InsightAlign: A Transferable Physical Design Recipe Recommender Based on Design Insights

Hao-Hsiang Hsiao[1], Sudipto Kundu[2], Wei Zeng[2], Wei-Ting J. Chan[3], Deyuan Guo[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA; [2]Synopsys Inc., Sunnyvale, CA; [3]Synopsys Inc., Hillsboro, OR
{thsiao, limsk}@gatech.edu; {Sudipto.Kundu, Wei.Zeng, Wei-Ting.Chan, Deyuan.Guo}@synopsys.com

2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11133146

***Abstract*—Physical design tools have complex workflows with many different ways of optimizing power, performance, and area (PPA) out of a large number of options and hyperparameters in different engines and functionalities. Black-box optimization techniques are widely adopted to automate quality-of-result (QoR) exploration. Such exploration often proves impractical in real-world customer environments due to high computational demands, lengthy exploration cycles, and the need for large parallel jobs. To reduce the exploration space for viable compute resource requirements, we propose a novel design methodology to enable transferable learning by incorporating design insights crafted on top of physical design experts' experience and streamlining QoR exploration as a sequence generation task for best recipe selection. We apply language model-inspired alignment techniques to learn the ranking of different recipe sets, enabling our model to generalize beyond known-good manually tuned expert design recipes. Extensive evaluations demonstrate our method's superior QoRs and runtime performance on unseen industrial designs and rigorous benchmarks.**

I. Introduction

Advancements in technology nodes boost the capabilities available to circuit designers but, on the other hand, introduce significant challenges at every stage of the physical design flow, driven by an explosion in design complexity. Commercial physical design tools address these challenges through extensive tunability. At each phase of the design process, these tools are meticulously calibrated to resolve specific design- or technology-related issues by balancing critical factors—such as congestion, cell density, and routing constraints—to optimize the quality of result (QoR) at flow signoff. However, as product requirements diversify, these tools are increasingly required to support a wider range of QoR intentions. This growing spectrum of tunable parameters can quickly overwhelm designers, making rapid and optimized design turnarounds ever more challenging.

Although recent works on design flow recipe recommendation have demonstrated progress in using modern optimization techniques to manage the rapidly-growing tool parameters, the search for systematic approaches to align automatic recipe recommenders and design experts' experiences remains open. To address this challenge, we developed a novel automatic flow recipe recommendation framework that comprehends physical design expert knowledge by encoding critical design flow health metrics—what we refer to as **"design insights"**—that go beyond traditional QoR measurements. These insights are the nuanced analyses that experienced designers perform while evaluating and debugging designs, encompassing both high-level flow analysis and detailed engine-level behavior to provide a holistic view of the design's flow trajectory. These expert analyses are systematically transformed into quantitative numerical labels, automatically generated during each place and route (P&R) run, which guide our framework into design-specific recommendations.

Building on insights that capture critical design behaviors, we frame the flow recipe recommendation problem as a sequence generation task. This approach enables us to leverage state-of-the-art large language model (LLM) alignment techniques—traditionally employed to align model outputs with human preferences—to align our model's recipe recommendations with QoR intentions in physical design flows: by incorporating design-specific insights and applying pairwise contrastive training on offline datasets, our model learns to rank recipe sets by their effectiveness, rather than simply memorizing high-performing configurations. This approach provides the model with a nuanced understanding of recipe effectiveness, empowering it to suggest unexplored recipes that align with QoR requirements tailored to each unique design. Additionally, this approach addresses the challenge of exact QoR prediction, which is particularly infeasible for industrial-scale designs due to the variability in design scales and QoR metrics. By focusing on relative effectiveness rather than absolute values, our model achieves robust, cross-design generalization in recipe recommendations. For the first time, LLM alignment techniques are applied to physical design flow recipe recommendation, and extensive experiments demonstrate optimal zero-shot recipe recommendation results, even for previously unseen designs. Our main contributions are as follows:

- We formalize the process of encoding expert knowledge-based health analyses of designs as quantitative labels, creating a comprehensive representation of design flow health metrics that extends beyond traditional QoR measurements. These labels serve as a foundation for informed recipe recommendations across diverse design traits and technology nodes.
- We propose a novel application of LLM alignment techniques to flow recipe recommendation, achieving alignment between recipe recommendations and QoR intentions of the physical design flow. Our approach leverages ranking-based learning to prioritize recipe sets based on relative effectiveness, thereby addressing the limitations of conventional supervised learning approaches that depend on memorizing previously observed recipes. Additionally, this method circumvents the challenge of exact QoR prediction, facilitating adaptable cross-design interoperability.
- This work embodies a tight integration of a flow recommender at an industrial scale, real-world designs and advanced technology nodes on top of an industrial physical design tool [1]. Extensive experimental evaluations demonstrate the model's superior zero-shot performance across unseen designs, highlighting its generalizability and effectiveness in practical deployment environments.

II. Background

Automating the tuning of Electronic Design Automation (EDA) flows has become essential for achieving optimal power, performance, and area (PPA) outcomes as design complexity increases. Due to the vast parameter space of physical design flows in advanced technology nodes, conventional design approach to sweep a limited set of key flow parameters is runtime infeasible and often leaves beneficial parameter combinations unexplored. Recent EDA research has thus focused on leveraging machine learning inspired optimization techniques, each with unique strengths and limitations, to automate this

process. This section summarizes several prominent approaches for automatic flow tuning.

- Bayesian Optimization (BO): BO [2]–[5] uses a Gaussian process surrogate model to predict PPA based on parameter settings, selecting configurations iteratively through an acquisition function. This method excels in high-cost evaluation settings but can be limited by the accuracy of its surrogate model, particularly in highly dynamic or noisy parameter spaces.
- Ant Colony Optimization (ACO): ACO [6] models parameter tuning as a graph search, with parameters represented as nodes and pheromone trails guiding configuration selection. While effective in escaping local optima, ACO may require extensive tuning of pheromone parameters and often converges slowly, making it less suitable for large-scale, high-dimensional searches.
- Recommendation-Based Optimization: Inspired by matrix factorization, this approach [7] assigns latent features to netlists and configurations, predicting QoR through feature similarity. While useful for suggesting high-quality configurations, this method can struggle in high-dimensional spaces and lacks domain-specific insights, potentially leading to suboptimal recommendations.
- Reinforcement Learning (RL)-Based Optimization: Reinforcement learning frameworks, such as that proposed by [8]–[10], iteratively refine expert-provided configurations to improve cumulative PPA rewards. Although effective for fine-tuning, this approach relies heavily on the initial configuration and is limited by episode length, making it less effective for discovering new configurations.

Existing use models of recipe recommender include (1) offline model trained on in-house design project archive, which may not generalize in new designs or technology nodes, and (2) online model that starts with an unexplored design and iteratively collects and trains itself as training data are accumulated through multiple iterations of a design flow, which could consume huge compute resources. Several previous works identified the respective challenges of generality and resource hungriness and suggested building transferable recipe recommender from offline design project archive to enable the kick-start of online learning [3], [4], [6], [9], [11]. As pointed out by [8] and [12], the observability of physical design flow health is crucial to allow recipe recommender to discover design similarity and achieve transferability. However, previous works' flow health metrics are limited to intuitive statistics of design characteristics and categorized QoR metrics. [8] suggests metrics dedicated to only cell placement. [12] and [13] suggest using static snapshots of QoR or resource metrics at each flow stage, which do not capture the natural metric fluctuations at different flow stages. [9] proposes building a graph neural network (GNN) to embed design characteristics for a transferable recommender. However, it does not address the complexity of tool behaviors under different QoR objectives.

III. Methodologies

The goal of our flow recipe recommendation framework, as depicted in Figure 1, is to select the optimal subset of recipes to meet multiple user-defined design objectives. Given a set of preconfigured recipes, the task is to identify the best subset of recipes that will yield superior QoR according to user intentions.

A. Problem Formulation

InsightAlign contains the following key components:

a) Design insights: Contextual insights from the prior run learning contain fine-grained real-time analysis of the complex workflow of design implementation flow. Typically, physical design tool users observe the final metrics in terms of timing (worst negative slack (WNS) and total negative slack (TNS)), area, power, routing design rule check (DRC) violations, etc. Conventional black-box optimization around the tool explores solution space based on the final metrics in an iterative loop. In our recommendation framework formulation, one of the key components is the ability to consume deep insights into the design, as well as the flow that is learned from the first run of the design. Typical examples of such insights are shown in Table I. The tool natively discovers timing, area, and power challenges in the form of insights. Our objective is to drive the recommendation framework with deep insights as an input, so that it overcomes the limitation of black-box optimization. Each time the P&R flow iteration is executed, the deep insights are collected and fed to the recommendation framework to suggest more relevant recipes that will provide faster convergence to the best PPA.

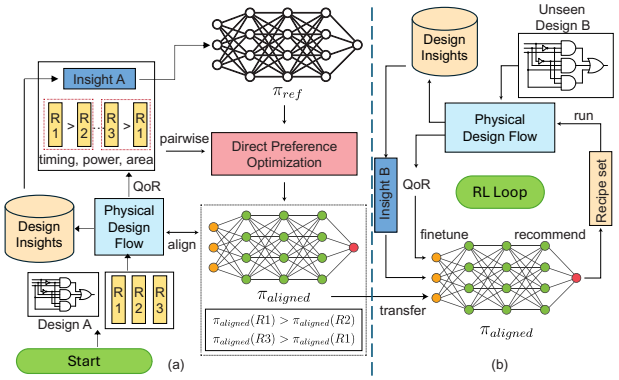

Fig. 1. Overview of InsightAlign: (a) offline alignment, (b) online fine-tuning.

TABLE I
Examples of Insights

Category	Insight Description	Range
Placement	Congestion level during placement step X	{low, medium, high}
Timing	Is easy to meet timing constraints	{yes, no}
Power	Good opportunity for power saving during step Y	{yes, no}
Power	Sequential-cell power is dominant	{yes, no}
Power	Leakage power is dominant	{yes, no}
Clock	Critical paths with harmful clock skew	{yes, no}
Timing	Instance count from hold-time fixes	$\mathbb{N}$
Timing	Weak cell percentage on critical paths	$\mathbb{R} \in [0.0, 100.0]$

TABLE II
Examples of Recipes

Category	Recipe Description
Design intention tradeoffs	Adjust tradeoffs among timing, power, and area metrics
Timing	Balance weights of early hold- and setup-time fixing, and placement perturbations
Clock tree	Adjust clock-tree synthesis (CTS) hyperparameters for tradeoffs among timing, skew and latency
Routing	Adjust knobs of routing congestion
Routing	Adjust global routing hyperparameters

b) Recipe: Commercial physical design tools usually offer thousands of flow parameters and options, creating an expansive search space that is infeasible to explore exhaustively. To manage this, we prune the space by defining a suite of preconfigured options, referred to as "recipes." Consisting of a set of flow parameters and options, each recipe serves a specific QoR intention. Example recipes are outlined in Table II. While each individual recipe has dedicated intention, we aim to capture the complex interactions among these

TABLE III
DETAILS OF INSIGHTALIGN MODEL ARCHITECTURE AND DIMENSIONS

Layer	Type	Input Size	Output Size
Decision Token Embed.	Embedding	$(40, 3)$	$(40, 32)$
Recipe Pos. Enc.	Positional Encoding	$(40, 32)$	$(40, 32)$
Insight Embed.	Linear $\times 1$	$(1, 72)$	$(1, 32)$
Transformer Dec.	Transformer Decoder $\times 1$	$(1, 32)$ $(40, 32)$	$(40, 1)$
Probabilistic	Sigmoid $\times 40$	$(40, 1)$	$(40, 1)$

recipes, and to combine recipes strategically to achieve multiple design objectives simultaneously.

B. Stage-Based Flow Recipe Recommender

From initial cell placement to design closure, each stage in physical design flow deals with different physical design problems and usually provides diverse parameters. Our framework provides separate recipe sets for each flow stage to fully leverage the parameter diversities, as shown in Figure 2. Note that our staged framework is not specific to a given stage definition and can be generalized to any flow variation. We refer to each run of physical design flow loaded with proposed recipe sets as an **iteration**. Each iteration can include up to N recipe sets, determined by the available computational resources.

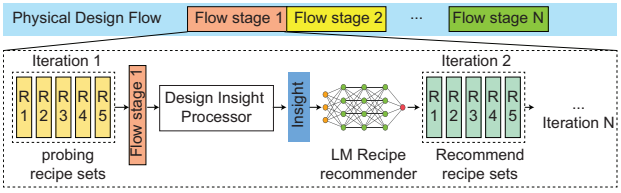

Fig. 2. InsightAlign incorporated into the physical design flow. Each "Rx" represents a recipe set, which consists of a subset of preconfigured recipes.

Our recommender intelligently consumes the first iteration, serving as an insight-probing phase, to generate initial insights. We refer to this insight-probing phase as "**offline alignment**." During offline alignment, our design insight analyzers, imitating human experts' flow probing process, perform analysis in various algorithm engines, monitor flow trajectory and accumulate insights in a non-volatile storage. After the first iteration, subsequent iterations leverage these accumulated insights and prior QoR results to iteratively refine the recipe sets, which we refer to "**online fine-tuning**." With each new iteration, additional insights are gathered, providing a progressively generalized view of the design. As more recipe sets are explored through successive iterations, the recommender becomes increasingly tailored, resulting in optimized QoR outcomes.

C. Language Model-Based Recipe Recommendation

Our recipe recommender is based on a decoder-only generative language model architecture, where recipes are treated as tokens and predicted sequentially in an autoregressive manner, as shown in Figure 3. The recipes are evaluated one by one at each time step t, and the model estimates the probability of selecting each recipe in the recommended subset based on the preceding context. This autoregressive process enables the model to construct recipe subsets progressively, adapting its choices to the design insights and previously chosen recipes. Our model includes the following components with details in Table III:

1) Decision Token Embedding: Each recipe decision r_t indicates whether a recipe is selected or not. We train separate embeddings for both decision, allowing the model to learn nuanced representations of each decision outcome in the context of recipe selection, i.e., "selected" (1), "not selected" (0), and an additional "SOS" (start-of-sequence) token for initiating the generation process.
2) Recipe Positional Embedding: Positional encoding is essential in language models to convey the order of tokens, and here it is used to represent the order of recipes in the tuning sequence. The decision token embedding is combined with a recipe-specific positional encoding, which provides necessary information such that the model can distinguish among recipes based on both their unique characteristics and their positions in the sequence.
3) Insight Embedding: Design insights are concatenated into a vector and processed by a linear layer to produce a high-dimensional embedding that will be further served as a contextual information for design specific recipe recommendation.
4) Transformer Decoder Layer: The model uses a single-head Transformer decoder layer [14] that leverages contextual information through cross attention between recipe and design insight embeddings. This architecture enables the model to learn interdependencies between recipe decisions to adaptively tune recipes based on previous decisions and design insights.
5) Probabilistic Layer: Finally, after the decoder layer generates a contextualized representation of each recipe, the model converts this representation into a probability for each recipe using a sigmoid activation function. This probability indicates the likelihood of selecting each recipe, enabling the model to make probabilistic, informed recommendations based on the embeddings and contextual analysis performed in prior layers.

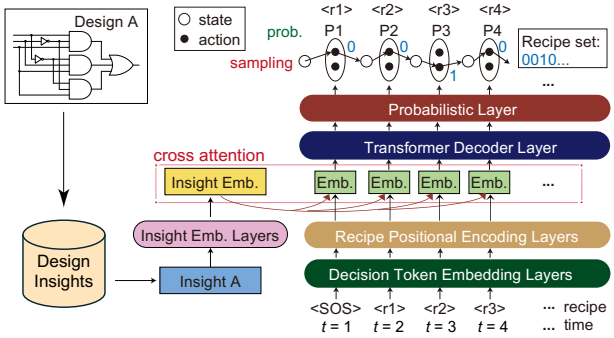

Fig. 3. InsightAlign model architecture for recipe set recommendation. The same model is used in offline alignment and online fine-tuning phases.

D. Model Training Overview

Ideally, tuning would be fully "online," using the QoR of each recipe set to directly guide model updates and the selection of subsequent recipes. However, the extended runtime of the commercial physical design tool—which can take days to weeks for industrial-scale designs—makes continuous online updates impractical. To address this, we adopt a two-stage training strategy.

Our model training involves two phases: (1) Offline alignment and (2) Online fine-tuning. The offline alignment process is detailed in Algorithm 1. Our framework performs offline preference-based alignment by training a language model to assign higher probabilities to recipes that historically achieved better QoR outcomes. For each design, all known recipe sets are pairwise compared based on their QoR values, and the model is optimized using margin-based direct preference optimization (DPO) [15] to reflect these preferences by assigning higher probabilities to recipe sets with better QoRs. During inference, the aligned model employs beam search with width K to generate the top-K recommended recipe sets for an unseen design

by keeping track of the K most promising partial sequences at each step. After offline alignment, the model enters the online fine-tuning phase, where it is further refined for specific designs. Here, the model continuously evolves based on direct feedback from the physical design tool, using immediate QoR evaluations of newly recommended recipe sets to make real-time adjustments. This online learning phase allows the model to adapt dynamically, providing fine-tuned recipe recommendations tailored to each design's unique characteristics.

Algorithm 1 Offline QoR-Alignment

Input:
Dataset $\mathcal{D} = \{(I^i, R^i, Q^i)\}_{i=1}^N$, where design insight $I^i \in \mathcal{I}$, recipe set $R^i \in \{0,1\}^n$, compound QoR score $Q^i \in \mathbb{R}$
Beam width K, Insight of a new design $I' \in \mathcal{I}$
Output: Policy π_ϕ and top-K recipe recommendations $\{\hat{R}_k\}_{k=1}^K$
function AlignmentTrain($\mathcal{D}$)
  Initialize policy parameters ϕ
  while *not converged* **do**
    for $I^k \in \mathcal{D}$ **do**
      $\mathcal{P}_k = \{(R^i, R^j) : I^i = I^j = I^k\}$ ▷ All recipe pairs for design k
      for $(R^i, R^j) \in \mathcal{P}_k$ **do**
        $(R_w, R_l) \leftarrow (R^i, R^j)$ if $Q^i > Q^j$ else (R^j, R^i)
        $\log \pi_\phi(R \mid I^k) = \sum_{t=1}^n \log P(r_t \mid r_{<t}, I^k; \phi)$ for $R \in \{R_w, R_l\}$
        Evaluate $\mathcal{L}_{\text{MDPO}}(\phi)$ with (2)
        $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_{\text{MDPO}}(\phi)$ ▷ Gradient descend
      end for
    end for
  end while
  return π_ϕ
end function
function BeamSearch(π_ϕ, I', K)
  $\mathcal{B}_0 \leftarrow \{([\text{SOS}], 0)\}$ ▷ Initial beam: (sequence, log probability)
  for $t = 1$ to n **do**
    $\mathcal{B}_t \leftarrow \emptyset$
    for $(R_{<t}, s) \in \mathcal{B}_{t-1}$ **do**
      $P_t \leftarrow \pi_\phi(\cdot \mid R_{<t}, I')$
      for $r_t \in \{0,1\}$ **do**
        $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{(R_{<t} \oplus r_t, s + \log P_t(r_t))\}$
      end for
    end for
    $\mathcal{B}_t \leftarrow$ top-K sequences from $\mathcal{B}_t$ by score
  end for
  return $\{R : (R, s) \in \mathcal{B}_n\}$ ▷ Return top-K sequences
end function
$\pi_\phi \leftarrow$ AlignmentTrain($\mathcal{D}$)
$\{\hat{R}_k\}_{k=1}^K \leftarrow$ BeamSearch(π_ϕ, I', K)
return $\pi_\phi, \{\hat{R}_k\}_{k=1}^K$

E. Offline Alignment

Our recommendation framework leverages alignment techniques inspired by large language model (LLM) [14] to predict optimal recipe sets that enhance QoR for complex industrial designs. Instead of relying on direct QoR prediction—which is challenging due to the variability in design scales and inherent complexity—our approach employs a preference-based strategy. This allows the model to make effective recipe recommendations by comparing relative performance, even when absolute QoR values are difficult to estimate. This reframing allows the model to focus on the relative performance among recipe sets, rather than attempting to predict absolute QoR values. The overall process is illustrated in Figure 1(a).

1) Offline Data Collection and Preference-Based Training: We first constructed an offline dataset using the commericial P&R tool. The dataset contains multiple (design, recipe set, QoR) combinations. Each data point is composed of (1) a design in the form of the insight vector with numerical and categorical data describing the contextual design knowledge, (2) a set of recipes selected for this run, and (3) the QoR outcome after applying that recipe set.

To align the model's recipe predictions with actual QoR-based preferences, we perform pairwise probability updates strategy. The process starts by pairing the data points from the same design. That is, for each design (insight), we compare pairs of recipe sets with known QoR outcomes, querying the model the probability of generating each recipe set, and then train the model to favor (modify model weights to assign higher probability to) recipes that contribute to better performance. This alignment process is iterated through all the pairs and all the designs. Each pairwise comparison guides the model to prioritize configurations that historically produce higher-quality results, ultimately tuning the model to reflect the QoR preferences.

2) Marginal Direct Preference Optimization: Traditional LLM alignment through Reinforcement Learning from Human Feedback (RLHF) [16] employs a two-stage approach: first training a reward model to evaluate outputs based on human-defined quality metrics, then utilizing proximal policy optimization (PPO) [17] to iteratively optimize a policy network. In contrast, DPO streamlines this process into a single-stage training paradigm by introducing an optimized loss function, eliminating the need for a separate reward model. The DPO loss $\mathcal{L}_{\text{DPO}}(\phi)$ is given by

$$-\mathbb{E}_{(R_w, R_l, I)\sim\mathcal{D}}\Big[\log\sigma\Big(\beta\log\frac{\pi_\phi(R_w \mid I)}{\pi_{\text{ref}}(R_w \mid I)} - \beta\log\frac{\pi_\phi(R_l \mid I)}{\pi_{\text{ref}}(R_l \mid I)}\Big)\Big] \tag{1}$$

Here, R_w and R_l denote the preferred ("winning") and non-preferred ("losing") recipe set in pairwise comparisons under design insight I. The probability of outputting R given design insight I is represented by $\pi_\phi(R \mid I)$ under model parameters ϕ, while $\pi_{\text{ref}}(R \mid I)$ denotes the reference policy probability. The preference model's sharpness is controlled by hyperparameter $\beta > 0$, and $\sigma(\cdot)$ represents the sigmoid function. While standard DPO effectively learns recipe rankings, it does not account for preference magnitude differences. A margin-based modification [18] addresses this by assuming policy preferences scale exponentially with recipe QoR scores and adopting a uniform reference policy $\pi_{\text{ref}}(R \mid I)$ to balance exploration and exploitation. The margin-based DPO loss is given by

$$\mathcal{L}_{\text{MDPO}}(\phi) = \mathbb{E}_{(R_i, R_j, I)\sim\mathcal{D}}\Bigg[\max\Big(0, \lambda(\text{QoR}(R_i) - \text{QoR}(R_j)) - \text{sign}(\text{QoR}(R_i) - \text{QoR}(R_j)) \cdot (\log\pi_\phi(R_i \mid I) - \log\pi_\phi(R_j \mid I))\Big)\Bigg] \tag{2}$$

3) Evaluate Probabilities of Recipe Selections: To compute the margin-based DPO loss in (2), we should be able to efficiently compute $\pi_\phi(R_i)$, the likelihood that each recipe set R_i will be predicted by our recipe model. The decoder-only language model is particularly well-suited for this task because it allows us to efficiently compute the likelihood of any recipe set by leveraging teacher forcing [19] in the training process. This setup is illustrated in Figure 4.

In the teacher forcing process, we input each recipe step by step as ground truth to guide the model's next prediction. Specifically, a recipe set, represented as a sequence $R = [r_1, r_2, \ldots, r_n]$ (with "1" indicating a recipe is selected and "0" indicating it is not), serves as

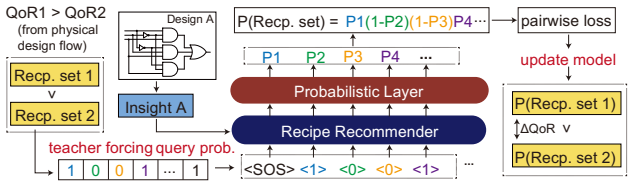

Fig. 4. The process of teacher forcing used in offline alignment.

input. At each time step $t = 1, \ldots, n$, the model receives the actual recipe r_t, which enables it to compute the conditional probability:

$$\log \pi_\phi(R \mid I^k) = \sum_{t=1}^{n} \log \pi_\phi(r_t \mid r_{<t}, I^k; \phi) \tag{3}$$

In this manner, the model can efficiently evaluate the probability of each recipe set and hence the pairwise margin-based DPO loss (2).

F. Recipe Set Recommendation

To recommend recipe sets, we employ beam search [19] with width K to identify the top-K recipe sets following our trained policy distribution. At each time step t, given an unseen design insight I' in the insight space $\mathcal{I}$, the algorithm maintains K partial sequences and extends each by sampling according to the policy probabilities $\pi_\phi(r_t \mid r_{<t}, I'; \phi)$, shown as "sampling" in Figure 3. The search accumulates log probabilities of selected recipes, retaining the K sequences with highest cumulative scores. This process continues until all recipe decisions are made, yielding K complete recipe sets that best align with our QoR-optimized policy distribution.

G. Online Fine-Tuning

After the offline alignment phase, the same model transitions into an online fine-tuning stage to further optimize its recipe recommendations for specific designs, as illustrated in Figure 1(b). Based on the offline learned policy, the model continuously adapts its policy π_ϕ through real-time feedback from the physical design tool. Each iteration follows a closed-loop learning system: the model proposes recipe sets, executes the physical design flow, and updates its parameters based on observed QoR metrics. This adaptive mechanism enables the model to capture design-specific optimization patterns that may not have been present in the offline training data, enhancing its ability to generate highly specialized recipes that better align with each design's unique characteristics. We implement batch online learning where the model proposes $K = 5$ recipes in each iteration and leverages both margin-based DPO [18] and PPO loss [17] to fine-tune the model.

IV. Experimental Results

A. Experiment Setup

We implemented our proposed recipe recommendation framework using *Python* and the deep learning library *PyTorch* [20]. To validate the effectiveness of our framework, we conducted experiments across 17 industrial-scale, real-world benchmarks. These benchmarks span a diverse range of design categories and advanced technology nodes, from 45 nm to sub-10 nm processes with gate counts up to 2 million, to ensure comprehensive evaluation under realistic conditions. Our optimization framework integrates $n = 40$ distinct recipes, each calibrated for distinct design objectives. The primary goal of the framework is to identify an optimal subset of recipes that collectively maximize a user-defined QoR intention as a compound score s based on multiple performance metrics m_i weighted by w_i, i.e.,

$$s = \sum_i w_i g_i (m_i - \text{mean}(m)_i)/\text{std}(m)_i \tag{4}$$

where $\text{mean}(m)_i$ and $\text{std}(m)_i$ are the mean and the standard deviation, respectively, of the ith metric over all datapoints of the same design, $g_i = 1$ if metric i is to be maximized, and -1 otherwise.

For illustration purposes, in this paper we define the QoR intention as minimizing the total power and TNS, with weights 0.7 and 0.3, respectively. In Algorithm 1, we use hyperparameter $\lambda = 2$ and beam width $K = 5$ in offline alignment and online fine-tuning.

We design different experiments to evaluate the efficacy of each phase. For the offline alignment phase, we construct a dataset $\mathcal{D}$ using a commercial physical design tool [1], comprising 3,000 datapoints collected from 17 designs with various recipe combinations. For the online fine-tuning phase, we iterate the recommendation and fine-tuning process for two designs on top of different offline alignment outcomes, and show how their QoRs are further improved and converged throughout iterations.

B. Offline Alignment Result

One of the key objectives of our work is to enable the framework to generalize optimization strategies to previously unseen designs without requiring retraining. To evaluate the model's transferability, we conducted zero-shot evaluations—meaning we tested the model on entirely new designs without any additional training or fine-tuning—on designs excluded from the training phase. This evaluation tests the model's ability to apply learned knowledge to new designs.

To rigorously evaluate the effectiveness of the offline phase, we employed a k-fold cross-validation strategy [21]. We divide the 17 designs into $k = 4$ random groups with roughly equal numbers of datapoints and carry out k iterations of evaluation. In iteration i, all designs in the ith group are held out for testing, and only the remaining designs are used for training and tuning the model. (As a result, we use different trained models to test designs in different groups to ensure all designs are unseen when evaluated.) By systematically rotating the combinations of training and testing splits, this approach ensures that all designs are evaluated once as unseen by the corresponding model. Therefore, the performance of the framework is assessed across the entire design spectrum.

Table IV summarizes the results of the zero-shot evaluation with cross-validation. For each design, we take five distinct recommendations from the model and compare the QoR in terms of power, TNS, and the compound score against the best-known datapoints of the same design in our offline dataset. (Recall that with cross-validation, the model never sees any datapoint from the design being tested.) The "Win%" column indicates the percentage of known recipe sets (out of approximately 200) that are outperformed by the best of five recommended recipe sets. It shows that the framework demonstrated performance comparable to, or even exceeding, the best-known recipe sets, without any fine-tuning.

Figure 5 visualizes the power-timing distribution of our zero-shot recommendations for four unseen designs: D4, D6, D11, and D14, compared to the QoR distribution of known recipe sets (unseen by the model). The recommended recipe sets are concentrated in the lower-left region of the plot, indicating superior performance with reduced power consumption and negative slacks. In contrast, the known recipe sets from the offline dataset exhibit a scattered distribution, reflecting the significant cost and effort required for extensive exploration, as opposed to our approach, which achieves optimal or near-optimal results at the very beginning of the design iteration.

These observations underscore the effectiveness of the design insight representation in capturing transferable knowledge, enabling the direct application of learned optimization strategies across designs. Additionally, they confirm the ability of our framework to capture the

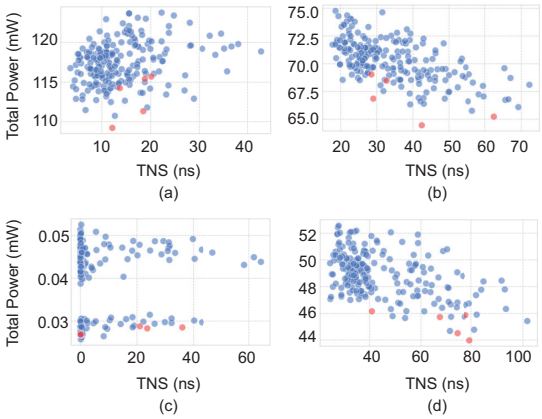

Fig. 5. QoRs of zero-shot recommended (red) compared with all known recipe sets in the dataset (blue) for designs (a) D4, (b) D6, (c) D11, (d) D14.

TABLE IV
EVALUATION OF OFFLINE ALIGNMENT ON UNSEEN DESIGNS BASED ON CROSS VALIDATION. "QOR SCORE" IS OUR OPTIMIZATION OBJECTIVE DEFINED BY (4). "WIN%" INDICATES THE PERCENTAGE OF KNOWN RECIPE SETS THAT OUR BEST RECOMMENDATION SURPASSES.

Design	Best known recipe set			Offline alignment			
	TNS (ns)	Power (mW)	QoR Score	TNS (ns)	Power (mW)	QoR Score	Win%
D1	20.23	1513.39	1.94	16.71	1539.99	1.42	98.7
D2	25.14	1177.70	2.12	55.20	1192.72	1.42	96.9
D3	18.28	2054.68	1.60	27.73	2046.93	1.70	100.0
D4	6.62	111.48	1.96	12.11	109.23	2.34	100.0
D5	0.0552	192.12	1.38	0.0823	191.56	1.49	100.0
D6	39.24	66.34	1.45	42.45	64.46	2.11	100.0
D7	18.12	143.17	1.40	14.47	144.00	1.44	100.0
D8	3.88	75.78	1.28	1.43	75.29	1.90	100.0
D9	0.816	640.01	1.98	0.824	614.90	2.65	100.0
D10	72.24	12.21	1.75	235.98	12.71	0.74	88.5
D11	0.157	0.0257	1.50	0.115	0.0268	1.40	98.1
D12	0.230	403.79	1.53	0.931	399.71	1.61	100.0
D13	118.70	311.99	3.54	379.85	338.46	0.56	95.2
D14	71.24	44.68	1.41	79.39	43.97	1.57	100.0
D15	0.0354	642.84	1.59	0.0838	640.65	1.68	100.0
D16	0.0000	0.706	1.66	0.0012	0.708	1.56	98.6
D17	282.06	688.48	1.11	793.51	677.32	0.93	97.8

complex interaction among recipes, recommend undiscovered, good-quality combinations of recipes, and provide excellent starting points of design iterations for previously unseen designs.

C. Online Fine-Tuning Result

Online fine-tuning provides an additional option to further tailor recipes for specific designs, enabling the achievement of optimal QoRs. Building on top of the offline alignment model, we show the online fine-tuning result on designs D10 and D6. As shown in Table IV, D10 is a design with slightly worse zero-shot recommendation compared to other benchmarks. Design D6, on the contrary, starts from a superior recommendation at the very beginning.

For design D10, our online reinforcement learning progressively refines the strategy for this specific design, allowing the model to recommend recipes that yield increasingly higher QoR scores over iterations. Figure 6(a) illustrates the average QoR score of the top five recipes encountered so far across iterations, highlighting the model's ability to continuously discover better-performing recipes. Additionally, Figure 7 visualizes the progressive QoR distribution over online learning. Notably, by leveraging the offline alignment knowledge, our model begins with a strong initial point. Over successive iterations,

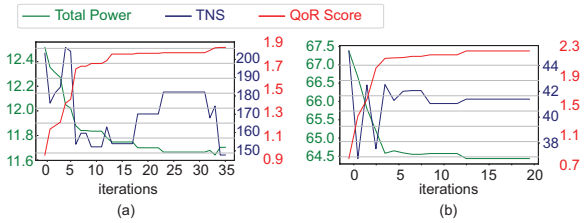

Fig. 6. Online fine-tuning trajectory: total power (lower-better), TNS (lower-better), and QoR score (higher-better) per iteration for (a) D10 and (b) D6.

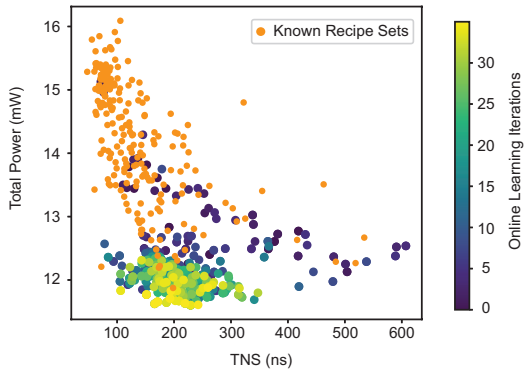

Fig. 7. Scatter plot of QoR for D10 in online fine-tuning phase. Early iteration points (darker colored) scattered on upper-right quickly move to lower-left (lighter colored) and converge to a better QoR than all known recipe sets.

the recommended recipes transition from the upper-right to the lower-left region of the plot and after a small number of iterations, surpassed all best-known recipe sets despite its suboptimal starting points. This demonstrates the continuous enhancement and provides designers with the flexibility to balance further QoR improvements against the cost of additional design iterations. For design D6, as shown in Figure 6(b), the online fine-tuning achieves even better QoR results than the zero-shot recommendation, and converges within even fewer iterations than D10, thanks to its superior starting points from offline alignment. These observations show that a good offline alignment can significantly reduce the number and hours of design iterations. With design-specific insights and user intention built in our framework, the online fine-tuning can achieve better QoR tailored to the specific design and different user intentions on top of the offline stage.

V. CONCLUSION

This work introduces a novel flow recipe recommendation approach that enables transferable learning by combining expert design insights with LLM alignment technique. Once the offline alignment model is established, online fine-tuning by running the physical design tool can achieve faster PPA closure by connecting recipe relevance with newly generated insights and QoRs. The combination of all these techniques addresses the challenge of high turnaround time and large compute demand, which is the most critical bottleneck of automatic flow tuning using expensive black-box tool evaluations. Extensive experiments on industrial-scale designs and advanced technology nodes demonstrate superior zero-shot PPA performance and fast-converging design iterations across unseen designs.

VI. ACKNOWLEDGEMENTS

We would like to express our gratitude to Vishal Khandelwal and Ravishankar Rao from Synopsys, Inc. for their support and valuable discussions.

REFERENCES

[1] Synopsys, Inc., "Fusion Compiler." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html

[2] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via Bayesian optimization," in *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*, 2019, pp. 1–6.

[3] H. Geng, Q. Xu, T.-Y. Ho, and B. Yu, "PPATuner: Pareto-driven tool parameter auto-tuning in physical design via Gaussian process transfer learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1237–1242.

[4] Z. Zhang, T. Chen, J. Huang, and M. Zhang, "A fast parameter tuning framework via transfer learning and multi-objective Bayesian optimization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 133–138.

[5] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, "PTPT: Physical design tool parameter tuning via multi-objective Bayesian optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 178–189, 2023.

[6] R. Liang, J. Jung, H. Xiang, L. Reddy, A. Lvov, J. Hu *et al.*, "FlowTuner: A multi-stage EDA flow tuner exploiting parameter knowledge transfer," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[7] M. Kazda, M. Monkowski, and G. Antony, "APEX: Recommending design flow parameters using a variational autoencoder," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.

[8] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[9] 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*, 2024, pp. 93–101.

[10] D. Luo, Q. Sun, Q. Xu, T. Chen, and H. Geng, "Attention-based EDA tool parameter explorer: From hybrid parameters to multi-QoR metrics," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[11] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany *et al.*, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 19–25.

[12] J. Jung, A. B. Kahng, S. Kim, and R. Varadarajan, "METRICS2.1 and flow tuning in the IEEE CEDA robust design flow and OpenROAD ICCAD special session paper," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[13] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "LAMDA: Learning-assisted multi-stage autotuning for FPGA design closure," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 74–77.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1–11.

[15] 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, 2024.

[16] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei *et al.*, "Fine-tuning language models from human preferences," 2020, *arXiv:1909.08593*.

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

[18] J. Wu, X. Wang, Z. Yang, J. Wu, J. Gao, B. Ding *et al.*, "α-DPO: Adaptive reward margin is what direct preference optimization needs," 2024, *arXiv:2410.10148*.

[19] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016, *arXiv:1610.09038*.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 1–12.

[21] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.