timing optimization results on *unseen* designs at advanced nodes.

LEGO-Size combines a token-based LLM-driven prediction framework with a GPU-accelerated, differentiable STA engine, enabling both accurate gate sizing predictions and gradient-based refinement. We validate LEGO-Size [3] against an industry-leading signoff tool at a commercial $3nm$ node containing over 10,000 library cells, demonstrating robust transferability to previously unseen designs. At a high level, LEGO-Size comprises two main components:

- A language-model–driven gate sizing prediction framework based on path tokenization and global graph context.
- INSTA [1] guided differentiable STA engine that refines LLM-predicted sizing probabilities to directly optimize PPA metrics.

A. Optimized Gate Size Prediction

At the signoff ECO stage, path-based timing analysis (PBA) is essential for overcoming GBA pessimism, making path-by-path gate sizing a natural optimization target. As shown in Fig.2, LEGO-Size performs *path-based* gate sizing predictions to align with signoff-level workflows, similar in scope to TransSizer[16], but with critical differences that improve scalability and generalization.

To incorporate global context into path-based predictions, we employ a Graph Transformer (GT)[23] with a linear-complexity attention mechanism, capturing full-netlist features beyond the local path scope. The key innovation lies in casting gate sizing as a language modeling problem: each library cell is tokenized into exactly three tokens—cell function, drive strength, and threshold voltage type (V_{th})—and entire timing paths are represented as graph-contextual sequences. This tokenization unlocks the transferability benefits of LLMs, in contrast to TransSizer and other prior works[14], [15] that rely on fixed handcrafted features and do not operate in a language-native representation space.

Furthermore, while TransSizer uses an encoder–decoder transformer for iterative gate-by-gate predictions, LEGO-Size employs a PD-customized encoder-only transformer that predicts sizes for all cells on a path simultaneously, improving efficiency and capturing inter-cell dependencies. To address data scarcity and improve generalization, we introduce self-supervised pre-training tasks—including masked token prediction and arrival-time increment (stage-delay) prediction—that require no optimization labels. These tasks initialize model parameters before supervised fine-tuning (SFT) on PBA-based signoff sizing labels, accelerating convergence and improving performance. Notably, pre-training data can be generated from pre-optimization design states, which are abundant in any industrial PD flow, thus removing a major bottleneck in model training. The key differences between TransSizer [16] and LEGO-Size is summarized in Table I.

TABLE I: Key differences: TransSizer [16] vs. LEGO-Size from [3].

features	TransSizer [16]	LEGO-Size (ours)
model architecture	encoder-decoder	encoder-only
prediction style	iterative gate-by-gate	all gates simultaneous
language-based	no	yes (string tokens)
exact prediction	no (bin-based)	yes
pre-training tasks	no	yes
prob. refinement	heuristic search in bin	differentiable STA

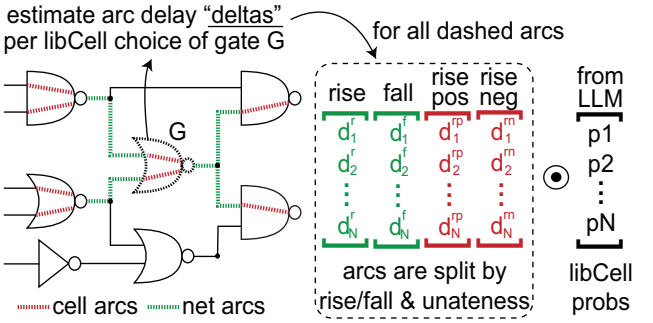

Fig. 3: Illustration of the expected delta delay computation for timing arcs in our differentiable STA process. Assume gate G has N libCells, each associated with a probability predicted by the LLM model. For each option, we use the PrimeTime estimate_eco command to estimate the delay change d for each affected arcs (dashed arcs), categorized by rise/fall and unateness. A dot product is then applied between these estimated delay changes and the probabilities of each gate size to compute the total expected delta delay from [3].

TABLE II: Parameter sampling ranges used in data generation from [4].

Parameter	Sampling Range
Driver/Sink Type	ASAP7 standard cell library
Driver/Sink Size	ASAP7 standard cell library
Fanout Count	[1, 100]
Delay Target	[100 ps, 1 ns]
Input Transition	[0, 0.2×T_{clk}]
Input Delay	[0, 0.2×T_{clk}]
Driver/Sink Placement	[0, Width] × [0, Height]
Leakage/Dynamic Ratio (%)	[0, 100%]

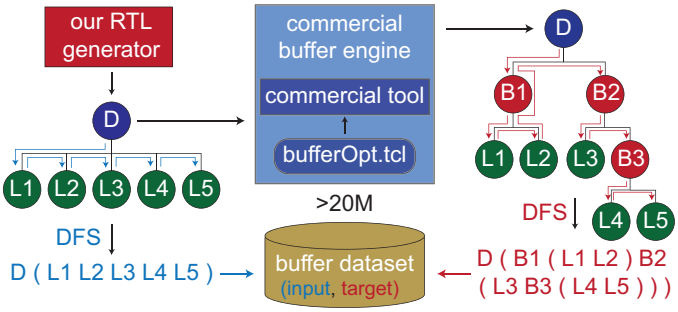

Fig. 4: Overview of our dataset generation with our RTL generator and commercial buffer engine from [4].

B. From Prediction to Optimization: INSTA-Guided Probability Refinement

Now we discuss the second component of LEGO-Size. We reckon that accurate prediction alone is insufficient as our objective is to surpass commercial tool quality. To achieve this, we integrate a GPU-accelerated differentiable STA engine capable of refining LLM-predicted sizing probabilities by directly optimizing total negative slack (TNS) through gradient descent.

Unlike previous GPU-accelerated [12], [**?**], [**?**] or differentiable STA works [18], [**?**] that mainly focus on full graph STA update, our differentiable STA propagation is based on an incremental (or delta-based) philosophy. Specifically, given the initial state of a netlist before signoff optimization, and the LLM-predicted gate size probabilities for refinement, we first sync the arc delays from the initial states, and then re-annotate them by multiplying the expected delay changes (incurred by new gate sizes), calculated via what-if analysis, by the corresponding predicted probabilities. For instance, given a cell arc t with an original delay D^t_{org}, and the LLM-predicted probability distribution $P = [p_1, p_2, p_3]$ across three potential library cell candidates, we first compute the expected delay changes $D = [d_1, d_2, d_3]$ for each candidate with what-if analysis. Then, the total expected delay change for arc t is calculated as $P \cdot D$, leading to a new annotated delay of $D^t_{new} = D^t_{org} + P \cdot D$. Note the this what-if analysis to compute the expected delay changes D is common in commercial signoff tools and can be performed efficiently using CPU multi-threading as it does not introduce any real update (i.e., commit) to the underlying netlist. In PrimeTime, the command for such analysis is called **estimate_eco**, which computes delay change estimates for millions of timing arcs in mere seconds [**?**].

Figure 3 illustrates the expected delta delay computation process in LEGO-Size. When sizing a cell G, we consider the impacted arcs as the dashed cell (red) and net (green) arcs in the figure, which approximately span a 3-hop neighborhood around G. Note that while the theoretical timing impact of sizing a cell can propagate throughout the netlist, this ripple effect is pointed out to diminish quickly in RL-Sizer [17], where a 3-hop neighborhood is shown to be sufficient for capturing the relevant timing impact.

Finally, after estimating the new delays for all arcs in the netlist impacted by the LLM-predicted gate size probabilities P, we leverage our STA kernels to perform instantaneous timing propagation with the newly annotated delays in an end-to-end differentiable manner. Particularly, our STA engine considers the gate size probabilities P as leaf differentiable variables, and calculates the gradient of TNS with respect to each gate size choice (i.e., $\frac{\partial TNS}{\partial P_i}$) directly. These gradients are used to refine the LLM-predicted probabilities, optimizing TNS in a truly global, full-graph optimization approach.

TABLE III: Cell and Pin Features for Model Input from [4].

Category	Features
timing arrival	arrival (max/min rise/fall)
capacitance	pin (max/min rise/fall)
slew rate	slew (max/min rise/fall)
slack metrics	slack (max/min rise/fall)
voltage levels	driver pin/ rail voltage (max/min)
fan Metrics	Fan-in; fan-out count; fan-out load
physical Dim.	cell area; bbox; pin count
Library Pin Attributes	Pin cap (max rise/fall); drive resistance (rise/fall); fan-out load

V. BUFFALO: Generalizable, One-Shot VLSI Buffering with High-Quality, Synthetic Dataset

In contrast to gate sizing which is often performed with global context, traditional buffering is typically executed incrementally, relying on local features and point-by-point decisions. This localized approach frequently leads to suboptimal PPA outcomes, as it overlooks broader netlist context and inter-dependencies. One of the most challenging buffering problems in PD is high-fanout net (HFN) buffering, which is inherently complex due to the need to balance electrical, physical, and timing constraints across a large set of sinks. Conventional solutions, such as Van-Ginneken–style algorithms, operate sequentially and heuristically, often ignoring valuable context such as local placement density, routing congestion, and path-specific timing sensitivities.

An additional challenge lies in obtaining clean, representative optimization data for model training. In production flows, buffers are typically inserted in multiple passes, each potentially separated by timing re-analysis, making it difficult to attribute a particular buffer insertion to a specific objective (e.g., long-path timing improvement versus local transition fixing). As a result, reconstructing a consistent, high-quality training corpus from organic tool logs is nontrivial.

To address these challenges, we present BUFFALO [4], a unified, one-shot buffering framework that integrates LLM-based decision modeling with the INSTA differentiable timing engine. BUFFALO treats HFN buffering as a high-dimensional, global action space problem and performs direct, one-pass optimization of the entire buffer tree. This is made feasible through Group Relative Policy Optimization (GRPO) [24], a reinforcement learning method whose effectiveness in PD is unlocked by INSTA's ability to provide fast, signoff-accurate gradients and high-fidelity PPA evaluation.

A. Large Buffering Dataset Generation

Robust LLM-style training demands large, diverse datasets that capture realistic buffering scenarios. To this end, we develop an

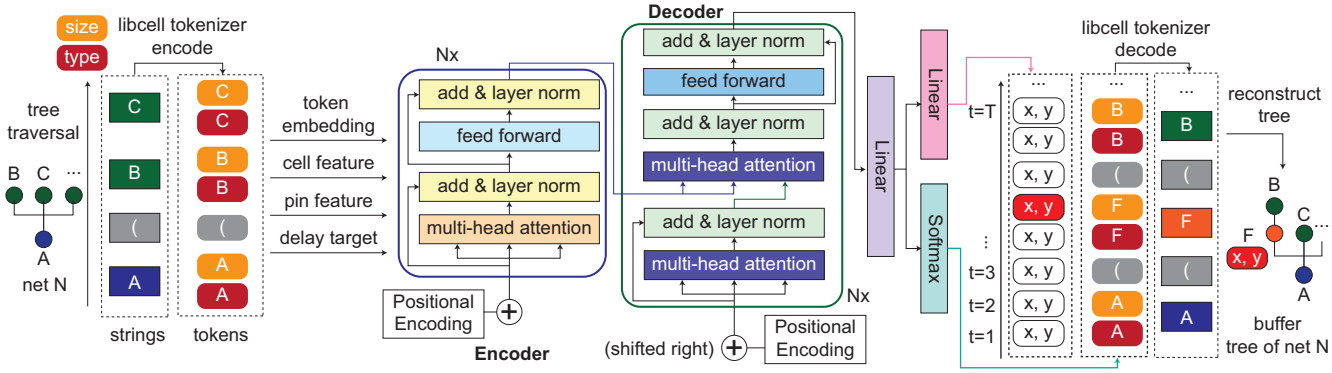

Fig. 5: Detailed architecture of our T5-based encoder-decoder model including dual-head decoding for structured tokens and coordinate prediction from [4].

automated, end-to-end data generation pipeline (Fig. 4) capable of producing over 20 million paired examples of unbuffered and buffered nets.

We begin by generating diverse high-fanout netlists using an RTL net generator, synthesizing under realistic constraints (Table II) that vary driver/sink placements, fanout counts, delay targets, input transitions, and input delays. This ensures that the resulting dataset spans a comprehensive range of electrical and physical configurations.

Next, we employ a custom bufferOpt.tcl script integrated into a commercial PD tool to insert buffers under realistic timing and design constraints. The output is a matched pair: the original unbuffered net and the fully buffered version optimized by the tool. Each pair is then serialized into a structured depth-first search (DFS) bracketed sequence—for example: • Unbuffered: D(L˙1 L˙2 L˙3 L˙4 L˙5) • Buffered: D(B˙1(L˙1 L˙2); B˙2(L˙3 B˙3(L˙4 L˙5)))

This serialization captures the complete topology, buffer types, and hierarchical structure of the net, enabling direct supervised learning and evaluation. The resulting dataset provides the diversity, realism, and consistency necessary for training high-capacity models that generalize to unseen HFNs in full-chip contexts.

B. BUFFALO Architecture

BUFFALO adopts a tokenization strategy similar to LEGO-Size, encoding each library cell into a finite set of tokens. These tokens are processed by a T5 encoder–decoder backbone (Fig. 5) customized for buffering tasks. Unlike gate sizing, buffering prediction must jointly determine the topology of the buffered net, the types of inserted library cells, and the physical coordinates of newly placed buffers. To handle this multi-output setting, we extend the T5 decoder to emit both structured tokens and associated spatial coordinates.

The architecture retains the standard T5 building blocks—stacked multi-head self-attention, cross-attention, and position-wise feed-forward layers—augmented by residual connections and layer normalization for stable optimization. The attention mechanisms are particularly well-suited to capturing the long-range dependencies that arise in complex net topologies.

Using the 20M high-quality synthetic net pairs generated from commercial tools, we fine-tune the model to convergence within one week on 8×A100 GPUs via data and model parallelism. We formulate buffer-tree synthesis as a sequence-to-sequence task, translating depth-first search (DFS)–linearized unbuffered net sequences into fully buffered nets in a single shot. Specifically, given an input net q, we perform the following in order:

- Tokenize each library cell into paired tokens (type and size) and extract associated cell-level features.
- Embed the token and feature streams separately, add positional encodings, and feed them jointly into the T5 encoder to produce context-rich embeddings.
- Decode the output sequence, which includes both structural tokens defining the buffer tree and spatial coordinates for buffer placements.

We train with a composite loss function that combines structural token classification and masked coordinate regression:

$$\mathcal{L} = \sum_i \Big[-\log p(\text{token}_i) + \lambda\, m_i\, \|\hat{\mathbf{l}}_i - \mathbf{l}_i\|_2^2 \Big], \tag{1}$$

where the mask m_i activates coordinate loss only at buffer size token positions, ensuring precise learning of meaningful buffer locations. Here $\hat{\mathbf{l}}_i$ and $\mathbf{l}_i$ denote predicted and ground-truth buffer coordinates, respectively, and the hyperparamete $lambda$ balances structural and spatial loss. The training dynamics are shown in Fig. 7, with rapid convergence achieved due to the combination of high-quality synthetic data and scalable parallel training.

C. Group Relative Policy Optimization (GRPO)

While supervised fine-tuning (SFT) teaches the policy π_θ to mimic commercial tool outputs, it assumes all reference solutions are equally optimal, neglecting the nuanced trade-offs between power, performance, and area (PPA). To explicitly model these trade-offs, we introduce Group Relative Policy Optimization (GRPO), a lightweight reinforcement learning refinement method that directly aligns model predictions with PPA objectives.

GRPO improves upon traditional preference-tuning techniques—such as RLHF [25], [26] and DPO [27]—by evaluating multiple candidate solutions per net in parallel and leveraging multi-candidate group feedback instead of single or pairwise preferences. For each net, GRPO, we do the following in order:

- Generates multiple candidate buffer trees.
- Evaluates each candidate's relative PPA quality using Group Relative Advantage Estimation (GRAE).
- Computes the mean advantage across the group as a baseline, avoiding the need for an explicit value-function critic or ranking model, thus reducing computational cost.

Particularly, the GRAE definition is as follow:

$$A^{(g)} = w_1 \frac{\Delta\text{TNS}^{(g)} - \mu_{\text{TNS}}}{\sigma_{\text{TNS}}} + w_2 \frac{-\Delta Area^{(g)} - \mu_{\text{Area}}}{\sigma_{\text{Area}}}, \tag{2}$$

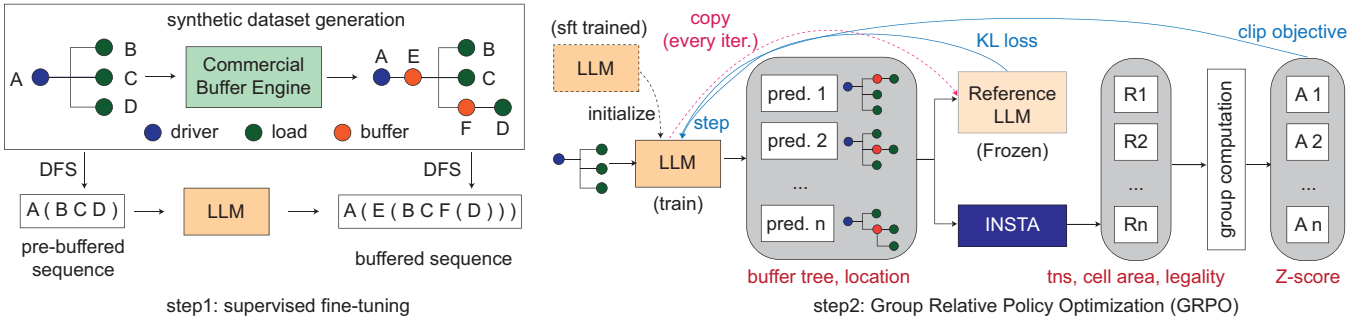

Fig. 6: Overall training workflow illustrating supervised fine-tuning followed by iterative GRPO policy refinement from [4].

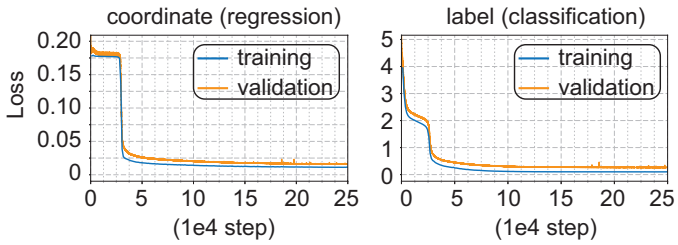

Fig. 7: Training and validation loss curves for coordinate regression (left) and label classification (right) during supervised fine-tuning, demonstrating smooth convergence for both tasks from [4].

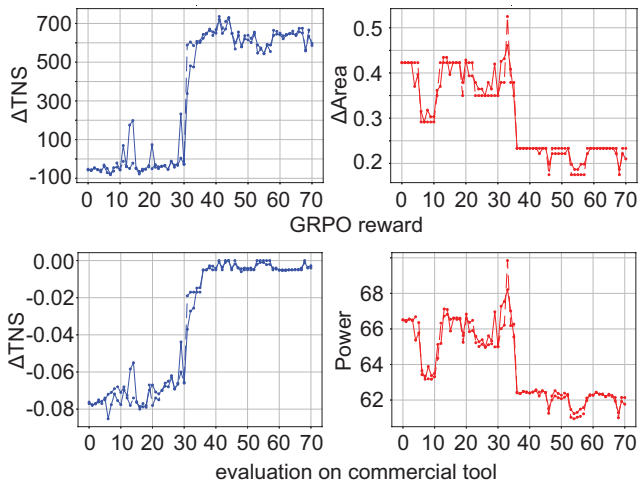

Fig. 8: Validation of GRPO reward proxy vs. actual design objectives from [4].

where $\Delta\mathrm{TNS}$ measures timing improvement and $\Delta\mathrm{Area}$ measures area overhead. Each objective is standardized to maintain scale invariance, then combined via convex weights w_1 and w_2. Finally, GRPO relies on INSTA [1] for rapid, signoff-accurate evaluation of each candidate, making it feasible to run RL-based refinement in minutes. Fig.8 shows the improvement trajectory, with each epoch representing a complete GRPO update cycle.

VI. Discussion

In this work, we have revisited recent advances in LLMS and shown how, when coupled with differentiable optimization and GPU acceleration, they can open a new design space for PD optimization. LLMs, with their ability to operate in token-native action spaces, provide a powerful way to represent and predict optimization primitives in critical PD tasks such as gate sizing and buffering. By moving beyond imitation of commercial tool outputs, these models can generalize to unseen designs and adapt to different PPA objectives with distinct conditioning inputs.

Further more, we showcase that differentiable techniques enabled by modern ML infrastructure as PyTorch is extremely powerful to guide effective PD optimization, which is exemplified by, INSTA, a GPU-accelerated, differentiable STA engine. We show that the timing gradients computed by INSTA provides key to refining model predictions into truly better-than-tool solutions. Furthermore, with custom CUDA operations defining forward and backward propagation of local operation, we argue that the differentiability allows gradient signals flow from global timing metrics such as WNS and TNS that have been focused in this paper to propagate directly to decision variables which will enable targeted, global optimization at scale, where traditional non-linear solvers cannot handle. In addition, with GPU-acceleration, we not only makes these gradients available at scale but also supports efficient search and exploration, making RL–based refinement methods such as GRPO, a key technique in modern LLM fine-tuning, being practical for PD optimization.

Another highlight of our study is the identification that high-quality, diverse training data is critical to the success of LLM-powered PD optimization. In certain optimization domain like buffering, such data is often scarce and labels are often unknown (e.g., given a buffer tree, it is hard to discern the reason why each buffer is inserted at a certain location with a particular library cell choice), particularly for rare but high-impact cases like high-fanout net buffering. BUFFALO [4] demonstrates how targeted synthetic dataset generation which is built directly from commercial tools, parameterized scenario sampling, and structured serialization can produce millions of realistic training examples for LLMs to learn from. This synthetic but high-fidelity approach not only fills coverage gaps but also ensures consistent, clean supervision signals for high-capacity models.

VII. Conclusion

In conclusion, in this work, we present a unified methodology that integrates LLM-driven prediction, differentiable refinement, GPU-enabled scalability, and synthetic data generation. This framework enables us to revisit long-standing and extremely critical PD primitives such as buffering and gate sizing with new capabilities, moving beyond the limitations of traditional heuristic, dataset-bound, imitation-based learning. Going forward, we believe that the synergy of LLMs, differentiable modeling, and targeted synthetic datasets will be central to redefining how classical PD algorithms are designed, trained, and deployed, ultimately paving the way for a new generation of globally optimal, PPA-driven optimization flows.

REFERENCES

[1] Y.-C. Lu, Z. Guo, K. Kunal, R. Liang, and H. Ren, "Insta: An ultra-fast, differentiable, statistical static timing analysis engine for industrial physical design applications," in *2025 62th ACM/IEEE Design Automation Conference (DAC)*, 2025.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] Y.-C. Lu, K. Kunal, G. Pradipta, R. Liang, R. Gandikota, and H. Ren, "Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes," in *Proceedings of the 2025 International Symposium on Physical Design*, pp. 152–162, 2025.

[4] H.-H. Hsiao, Y.-C. Lu, S. K. Lim, and H. Ren, "Buffalo: Ppa-configurable, llm-based buffer tree generation via group relative policy optimization," in *IEEE/ACM International Conference on Computer-Aided Design*, 2025.

[5] H.-H. Hsiao, P. Vanna-Iampikul, Y.-C. Lu, and S. K. Lim, "Ml-based physical design parameter optimization for 3d ics: From parameter selection to optimization," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

[6] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Doomed run prediction in physical design by exploiting sequential flow and graph learning," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.

[7] H.-H. Hsiao, Y.-C. Lu, P. Vanna-Iampikul, and S. K. Lim, "Fasttuner: Transferable physical design parameter optimization using fast reinforcement learning," in *Proceedings of the 2024 International Symposium on Physical Design*, pp. 93–101, 2024.

[8] Y.-C. Lu, W.-T. Chan, D. Guo, S. Kundu, V. Khandelwal, and S. K. Lim, "Rl-ccd: concurrent clock and data optimization using attention-based self-supervised reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[9] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[10] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Gamer: Gpu-accelerated maze routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 583–593, 2022.

[11] S. Lin, L. Xiao, J. Liu, and E. F. Young, "Instantgr: Scalable gpu parallelization for global routing," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2024.

[12] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *Proceedings of the 39th international conference on computer-aided design*, pp. 1–9, 2020.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[14] Y.-C. Lu, S. Nath, S. Pentapati, and S. K. Lim, "Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–22, 2023.

[15] C.-K. Cheng, C. Holtz, A. B. Kahng, B. Lin, and U. Mallappa, "Dagsizer: A directed graph convolutional network approach to discrete gate sizing of vlsi graphs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–31, 2023.

[16] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren, "Transsizer: A novel transformer-based fast gate sizer," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.

[17] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, "Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 733–738, IEEE, 2021.

[18] P. Pham and J. Chung, "Agd: A learning-based optimization framework for eda and its application to gate sizing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[19] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th Annual Design Automation Conference 2022*, ACM, 2022.

[20] L. P. Van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," in *1990 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 865–868, IEEE, 1990.

[21] J. Lillis, C.-K. Cheng, and T.-T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, 1996.

[22] R. Liang, S. Nath, A. Rajaram, J. Hu, and H. Ren, "Bufformer: A generative ml framework for scalable buffering," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pp. 264–270, 2023.

[23] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer, 2022."

[24] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[25] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, pp. 53728–53741, 2023.