Advancing AI-assisted Hardware Design with Hierarchical Decentralized Training and Personalized Inference-Time Optimization

Hao (Mark) Chen[†], Zehuan Zhang[†], Wanru Zhao[‡]
Nicholas Lane[‡], Wayne Luk[†], Hongxiang Fan[†]
[†]Imperial College London, London, UK
{hc1620, zehuan.zhang22, w.luk, hongxiang.fan}@imperial.ac.uk
[‡]University of Cambridge, Cambridge, UK
{wz341, ndl32}@cam.ac.uk

Abstract*—Recent years have witnessed a significant increase in the adoption of AI techniques to enhance electronic design automation. In particular, the emergence of Large Language Models (LLM) has sparked significant interest in LLM-assisted hardware design generation, spanning applications from classical digital circuits to quantum computing. Despite substantial progress in this direction, the quality of LLM-generated hardware design still cannot meet the requirements for practical deployment. In this work, we identify three critical challenges hindering the development of LLM-assisted hardware design generation: *1)* limited data availability, *2)* varied data quality, *3)* inadequate inference-time efficiency. To address these fundamental challenges, this paper introduces a two-stage framework for AI-assisted hardware design by exploring decentralized training and personalized inference. In the first stage, we propose to harness private domain design sources through a hierarchical decentralized training mechanism that addresses data-sharing constraints. To mitigate the impact of low-quality data, we identify optimization opportunities in hardware generation tasks, using user-defined metrics for model aggregation. The second stage focuses on client personalization to enhance both speed and quality. We introduce a new metric, Trueput, to analyze LLM-assisted hardware generation efficiency. To optimize Trueput, we implement personalized inference-time acceleration and customized sampling strategies. Evaluating both classical and quantum benchmarks, our experimental results demonstrate that the proposed two-stage framework can significantly improve the model capability for hardware design generation. As orthogonal enhancements to existing methods, our framework can achieve $33\% \sim 50\%$ semantic accuracy improvement and 2.3 times speedup, depending on the difficulty of the generation tasks. Both the code and benchmarks will be released publicly upon the paper's acceptance.

I. Introduction

Recent advancements in Large Language Models (LLMs) has demonstrated their great potential in automated software programming [16] and debugging [17]. This impressive capability has sparked significant research and industrial interest in leveraging LLMs to automate hardware design development for both classical and quantum domains. In the classical hardware design, extensive research has targeted RTL design generation [4], [28], [29], [31], [43], High-Level Synthesis (HLS) coding [26], [47], [48], and hardware debugging [9], [44], [49]. In quantum design generation, IBM has pioneered the use of LLMs for quantum programming [7], which has been integrated into their Qiskit Code Assistant tool[*].

[*]https://docs.quantum.ibm.com/guides/qiskit-code-assistant

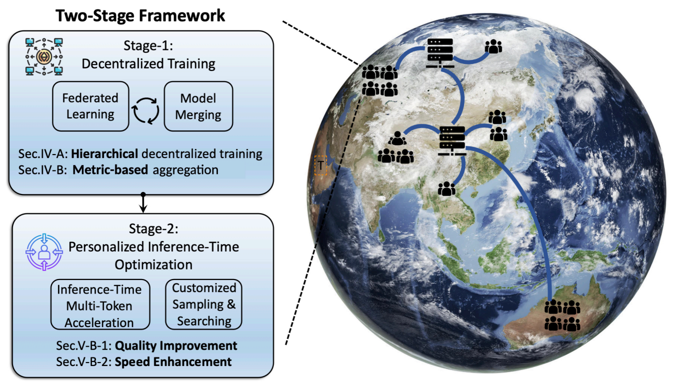
Two-Stage Framework
Stage-1: Decentralized Training
Federated Learning
Model Merging
Sec.IV-A: Hierarchical decentralized training
Sec.IV-B: Metric-based aggregation
Stage-2: Personalized Inference-Time Optimization
Inference-Time Multi-Token Acceleration
Customized Sampling & Searching
Sec.V-B-1: Quality Improvement
Sec.V-B-2: Speed Enhancement

Fig. 1: An overview of our proposed two-stage framework for the future of AI-assisted hardware design.

Although significant research efforts have been devoted to exploring LLM-assisted design generation for both classical and quantum hardware, there are still three key challenges hindering their practical use and deployment in assisting hardware design:

- *Challenge-1*: Limited availability of hardware design sources for training. Due to the low-resource nature of hardware description languages, the amount of publicly accessible classical and quantum design sources is much lower than that of software programs. For example, the size of the Qiskit dataset [7] is more than 1000 times smaller than that of the Python dataset [30].
- *Challenge-2*: Varied quality of training data. High-quality hardware designs are often proprietary and unavailable to the public. The existing training data sourced from public repositories may lack the embedded knowledge necessary for designing high-quality hardware [29].
- *Challenge-3*: Insufficient generation quality. Existing LLM-assisted methods still exhibit limited accuracy in hardware generation tasks [30]. The lack of personalized and customized optimizations during deployment time further limits the potential of LLM for hardware design generation.

Therefore, current LLM-assisted hardware generation is still in its early stages, facing significant challenges that hinder its practical deployment.

To fully unleash the potential of generative AI for the future of AI-assisted hardware design, this work proposes a two-stage framework by leveraging decentralized and personalized learning. To address *Challenge-1* of data availability, we aim to harness private domain design sources through a hierarchical decentralized training mechanism that addresses data-sharing constraints. This proposed approach includes recent advancements in federated learning [36] and model merging [50], introducing a novel hierarchical model update mechanism to facilitate broad adoption among users and organizations by accommodating varied hardware capabilities, communication infrastructures, and individual preferences. To tackle *Challenge-2* of data quality, we propose a metric-based model aggregation and merging strategy. Although data quality control in the decentralized setting is a challenging task [2], [8], [21], we identify the unique optimization opportunities for hardware design code such as correctness and performance. To overcome *Challenge-3* of generation quality, we propose personalized inference-time optimizations to enhance the generation capability. This includes speed optimization using inference-time multi-token acceleration and quality improvement through customized output token sampling. As shown in Fig. 1, these optimizations follow the decentralized training process, forming a general two-stage framework for the future of AI-assisted hardware design.

Overall, our contributions can be summarized:

- A hierarchical decentralized training paradigm with metric-based model aggregation, facilitating a broader and more diverse pool of participants for collaborative training in AI-assisted hardware design (Sec. IV).
- Personalized inference-time acceleration together with customized sampling strategies, improving both processing speed and design quality of LLM for automatic hardware generation (Sec. V).
- A comprehensive benchmarking and evaluation of the proposed two-stage framework in both classical and quantum hardware design, highlighting the effectiveness and versatility of our approach (Sec. III & Sec. VI).

II. Background and Related Work

A. Decentralized Training

Decentralized training distributes the model training across multiple nodes and devices, with only the communication of weights or gradients for model updates. The primary benefits of decentralized training are two-fold: *1)* Proprietary data preservation: By maintaining data locally on the client side for training, decentralized training circumvents data-sharing constraints. *2)* Compute efficiency: The vast computational resources available on billions of client devices can be utilized for training. In this paper, we mainly focus on two mainstreaming decentralized training: federated learning and model merging. Other advanced decentralization techniques, such as gossip learning [12], are left in future work. It is worth noting that our proposed framework is general and can be extended to accommodate any decentralized training approaches.

1) Federated Learning: As a promising approach to achieve decentralized deep learning [40], federated learning [19], [36] has been extensively studied and optimized over the past decade. The key concept of federated learning is to move model training from a central server to distributed client devices. Depending on the structure of parties (e.g. organizations or individual clients), the scale of participants, and data characteristics, federated learning is typically categorized into cross-device and cross-silo methods [18]. By performing the training locally on client devices, federated learning periodically collects and aggregates model updates from different clients. Following the introduction of the classical FedAvg algorithm [33], recent research in federated learning has primarily focused on addressing challenges related to data and system heterogeneity [51] to enable practical deployment.

2) Model Merging: With the recent advancements in language models and the increasing number of open-sourced pretrained models [46], significant research efforts have been focused on model merging that integrates the weights of multiple different models to enhance the general capability of the merged model without the need to access the original training data [15]. As the data are not shared during model merging, it provides an efficient and flexible way to learn the different expert knowledge by merging multiple domain-specific models. To preserve the generalizability and capacity of the merged model, various techniques [50] have been introduced such as weighted-based merging [32], [53], subspace-based methods [52], and routing-based approaches [35]. Mergekit [11] provides an integrated package designed to support numerous SOTA model merging techniques in a resource-efficient manner.

B. Optimization of LLM Inference

Various techniques, such as quantization , and prunning, have been introduced to improve the inference efficiency of LLM in terms of generation quality and proceeding speed. This paper mainly focuses on optimization techniques that avoid time-consuming re-training and major modifications to the model architecture.

1) Speculative Decoding & Parallel Decoding: The autoregressive nature of LLM inference predicts tokens sequentially, introducing significant data dependency and making the performance heavily memory-bound. To mitigate these issues, previous research has explored multi-token prediction approaches, where output tokens are generated in parallel rather than sequentially. Speculative decoding [24], [34] and parallel decoding [3], [5] are two mainstreaming approaches for multi-token generation, both following an iterative guess-and-verify process. During the guessing step, speculative decoding adopts a separate model to generate multiple draft tokens, while parallel decoding employs lightweight prompt tokens and embedding for multi-token drafting. In the verification step, the original model or another reward model is deployed to validate the correctness of the tokens generated during the guessing phase.

2) Inference-Time Scaling Strategy: Due to the slowing pace of LLM training law, recent research has investigated inference-time optimization to boost model performance [41].

One primary approach is self-improvement refinement, such as recursive introspection [38], where the model iteratively refines its answer conditioned on previously generated content. Another method adopts the search-and-verify paradigm [27], [45], in which the LLM performs multiple samplings with a verifier deployed in assisting the searching process to improve the final answer. This verifier can be a process-based reward model or an end-to-end evaluator to assess the quality of the generated solution.

C. Related Work

Hierarchical training has been investigated in previous research on federated learning to address network heterogeneity issues [1], [10], [14]. In contrast to previous studies, this paper considers diverse clients' conditions in the context of hardware design generation and explores a hierarchical decentralized training with a hybrid use of federate learning and model merging, encouraging a broader and more diverse pool of participants.

Applying client personalization after the training of global models has been investigated in the contexts of federated learning [20]. Different basic fine-tuning approaches have been employed for model personalization, such as regularised fine-tuning [42] and selective parameter method [25]. More advanced techniques, such as mete-learning [13], [22], have been also explored for client personalization. In our work, we focus on exploring unsupervised client personalization to accelerate LLM inference, complemented by customized sampling strategies to enhance generation quality.

III. Framework Overview and Benchmarks

A. Framework Overview

An overview of our proposed framework is illustrated in Fig. 1. Designed to leverage private-domain data for model training with privacy consideration while maximizing deployment efficiency and performance, our framework mainly consists of two stages: decentralized training and personalized inference-time optimization. These stages can be applied iteratively to collaboratively enhance the model's capabilities for AI-assisted hardware design.

The first stage of decentralized training features a hierarchical mechanism (Sec. IV-A) with hybrid federated learning and model merging, which facilitates a broader and more diverse pool of participants by considering varying hardware capabilities, connection restrictions, and individual preferences. In the second stage, different inference-time optimizations are personalized (Sec. V) for each client to enhance the inference speed and generation quality, with different hyperparameters optimized and customized to maximize the deployment performance and efficiency for diverse use cases.

B. Benchmarks

To demonstrate the effectiveness of our approaches, we perform evaluation on two different benchmarks: one for classical hardware and the other for quantum hardware.

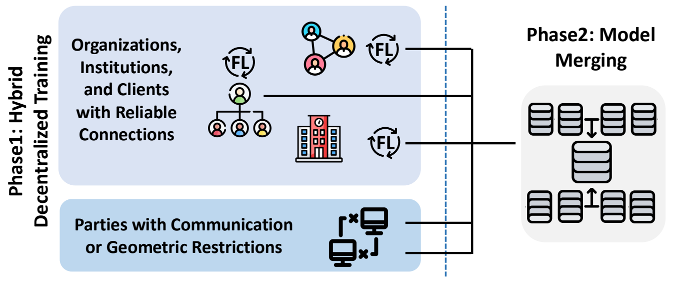
Phase1: Hybrid Decentralized Training
Organizations, Institutions, and Clients with Reliable Connections
FL
FL
FL
Parties with Communication or Geometric Restrictions
Phase2: Model Merging

Fig. 2: The vision and overview of our proposed framework for the future of AI-assisted hardware design.

Classical Hardware Benchmark. To validate the applicability of our framework in facilitating classical hardware designs, we evaluate it on a C-based High-Level Synthesis (HLS) benchmark comprising 7437 training samples and 1860 test samples. Each sample consists of a high-level design description and a canonical HLS program. The HLS designs include a wide range of domains, such as Matrix and Linear Algebra Operations, Scientific Simulation, Statistical Computation, etc.

Quantum Hardware Benchmark To evaluate the effectiveness of our approach in quantum hardware generation, we adopt a Qiskit benchmark that includes 10896 training samples and 50 test samples. Each training sample contains a Qiskit program with human-written comments, while each test sample consists of a functionality description and a canonical Qiskit program.

IV. Decentralized Training

A. Hierarchical Decentralized Training

Federated learning has demonstrated its potential for decentralized training, with both cross-silo and cross-device settings studied for various user scenarios. However, its practicality and effectiveness might decrease when deploying it for clients with poor or unreliable communication. This challenge becomes even more pronounced in extra-large-scale collaborative training settings, where geometric and infrastructural restrictions might affect deployment feasibility. Additionally, most federated learning approaches assume a shared network architecture for locally trained models, which becomes impractical in the context of LLM due to the high computational and memory requirements for LLM training on client devices. To promote the broader adoption of decentralized training for foundation models in AI-assisted hardware design, this paper proposes a hierarchical decentralized training scheme.

As illustrated in Fig. 2 and Algorithm 1, our hierarchical decentralized training consists of two tiers. The first tier is referred to as hybrid decentralized training. For clients or organizations with reliable communication channels without any restrictions, federated learning is employed for collaborative training. Multiple groups will employ federated learning within each group independently, resulting in several separately trained federated models. Meanwhile, for parties with isolated environments due to geographical or infrastructural limitations, individual local training is performed. In the second tier, different models with diverse domain knowledge, learned via either federated or local training, are combined together using model merging techniques. This hierarchical, two-tier decentralized training framework enables efficient utilization of private domain data regardless of physical or regulatory restrictions.

Algorithm 1 Hierarchical Decentralized Training

```
1: **Notation**
2: $\mathcal{C}$: Set of all clients
3: $\mathcal{C}_F$: Subset of clients with reliable communication, partitioned into $G$ groups $\mathcal{C}_g^F$, where $g \in \{1, 2, \ldots, G\}$
4: $\mathcal{C}_L$: Subset of clients with no reliable communication
5: $\text{FL}(\cdot)$: Federated learning function
6: $\text{LT}(\cdot)$: Local training function
7: $\text{MM}(\cdot)$: Model merging function
8: $M_{\text{global}}$: Final global model after merging
9: **Tier 1: Hybrid Decentralized Training**
10: **for** each group $\mathcal{C}_g^F$ in $\mathcal{C}_F$ **do**
11:     Train model $M_g^F = \text{FL}(\mathcal{C}_g^F)$        ▷ Federated Learning
12: **end for**
13: **for** each client $C_i^L$ in $\mathcal{C}_L$ **do**
14:     Train model $M_i^L = \text{LT}(C_i^L)$        ▷ Local Training
15: **end for**
16: **Tier 2: Model Merging**
17: Gather: $\mathcal{M} = \{M_1^F, \ldots, M_G^F\} \cup \{M_i^L \mid C_i^L \in \mathcal{C}_L\}$
18: Merge: $M_{\text{global}} = \text{MM}(\mathcal{M})$
19: **Output:** Global model $M_{\text{global}}$
```

B. Metric-based Aggregation

Adaptive approaches, such as client selection and quality-aware model aggregation [37], have been studied in the context of federated learning and model merging. However, prior research has mainly focused on traditional AI tasks such as classifications and segmentation, leaving their effectiveness in the broader generative AI tasks unexplored. A key challenge in applying adaptive model aggregation to generative tasks lies in evaluating the quality of generated content. Although metrics such as perplexity can be employed, these metrics require reference answers and primarily measure similarity to these references rather than the intrinsic quality of the generated content. For instance, two programs with distinct coding styles and variable naming might achieve the same functionality and performance, yet exhibit vastly different perplexity scores. Furthermore, although neural metrics like LLM-as-Judge have been proposed, these methods often lack explainability and analytical insights.

To address these challenges, this paper identifies a key optimization opportunity in hardware design generation tasks. Unlike traditional generative tasks, hardware generation inherently provides quantitative evaluation metrics—including design syntax accuracy, hardware functional correctness, and hardware latency—that can serve as robust criteria for model aggregation and merging. Leveraging this observation, we propose a flexible aggregation framework that enables users to define custom metrics for weighting model contributions. Formally, given the i-th client model M_i from a set of N client models, the final aggregated model M_f is computed as $M_f = \sum_{i=1}^{N} g(M_i) \cdot M_i$, where $g(\cdot)$ is a user-defined metric applied to client model to determine its contribution. For instance, $g(\cdot)$ could be defined using syntax accuracy to filter out model weights from clients trained on syntactically incorrect data, thereby ensuring high-quality training data for the aggregated model.

It is worth noting that our framework is not restricted to using hardware-specific metrics such as syntax accuracy and functional correctness. The specification is designed to accommodate a wide variety of model aggregation strategies. For example, if $g(.)$ is parameterized as the ratio of client training samples, the aggregation replicates the FedAvg algorithm. By enabling customization of $g(.)$, our framework caters for diverse requirements across different hardware generation tasks, allowing users to tailor aggregation strategies to their specific needs.

V. Personalized Inference-Time Optimizations

A. Trueput: Efficiency Analysis for Design Generation

To analyze the efficiency of LLM-assisted design generation, we propose a new metric, **Trueput**, which quantifies the number of functionally correct designs generated per unit of time. It is defined as:

$$\textbf{Trueput} = \frac{\text{Pass@k}}{T_{\text{inf}}} \tag{1}$$

where Pass@k represents the expected functionality pass rate when k samples are generated, and T_{inf} denotes the expected inference latency per output design.

Next, we analyze Trueput under the constraint of limited computational resources. When batching is employed, the inference latency T_{inf} is expressed as $T_{\text{inf}}(k)$ since the batch size depends on k. According to the Codex [6], an unbiased estimate of Pass@k can be written as $1 - (1 - p)^k$ with functionality pass probability p. Substituting this into the definition of **Trueput**, we obtain:

$$\textbf{Trueput}_{\text{batch}} = \frac{1 - (1 - p)^k}{T_{\text{inf}}(k)} \tag{2}$$

This formulation enables the analysis of efficiency of the inference framework by accounting for both the accuracy of the generated designs and the latency associated with batching during inference. Increasing $\textbf{Trueput}_{\text{batch}}$ requires decreasing the inference time $T_{inf}(\cdot)$, and improving functionality pass rate p. Given the form in (2), we hypothesize that a global maximum of $\textbf{Trueput}_{\text{batch}}$ exists at some finite value of k, for fixed p and $T_{inf}(\cdot)$. Therefore, the value of k should be optimized for each client to maximize $\textbf{Trueput}_{\text{batch}}$. To address these goals, this paper explores personalized test-time optimizations that target both speed enhancement to reduce latency and quality improvement to increase pass rate.

B. Inference-Time Speed and Quality Enhancement

The scaling law of inference [†] has indicated its potential to improve the performance for most natural language tasks. In this work, we investigate their effectiveness in hardware design generation and propose customization to further enhance their flexibility and efficiency.

Customized Quality Improvement. Various test-time optimizations [41] can enhance output generation quality, with popular methods including Best-of-N sampling and temperature tuning. Since clients have diverse domains, such as classical or quantum, and their focus on designing different hardware architectures, the choice of optimization techniques can vary across different scenarios to maximize the generation quality. Moreover, these techniques introduce multiple hyperparameters, presenting a design space for optimization. To leverage this opportunity, our framework enables clients to customize and optimize their sampling strategy and hyperparameters to meet specific requirements, for example, allowing them to balance hardware design quality and generation latency by adjusting the sampling count. Table I presents the sampling strategies supported in our framework with their associated hyperparameters. To tailor the sampling strategy for each client, a grid search can be applied to optimize the sampling configurations.

TABLE I: Sampling Strategies and Hyperparameters

Strategy	Description	Hyperparameters
Nucleus Sampling	Selects tokens with cumulative probability p	p (cumulative probability)
Temperature Sampling	Scales token probabilities by temperature	Temperature, Number of generated candidates
Top-k Sampling	Chooses from the top k most probable tokens	k (number of sequences to consider)
Beam Search	Expands search using a fixed beam width	Beam width

Personalized Inference-Time Acceleration. Generating an optimized hardware design may require a large number of samples, resulting in high generation latency and energy costs. Since the performance of auto-regressive generation in LLM inference is typically memory-bound, several techniques have been introduced to leverage idle compute resources to accelerate LLM inference. Among these are speculative decoding [24], [34] and parallel decoding [3], [5], which generates multiple tokens in parallel to improve the processing speed. However, most existing approaches rely on a separate training process to learn the multi-token generation capability.

In this work, we propose an inference-time learning approach, where each client locally learns acceleration parameters during the model's deployment phase while serving real user requests. Specifically, we observe that the learning process of multi-token generation involves tuning the acceleration parameters to approximate the predictive distribution of the original model. Therefore, rather than depending on a training dataset, our approach utilizes the generation outputs produced during deployment, while serving user requests, for learning multi-token generation. This method offers two key benefits. First, by leveraging user-generated content as labels, the approach can be seen as an unsupervised learning technique, eliminating the need for extra datasets. Second, the learning process is performed during the model deployment time, avoiding a separate training process to learn multi-token generation. In this paper, we consider parallel decoding approaches as they are more training-efficient compared to other speculative decoding methods, making it suitable for online learning. The training objective is formulated as follows:

$$\arg\min_{\phi} \mathbb{E}_{x\sim\mathcal{D}} \left[\mathcal{KL}\big(P_{\text{a}}(\mathbf{y}_{t+1:t+k} \mid \mathbf{y}_{1:t}, x; \phi), P_{\text{o}}(\mathbf{y}_{t+1:t+k} \mid \mathbf{y}_{1:t}, x; \theta)\big)\right]$$

where ϕ are the acceleration parameters, and $\mathcal{D}$ is the deployment data distribution. The $\mathcal{KL}$-divergence measures the difference between the P_{a} distribution for acceleration and target P_{o} distributions. $\mathbf{y}_{t+1:t+k}$ represents the predicted token sequence, and $\mathbf{y}_{1:t}$ the previously generated tokens by target model with parameter θ.

VI. Experiments

A. Evaluation Setup

Models and Datasets For classical hardware experiments, we use CodeLlama-7B [39] as the base model and the HLS benchmark described in Sec. III-B. This benchmark contains machine-generated instructions (MachineEval) produced by GPT for HLS generation. To evaluate the model's generalizability, we involve human experts to manually refine 50 samples, creating a HumanEval version. For Qiskit quantum design generation, we use StarCoder2-3B [30] with the dataset introduced in Sec. III-B.

Federated Learning. For both classical and quantum benchmarks, 40 clients are trained with datasets partitioned using a Dirichlet distribution [23]. Each round involves training for one epoch with 10% of the clients participating. Two aggregation metrics were tested: the number of data samples (Ratio) and hardware syntax accuracy (Acc). A separate validation dataset was used to calculate syntax accuracy.

Model Merging. Hardware syntax accuracy on a validation dataset was used as the weight for model aggregation for both benchmarks. Client datasets were split using a Dirichlet distribution. DARE [52] was used for hierarchical model aggregation. In this setting, datasets were split in an IID manner across FL clients and clients that perform local training.

Personalized Test-Time Optimization. We adopt parallel decoding for inference-time acceleration. A validation dataset with hardware design instructions is used to simulate user requests. Different sampling strategies and the associated hyperparameters are summarized in Table I.

B. Effect of Hierarchical Approach

To evaluate the effectiveness of our proposed hierarchical approach, we conduct experiments on both classical and quantum benchmarks, as shown in Fig. 3. We compare our method against two baselines: the base model without fine-tuning and a

[†]https://openai.com/index/introducing-openai-o1-preview/

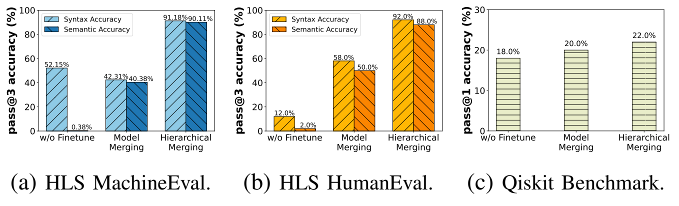
(a) HLS MachineEval. (b) HLS HumanEval. (c) Qiskit Benchmark.

Fig. 3: Effect of hierarchical approach on both classical and quantum hardware benchmarks.

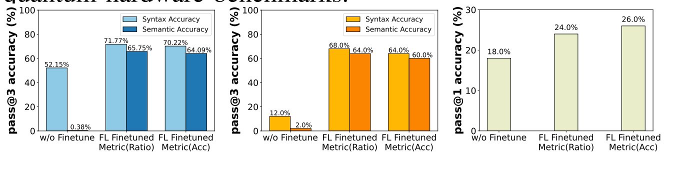
(a) HLS MachineEval. (b) HLS HumanEval. (c) Qiskit Benchmark.

Fig. 4: Evaluation of federated learning on both classical and quantum hardware benchmarks.

model merging without hierarchical aggregation. For classical hardware generated via HLS, we assess both syntax and semantic accuracy with template generation enhancement. For the quantum benchmark, we primarily focus on semantic accuracy evaluation. In both classical and quantum evaluations, our approach demonstrated accuracy improvement. As shown in Fig. 3a&3b, the hierarchical approach demonstrates significantly greater improvement for classical hardware generation tasks in both MachineEval and HumanEval, achieving nearly an 80% increase in syntax and semantic accuracy compared to the model without fine-tuning, and approximately 50% over the model obtained through model merging. However, the improvements in the Qiskit benchmark were less significant. This could attributed to the higher difficulty of quantum design generation, which can be further improved by incorporating higher-quality training datasets.

C. *Evaluation of Federated Learning*

Fig. 4 present the evaluation of federated learning on classical and quantum benchmarks using two different aggregation strategies. As shown in Fig. 4c, leveraging hardware syntax accuracy during model aggregation achieves the best result in the Qiskit Benchmark. For classical hardware generation as shown in Fig. 4a&4b, Both ratio-based and Acc-based approaches achieve similar results in MachineEval and HumanEval, with up to a 60% increase in semantic accuracy. These findings demonstrate the effectiveness and flexibility of federated learning with metric-based aggregation in training models for both classical and quantum hardware design generation.

D. *Personalized Inference-Time Optimizations*

As discussed in Sec. I, the lack of personalized optimizations restricts the potential for maximizing both speed and quality in model inference. Thus, we evaluate the impact of two test-time optimizations: multi-token generation for acceleration and customized sampling for quality improvement. We show that by tailoring multi-token generation configuration and sampling strategies to the client's compute budget, the optimization achieves 2.3 $\times$ speedup ratio and upto 46%

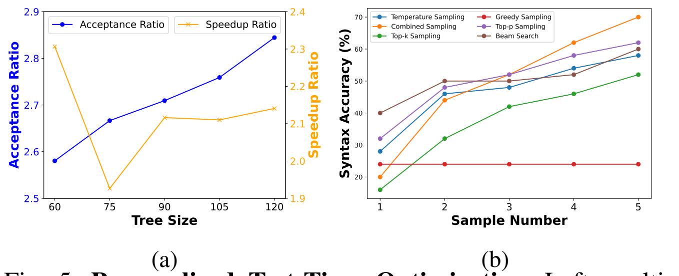
(a) (b)

Fig. 5: **Personalized Test-Time Optimization**. Left: multi-token generation. Right: Customized sampling for HLS models. Combined Sampling uses both top-k and top-p filtering.

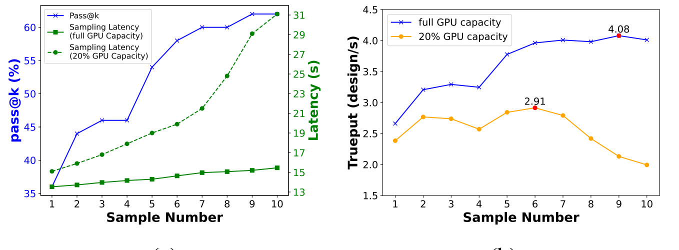
(a) (b)

Fig. 6: **Trueput Evaluation**. Left: Pass@k and latency for different sample sizes. Right: **Trueput** across GPU capacities.

syntax accuracy improvement over default greedy decoding. In Fig. 5a, we evaluate the speedup ratio with respect to the tree size, a parameter reflecting the token parallelism. While the acceptance ratio increases with larger tree sizes, the speedup ratio peaks at a tree size of 60 due to limited idle compute resources. This aligns with previous research [5], which shows that tree size must be optimized per client to achieve maximum speedup. In Fig. 5b, we examine how different sampling strategies affect syntax accuracy. Beam search performs best with sample sizes under 3, while combined sampling outperforms others for larger sample sizes. Our findings emphasize the need for personalized inference-time optimizations to advance AI-assisted hardware design.

E. *Effect of Sample Number on Trueput Optimization*

In Sec. V, we introduce **Trueput** to analyze the efficiency of design generation. To maximize $\textbf{Trueput}_{\text{batch}}$, in addition to the previously mentioned personalized test-time optimization, it is important to search for the optimal sample number k, as shown in (2). As depicted in Fig. 6a, both Pass@k and sampling latency increase with the sample number k, but with different rates. Fig. 6b shows that a local maximum of $\textbf{Trueput}_{\text{batch}}$ exists, and the optimal value of k varies depending on the GPU capacity. This observation highlights that the sample number should be optimized per client to achieve the maximum **Trueput**.

VII. Conclusion

Recent advancements in AI have shown great potential in revolutionizing the traditional hardware design process. However, the limited quality and quantity of available data remain critical barriers to the development of AI-assisted hardware design. In this paper, the authors argue that addressing this fundamental challenge requires decentralized and personalized learning approaches. To this end, we present a two-stage framework featuring a novel hierarchical decentralized training paradigm with metric-based model aggregation for model training, along with personalized inference-time optimizations to enhance deployment efficiency. Comprehensive evaluations on both classical and quantum hardware design tasks demonstrate the effectiveness of our approach. We hope that the benchmarking results of this work will encourage broader engagement from both industrial and individual parties in jointly advancing AI-assisted hardware design.

References

[1] Mehdi Salehi Heydar Abad et al. Hierarchical federated learning across heterogeneous cellular networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8866–8870. IEEE, 2020.

[2] Alon Albalak et al. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827*, 2024.

[3] Tianle Cai et al. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. *arXiv preprint arXiv:2401.10774*, 2024.

[4] Kaiyan Chang et al. Chipgpt: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019*, 2023.

[5] Hao Mark Chen et al. Hardware-aware parallel prompt decoding for memory-efficient acceleration of llm inference. *arXiv preprint arXiv:2405.18628*, 2024.

[6] Mark Chen et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[7] Nicolas Dupuis et al. Qiskit code assistant: Training llms for generating quantum computing code. *arXiv preprint arXiv:2405.19495*, 2024.

[8] Yanai Elazar et al. What's in my big data? In *The Twelfth International Conference on Learning Representations*, 2024.

[9] Weimin Fu et al. Llm4sechw: Leveraging domain-specific large language model for hardware debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2023.

[10] Wentao Gao et al. Federated learning as a service for hierarchical edge networks with heterogeneous models. *arXiv preprint arXiv:2407.20573*, 2024.

[11] Charles Goddard et al. Arcee's mergekit: A toolkit for merging large language models. *arXiv preprint arXiv:2403.13257*, 2024.

[12] István Hegedűs et al. Gossip learning as a decentralized alternative to federated learning. In *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings 19*, pages 74–90. Springer, 2019.

[13] Timothy Hospedales et al. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

[14] Nathaniel Hudson et al. Flight: A faas-based framework for complex and hierarchical federated learning. *arXiv preprint arXiv:2409.16495*, 2024.

[15] Gabriel Ilharco et al. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[16] Juyong Jiang et al. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[17] Haolin Jin et al. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479*, 2024.

[18] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

[19] Jakub Konečnỳ et al. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[20] Viraj Kulkarni et al. Survey of personalization techniques for federated learning. In *2020 fourth world conference on smart trends in systems, security and sustainability (WorldS4)*, pages 794–797. IEEE, 2020.

[21] Yongchan Kwon et al. Datainf: Efficiently estimating data influence in loRA-tuned LLMs and diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024.

[22] Royson Lee et al. Fedl2p: Federated learning to personalize. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Qinbin Li et al. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th international conference on data engineering (ICDE)*, pages 965–978. IEEE, 2022.

[24] Yuhui Li et al. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[25] Paul Pu Liang et al. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

[26] Yuchao Liao et al. Are llms any good for high-level synthesis? *arXiv preprint arXiv:2408.10428*, 2024.

[27] Hunter Lightman et al. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[28] Mingjie Liu et al. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2023.

[29] Shang Liu et al. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5. IEEE, 2024.

[30] Anton Lozhkov et al. Starcoder 2 and the stack v2: The next generation, 2024.

[31] Yao Lu et al. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 722–727. IEEE, 2024.

[32] Michael S Matena et al. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

[33] Brendan McMahan et al. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[34] Xupeng Miao et al. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[35] Mohammed Muqeeth et al. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745*, 2023.

[36] Dinh C Nguyen et al. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[37] Pian Qi et al. Model aggregation techniques in federated learning: A comprehensive survey. *Future Generation Computer Systems*, 150:272–293, 2024.

[38] Yuxiao Qu et al. Recursive introspection: Teaching language model agents how to self-improve. *arXiv preprint arXiv:2407.18219*, 2024.

[39] Baptiste Roziere et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[40] Reza Shokri et al. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[41] Charlie Snell et al. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[42] Canh T Dinh et al. Personalized federated learning with moreau envelopes. *Advances in neural information processing systems*, 33:21394–21405, 2020.

[43] Shailja Thakur et al. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[44] Yun-Da Tsai et al. Rtlfixer: Automatically fixing rtl syntax errors with large language models. *arXiv preprint arXiv:2311.16543*, 2023.

[45] Peiyi Wang et al. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024.

[46] T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[47] Chenwei Xiong et al. Hlspilot: Llm-based high-level synthesis. *arXiv preprint arXiv:2408.06810*, 2024.

[48] Haocheng Xu et al. Optimizing high-level synthesis designs with retrieval-augmented large language models. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5. IEEE, 2024.

[49] Zhiyuan Yan et al. Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms. *arXiv preprint arXiv:2402.00386*, 2024.

[50] Enneng Yang et al. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*, 2024.

[51] Mang Ye et al. Heterogeneous federated learning: State-of-the-art and research challenges. *ACM Computing Surveys*, 56(3):1–44, 2023.

[52] Le Yu et al. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

[53] Frederic Z Zhang et al. Knowledge composition using task vectors with learned anisotropic scaling. *arXiv preprint arXiv:2407.02880*, 2024.