See it to Place it: Evolving Macro Placements with Vision-Language Models

Ikechukwu Uchendu[1,2,*] **Swati Goel**[1] **Karly Hou**[1] **Ebrahim Songhori**[3,†]

Kuang-Huei Lee[3] **Joe Wenjie Jiang**[3,†] **Vijay Janapa Reddi**[1] **Vincent Zhuang**[3,†]

Abstract

We propose using Vision-Language Models (VLMs) for macro placement in chip floorplanning, a complex optimization task that has recently shown promising advancements through machine learning methods. Because human designers rely heavily on spatial reasoning to arrange components on the chip canvas, we hypothesize that VLMs with strong spatial reasoning capabilities can effectively complement existing placement algorithms. We introduce VeoPlace (Visual Evolutionary Optimization Placement), a novel framework that uses a VLM—without any fine-tuning—to guide the actions of a base placer by constraining it to subregions of the chip canvas. The VLM proposals are iteratively optimized through an evolutionary search strategy with respect to resulting placement quality. Using global half-perimeter wirelength (gHPWL) after standard-cell placement and legalization as the evaluation metric, VeoPlace boosts ChiPFormer performance on 9 of 10 benchmarks with peak reductions exceeding 32%. We further demonstrate that VeoPlace generalizes to analytical placers, improving DREAMPlace on all 8 evaluated Superblue benchmarks with gains up to 4.4%. Our approach opens new possibilities for electronic design automation tools that leverage foundation models to solve complex physical design problems.

1 Introduction

Computer chip floorplanning is a critical step in the integrated circuit design process, involving the strategic arrangement of macros on the chip canvas. Determining the optimal placement is a complex multi-objective problem, in which performance, power, and area (PPA) must be optimized while minimizing routing congestion. The vast combinatorial design space makes manual chip floorplanning a time-consuming and expertise-driven task while posing a significant challenge for automated methods.

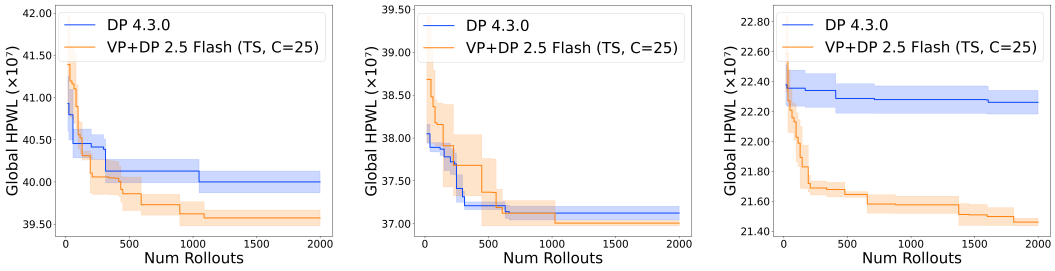
A variety of approaches have been proposed for automated chip floorplanning, including black-box optimization [37], analytical methods [25, 8, 27], and learning-based methods [28, 19, 20, 23]. Among these, learning-based approaches have achieved state-of-the-art performance, but have a severe limitation: policies trained from scratch struggle to generalize to *unseen chips* without additional interaction, an issue exacerbated by the limited training data available in chip design. In contrast, human designers leverage high-level prior knowledge and spatial reasoning to efficiently tackle new design spaces. Our work aims to bridge this gap by harnessing Vision-Language Models (VLMs) to provide spatial reasoning and guide the exploration of existing placement algorithms. While human

[1]Harvard University.
[2]Kempner Institute, Harvard University.
[3]Google DeepMind.
*Work done partly as a Student Researcher at Google. Correspondence: iuchendu1@gmail.com.
†Work done while at Google DeepMind.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

designers develop placement intuition over years of experience, VLMs can condition on dozens of prior floorplans and their evaluation metrics in a single prompt, allowing them to extract visual placement patterns and refine proposals through in-context learning.

Existing learning-based approaches first pre-train models on a set of training chips, then fine-tune on sampled placements from unseen chips [28, 19, 20]. We consider a general formulation of this setup: for an unseen chip and a fixed budget of B placement attempts, what is the best possible placement that can be generated? We posit that efficiently using this budget of online attempts requires spatial reasoning as well as learning from prior attempts, both of which have been exhibited by modern VLMs.



(a) `superblue5` (b) `superblue16` (c) `superblue18`

Figure 1: VeoPlace-guided DREAMPlace on (a) `superblue5`, (b) `superblue16`, and (c) `superblue18`. VP+DP denotes VeoPlace with DREAMPlace; DP 4.3.0 is standalone DREAMPlace. We report mean and standard error of the best gHPWL observed ($\times 10^7$, lower is better) across three random seeds with 2,000 rollouts each, using Gemini 2.5 Flash. The continued decrease in gHPWL demonstrates how VeoPlace improves placement quality over the fixed rollout budget, whereas standalone DREAMPlace converges early.

We introduce VeoPlace (Visual Evolutionary Optimization Placement), a novel framework that uses a high-level VLM planner to guide a low-level placer by constraining it to promising regions. Crucially, these VLM proposals are iteratively refined through an evolutionary process, as visualized in Figure 2. VeoPlace requires no fine-tuning of the VLM (we use the public Gemini models [41]) and uses an independent low-level placer (e.g. ChiPFormer [20] or DREAMPlace [25]). On open-source benchmarks (ISPD 2005 [30], ICCAD 2004 [1, 2], and Ariane [33]), VeoPlace boosts ChiPFormer on 9 of 10 benchmarks with peak wirelength reductions exceeding 32%. Furthermore, we show VeoPlace generalizes to analytical placers, improving DREAMPlace performance on all eight ICCAD 2015 Superblue benchmarks, as illustrated in Figure 1 and quantified in Table 3, with gains up to 4.4%. Our main contributions are:

- **Unified VLM-guided placement framework**: We introduce VeoPlace, a framework that uses multimodal foundation models to guide specialized placement algorithms through spatial reasoning, without requiring fine-tuning. We instantiate this idea with action masking for learning-based policies and soft anchor constraints for analytical placers.
- **Plug-and-play performance gains**: VeoPlace improves DREAMPlace 4.3.0 on all eight ICCAD 2015 Superblue benchmarks (up to 4.4%) and boosts ChiPFormer on 9 of 10 benchmarks (up to 32%), without VLM fine-tuning or modifying the underlying placer. This shows that VLM guidance can improve both analytical and learning-based placers while remaining compatible with existing EDA workflows.

2 Related Work

Vision-Language Models for Decision-Making. Vision-Language Models (VLMs) are trained on vast datasets of text and images, and therefore contain rich priors valuable for tasks requiring both vision and language [10]. The use of VLMs to perform decision-making has been explored in several fields, including robotics, where VLMs interpret natural language commands within a visual scene to guide robot actions or planning [3, 16, 38, 15, 4, 17, 42, 24]. Systems such as SayCan [3] and RT-2 [4] demonstrate how VLMs can translate high-level instructions into actionable plans that low-level controllers can execute.

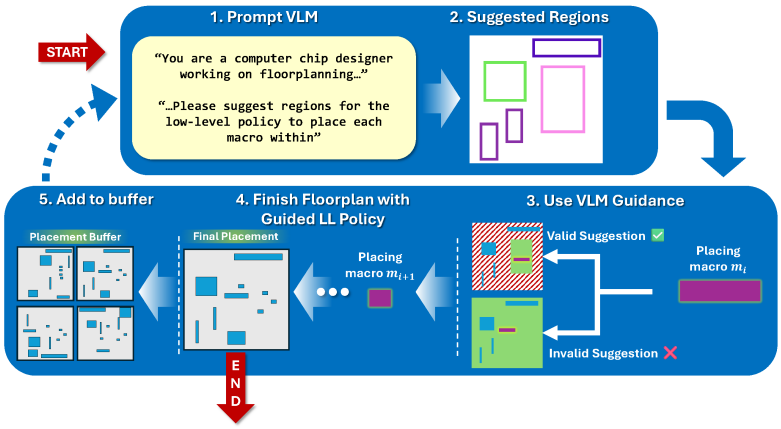


Figure 2: VeoPlace framework overview. The VLM suggests placement regions (1-2) to constrain a low-level placer (3) for macro placement (4). A history buffer that stores the existing population of placements (5) facilitates evolutionary in-context improvement, creating a feedback loop to improve placement quality.

Our work leverages these VLM capabilities in a similar hierarchical approach. The VLM perceives chip placement images along with their performance metrics, analyzes spatial arrangements, and provides high-level guidance to a low-level placer in the form of suggested bounding regions for macros. These bounding regions constrain the low-level placer, effectively creating a division of labor where the VLM handles high-level spatial reasoning while a specialized placer executes precise placement decisions within these constraints.

Automated Chip Floorplanning. Automating computer chip floorplanning has been studied through various approaches (Table 1), including analytical methods [25], black-box optimization techniques such as simulated annealing [45] and genetic algorithms [39], more recent guided black-box methods [37], and various learning-based methods [28, 20, 11]. Within the learning-based category, a prominent line of work formulates chip floorplanning as a reinforcement learning (RL) problem where macros are sequentially placed onto a chip canvas [28, 19, 20]. Alternatively, recent works have proposed learning to refine existing chip floorplans, employing techniques such as diffusion models [23] or RL algorithms [46] for post-processing. Our approach can be viewed as a generalization of learning-based approaches (and can explicitly leverage them in the inner loop) by using a high-level VLM to guide them at test-time.

Table 1: Comparison of macro placement approaches. RL: Reinforcement Learning, MCTS: Monte Carlo Tree Search, IL: Imitation Learning, BBO: Blackbox Optimization, VLM: Vision-Language Model, LLM: Large Language Model.

Method	**Category**
DREAMPlace [25]	Analytical
NTUPlace3 [7]	Analytical
RePlace [8]	Analytical
SP-SA [29]	Packing
ChiPFormer [20]	IL
DeepPR [9]	RL
EfficientPlace [11]	RL + MCTS
GraphPlace [28]	RL
MaskPlace [19]	RL
EvoPlace [48]	LLM + Analytical
MacroRegulate [46]	RL + Analytical
WireMaskBBO [37]	BBO
VeoPlace (Ours)	VLM + IL VLM + Analytical

Pairing LLMs with Evolution. Pairing LLMs with evolutionary search has achieved success in fields such as program generation [35, 14, 26], planning and reasoning [22], scientific discovery [47, 12], robotics [31], and chip design [32, 48, 46, 37]. Most relevant to our work, EvoPlace [48] uses LLMs to evolve optimization algorithms for analytical global placement. In contrast, VeoPlace uses a VLM to evolve the global placement *solutions* directly, suggesting where macros should go based on visual reasoning over prior placements. These suggestions are evaluated, and high-performing ones are selected to inform the VLM's next generation of proposals, given feedback from an objective function. Our selection strategy samples a population of high-performing, geometrically similar placements. That is, VeoPlace explicitly focuses on evolution in a local region, a principle that has been shown to be effective in sparse Gaussian processes [43] and island models in genetic algorithms [35, 22, 40, 5].

3 Preliminaries

Macro Placement. We consider macro placement in chip floorplanning, where a set of *macros* $M = \{m_1, \ldots, m_N\}$, defined by their dimensions and connectivity, are placed on a 2D chip *canvas*. Connectivity is given by a *netlist* $G = (V, E)$, a hypergraph over circuit objects $V = M \cup C$, where C denotes standard cells and each hyperedge (*net*) connects a subset of objects. The objective is to find a macro placement $P = \{p_1, \ldots, p_N\}$, where p_i is the bottom-left corner of macro m_i, that minimizes wirelength. We measure wirelength using half-perimeter wirelength (HPWL), the standard proxy for the total interconnect needed to connect components within each net, and a crucial metric for a chip's performance, power, and area (PPA) [25, 28].

Learning-based macro placers often formulate placement as a Markov Decision Process (MDP) [28, 20, 19]: macros are placed sequentially on a discretized canvas, with state s_t encoding the partial placement, current macro features, and possibly netlist structure. An action a_t selects the grid cell for the current macro's reference point. After all macros are placed, the terminal reward is based on HPWL, which is the sum over nets of the half-perimeter of the smallest bounding box enclosing each net's connected components. In this work, we report global HPWL (gHPWL) after standard-cell placement and legalization, ensuring that the evaluated placement is complete and overlap-free.

Inference-time Optimization. Online RL requires many environment interactions and model updates to produce optimal placements for new netlists. Recent work suggests that offline RL pre-training provides strong zero-shot performance but benefits from fine-tuning on a small amount of online interaction [20]. We consider an inference-time optimization setting that uses a **hierarchical approach**: we are allowed a fixed budget of placement evaluations, but do not fine-tune either the VLM (our high-level strategic guide) or the low-level placer.

Analytical Placer. Analytical placers formulate placement as a nonlinear optimization problem, minimizing wirelength subject to density constraints. We use DREAMPlace 4.3.0 [25], a state-of-the-art GPU-accelerated analytical placer. In our framework, we guide DREAMPlace by incorporating VLM suggestions as soft anchor constraints in the optimization objective, pulling macros toward suggested target locations while still allowing the solver to find low-wirelength, density-aware arrangements.

Learning-Based Placer. ChiPFormer [20] is an autoregressive Transformer that formulates macro placement as offline reinforcement learning via a Decision Transformer objective [6, 21], achieving state-of-the-art performance across multiple chip designs. We adopt it as our learning-based low-level placer due to its multi-task generality. Crucially, ChiPFormer outputs a probability distribution over grid cells for each macro. In our framework, the VLM constrains this distribution by masking out regions outside its suggested placement areas, steering the policy toward better design choices without requiring any fine-tuning.

4 Method

In this section, we describe VeoPlace, our evolutionary framework for using VLM spatial reasoning in chip floorplanning. As shown in Figure 2 and Algorithm 1, VeoPlace maintains a history buffer of evaluated placements and repeatedly improves it over a fixed rollout budget. Algorithm 1 initializes this buffer and iterates over batches of rollouts (Lines 1–2). Each iteration first samples an unguided batch from the base placer, legalizes each placement, evaluates its gHPWL, and stores the result in the history buffer (Lines 3–8). The method then builds a context from the current history and queries the VLM for a batch of macro-region suggestions (Lines 9–10). Finally, it runs a guided batch in which the low-level placer is constrained by those VLM suggestions, legalizes and evaluates the resulting placements, and adds them back to the history buffer (Lines 11–16). This feedback loop lets later VLM queries condition on previous successes and failures.

4.1 VLM and Low-Level Placer Interface

The manner in which VLM suggestions are incorporated depends on the low-level placer: for analytical placers, suggestions become soft anchor constraints in the optimization objective; for learning-based policies, suggestions mask the action space to constrain sampling. We demonstrate VeoPlace with both approaches. Algorithm 1 summarizes the overall procedure, while PLACEMACROS is instantiated for analytical and learning-based placers in Algorithms 2 and 3, respectively. We set $q = 8$ because each Gemini query returns eight candidate suggestions. Thus, one VeoPlace iteration consists of eight unguided rollouts, one VLM query, and eight guided rollouts. With $E = 2{,}000$ total rollouts, this yields $E/(2q) = 125$ VLM calls.

Algorithm 1: VeoPlace

Require: V: VLM; $\mathcal{P}$: Placer; G: Netlist; C: Context length; E: Total rollout budget; q: Batch size
Initialize history buffer $H \leftarrow \emptyset$
for $t = 1$ to $E/(2q)$ **do**
  Phase 1: Unguided batch
  for $j = 1$ to q **do**
    $P_{t,j} \leftarrow \textsc{PlaceMacros}(\mathcal{P}, G, \emptyset)$
    $P_{t,j} \leftarrow \textsc{Legalize}(P_{t,j})$
    $H \leftarrow H \cup \{(P_{t,j}, \mathrm{gHPWL}(P_{t,j}))\}$
  end for
  context $\leftarrow \textsc{BuildContext}(H, C)$
  $\{S_1, \ldots, S_q\} \leftarrow V(\text{context}, G)$
  Phase 2: VLM-guided batch
  for $j = 1$ to q **do**
    $P_{t,j} \leftarrow \textsc{PlaceMacros}(\mathcal{P}, G, S_j)$
    $P_{t,j} \leftarrow \textsc{Legalize}(P_{t,j})$
    $H \leftarrow H \cup \{(P_{t,j}, \mathrm{gHPWL}(P_{t,j}))\}$
  end for
end for
return best placement in H

We use a 256-macro VLM interface, which matches ChiPFormer's sequence length in the learning-based track and keeps prompting cost fixed across both tracks. When a design contains more than 256 macros, we choose the guided subset by sorting macros by area, with connectivity used as a tie-breaker, so the largest and most connected macros receive VLM guidance first. After each full placement, we legalize placement P using DREAMPlace's built-in legalizer to ensure a valid macro and standard-cell placement.

4.1.1 Guiding Analytical Placers

For analytical placers, we interpret VLM suggestions as target macro locations. Let x denote macro locations, y denote standard-cell locations, E denote the set of nets, and M denote the set of macros. The standard DREAMPlace objective minimizes $\sum_{e \in E} \mathrm{WL}(e; x, y) + \lambda\, D(x, y)$, where D is the density penalty. We incorporate VLM suggestions by adding an anchor term with weight λ_A:

Algorithm 2: PLACEMACROS (Analytical)

Require: G: Netlist; S: VLM suggestions (possibly empty)
if $S = \emptyset$ **then**
  Solve standard DREAMPlace objective to obtain P
else
  Convert each $s_i \in S$ to anchor $\hat{x}_i$
  Solve anchored objective with A to obtain P
end if
return P

$$\min_{x,y} \sum_{e \in E} \mathrm{WL}(e; x, y) + \lambda\, D(x, y) + \lambda_A\, A(x; \{\hat{x}_i\}), \tag{1}$$

where A penalizes distance from each guided macro i to its suggested location $\hat{x}_i$. Since macro locations are represented by their bottom-left reference coordinate, we convert each VLM-proposed macro box into an anchor by taking its bottom-left corner. Let $M_g \subseteq M$ be the set of macros with valid VLM suggestions. We use a quadratic anchor, $A(x; \{\hat{x}_i\}_{i \in M_g}) = \sum_{i \in M_g} \|x_i - \hat{x}_i\|_2^2$, which pulls guided macros toward the suggested targets. The weight λ_A controls the strength of VLM guidance, where $\lambda_A = 0$ recovers unconstrained DREAMPlace. DREAMPlace then solves this anchored optimization where macros are softly pulled toward VLM targets (visualized in Figure 3).

4.1.2 Guiding Learning-Based Placers

For learning-based placers, the low-level policy π places macros sequentially by sampling from a distribution over grid locations. When a macro has a VLM suggestion, we represent it as a bounding box region s_t on the chip canvas (visualized in Figure 5) and constrain the policy to that region, denoted by $\pi(\cdot|m_t, P_e, s_t)$ (Algorithm 3). We set out-of-region logits to $-\infty$ and renormalize over valid in-region cells before sampling, so π still controls the exact placement coordinate within the VLM-suggested region.

Algorithm 3: PLACEMACROS (Learned)

Require: π: Policy; G: Netlist; S: VLM suggestions
Order macros by size, connectivity
Initialize $P \leftarrow \emptyset$
for macro m_t **do**
  if S provides valid suggestion s_t **then**
    $p_t \sim \pi(\cdot|m_t, P, s_t)$
  else
    $p_t \sim \pi(\cdot|m_t, P)$
  end if
  $P \leftarrow P \cup \{(m_t, p_t)\}$
end for
return P

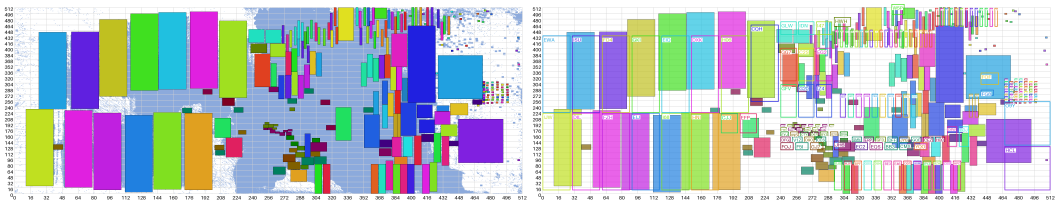


Figure 3: VLM guidance visualization on `superblue1`. **Left:** Final macro placement produced by VeoPlace + DREAMPlace 4.3.0. **Right:** The same placement with VLM-suggested regions overlaid. The VLM proposes target positions for selected macros, and DREAMPlace's loss function is modified with anchor weights to keep macros close to VLM suggestions while allowing standard cells to optimize freely (standard cells removed for visual clarity).

Because the low-level policy autoregressively places macros, a VLM suggestion s_t may be invalid for any $t > 1$ due to already-placed macros overlapping the region. If no valid in-region cell exists, the macro m_t is placed by sampling from the original, unconstrained policy distribution, $p_t \sim \pi(\cdot|m_t, P_e)$. Empirically, this fallback is triggered for $\sim$20% of suggestions once evolution stabilizes (Figure 8d). After completing each placement P_e, we calculate its quality, $\text{gHPWL}(P_e)$, and add $(P_e, \text{gHPWL}(P_e))$ to the history buffer H.

4.2 Structured Prompt

VeoPlace prompts a VLM to propose rectangular regions for the selected macro subset, conditioned on prior placements and their scores $\{(P_i, \text{gHPWL}(P_i))\}$. Each VLM response specifies one region per selected macro, allowing a single query to provide coordinated spatial guidance for the entire selected subset rather than issuing separate queries per macro. The prompt is a simple structured scaffold: it specifies the objective, coordinate system, macro dimensions and color groups, prior placements with metrics, and the required rectangular-region output format. Its key characteristics are (1) this lightweight structure, which elicits spatial reasoning, and (2) the selection of in-context examples (detailed in Section 4.3, with a full example in Appendix E.1). Our structured prompt guides the VLM with clear objectives, constraints, and a standardized format (Table 2), transforming its general visual reasoning into useful spatial guidance. To ensure generalizability and avoid overfitting, we developed the prompt scaffold exclusively on `adaptec1`, using it only to validate the prompt interface (coordinate convention, output validity, and parser compatibility), and then applied the fixed scaffold unchanged to all other benchmarks.

Table 2: Components of the structured prompt for VeoPlace.

Component	Description
Grid Repr.	• 84×84 grid for ChiPFormer experiments (matching [20]); 512×512 grid for DREAMPlace experiments. • Macros positioned by bottom-left corner coordinates. • Origin (0,0) at the bottom-left.
Visual Repr.	• Image of canvas showing placed macros and colors. • Visual context for spatial relationships and patterns.
Context	• Grid specs and macro properties (dimensions, color). • History of prior placements with performance metrics. • Current state (locations) of placed macros.

4.3 Context Selection Strategies for Evolution

The core component of our evolutionary algorithm is prompting the VLM to generate a superior placement suggestion given a set of prior placements and their evaluations. Because each placement uses hundreds of tokens, only a small number can be provided to the model while maintaining reasonable inference cost. Given this limited budget, the examples should be (1) high-quality, so the model improves upon good placements, and (2) informative enough for the model to effectively deduce better placements via reasoning.

We compare multiple context selection strategies (FIFO, Random, Best, Diverse, Top Stratified) in Section 5.3, and full definitions are provided in Appendix D.6. We use **Top Stratified** as our default because it balances exploration (sampling across clusters) with exploitation (favoring high-performing placements within a promising cluster).

5 Experiments

We design experiments to address four key questions: **(Q1)** Can VLM guidance improve the placement quality of both learning-based policies and analytical placers using only inference-time computation? **(Q2)** Which VeoPlace design choices (prompt strategy, input modality, VLM, anchor weight, context selection strategy, and context length) most affect performance? **(Q3)** How does VeoPlace perform on downstream PPA metrics? **(Q4)** What runtime and API-cost overhead does VLM guidance introduce?

Evaluation scope. VeoPlace is a guidance framework rather than a standalone placer. We therefore evaluate the additional value of VLM guidance by comparing each underlying placer against the same placer augmented with VLM-generated spatial proposals, using the same rollout budget, legalization flow, and gHPWL evaluation protocol. This design tests whether an off-the-shelf VLM can serve as a history-conditioned spatial proposal operator for existing placers. This is nontrivial because the model must interpret prior placements and scores, identify useful geometric patterns, propose parseable macro regions under boundary and overlap constraints, and balance exploration with refinement without a benchmark-specific mutation heuristic.

5.1 Experimental Setup

For both experimental tracks, VeoPlace generates an equal number of guided and unguided rollouts in each batch: 8 episodes from directly sampling the base placer, and 8 episodes guided by VLM suggestions (see Table 9 for Gemini sampling parameters). All reported gHPWL values are computed after standard cells are placed and the full floorplan is legalized.

5.1.1 Analytical Placer

We evaluate on the ICCAD 2015 Superblue benchmarks [18] using DREAMPlace 4.3.0 [25] as the base placer. In this track, DREAMPlace is the low-level placer, so each rollout jointly optimizes macros and standard cells while incorporating VLM suggestions as soft anchor constraints in the DREAMPlace objective. We discretize the canvas to a 512×512 grid for VLM suggestions and use Gemini 3.6 Flash with $C{=}25$ context examples. VP+DP uses one global anchor weight $\lambda_A{=}0.01$ on all eight designs. All experiments run 2,000 rollouts across five seeds and report gHPWL (see Appendix C.3.1 for full hyperparameters).

5.1.2 Learning-Based Placer

Following Lai et al. [20], we evaluate on open-source chip benchmarks from the ISPD 2005 challenge [30] and ICCAD 2004 [1, 2], as well as the Ariane RISC-V CPU [33]. For this track, the selected benchmarks vary in complexity, with tens to hundreds of macros or macro-like objects and up to hundreds of thousands of standard cells. We train ChiPFormer from scratch using the public repository[1] and use an $84{\times}84$ grid to match its original setup. We use Gemini 2.5 Flash with $C{=}1$, as even a single in-context example is enough to guide ChiPFormer to better solutions. We run 2,000 rollouts across three seeds (further details in Appendix C.2.2; see Figure 5 for a visualization of VLM-guided placement).

Our setup differs from Lai et al. [20] in two aspects. (1) During the evolutionary search, we cluster standard cells for faster reward computation, following prior work [28, 23], but report unclustered results for final evaluation. (2) During final evaluation, ChiPFormer/VeoPlace macro locations are fixed and DREAMPlace places standard cells around them, following the two-stage evaluation protocol of Mirhoseini et al. [28]. This evaluates the macro placement produced by the learning-based track. Empirically, allowing movable macros results in significant changes to their final placements, confounding the actual efficacy of the base placer (see Figure 7).

[1] `https://github.com/laiyao1/chipformer`

5.2 Q1: Does VLM Guidance Improve Placement Quality?

Table 3: VeoPlace-guided DREAMPlace (VP+DP) vs. DREAMPlace 4.3.0 (DP) and Superblue baselines reported by Xue et al. [46]. All columns report gHPWL ($\times 10^7$, lower is better). VP+DP uses Gemini 3.6 Flash and one global anchor weight λ_A=0.01 on all designs. VP+DP/DP report mean ± standard error of the best gHPWL observed across five seeds with 2,000 rollouts. Percentages compare VP+DP to matched DP 4.3.0.

Benchmark	VP+DP	DP 4.3.0	DP 4.0	AutoDMP	WireMask-EA	MaskPlace	MaskRegulate
superblue1	**37.29±0.03** (−1.0%)	37.66±0.01	89.6±8.4	81.3±1.7	80.7±3.8	79.3±0.6	57.7±0.5
superblue3	**43.81±0.04** (−3.4%)	45.38±0.03	128.7±17.3	81.3±6.9	93.7±8.1	89.0±1.7	70.5±0.3
superblue4	**28.68±0.06** (−1.4%)	29.09±0.03	68.1±2.3	45.7±7.8	55.1±0.7	52.8±0.3	41.5±0.6
superblue5	**39.79±0.04** (−0.7%)	40.09±0.03	87.8±14.7	126.7±40.9	102.3±6.8	98.1±0.3	69.4±0.0
superblue7	**53.88±0.09** (−1.0%)	54.45±0.04	227.0±9.1	100.4±16.3	100.5±3.4	99.9±0.5	79.0±0.3
superblue10	**68.33±0.12** (−4.4%)	71.45±0.12	148.1±13.3	114.8±17.8	135.2±22.5	109.4±2.1	112.3±3.8
superblue16	**36.44±0.01** (−0.5%)	36.63±0.01	102.2±5.8	61.2±24.9	61.3±1.4	55.3±0.4	55.3±0.4
superblue18	**21.75±0.11** (−0.8%)	21.92±0.01	49.7±13.8	30.2±0.1	34.6±0.7	34.9±0.6	29.6±0.1

Analytical Placers. As shown in Table 3, VeoPlace outperforms DREAMPlace 4.3.0 on all eight Superblue benchmarks, with improvements ranging from 0.5% (superblue16) to 4.4% (superblue10). VeoPlace also achieves lower reported gHPWL than the published DREAMPlace 4.0, AutoDMP, WireMask-EA, MaskPlace, and MaskRegulate baselines included in Table 3 on all eight benchmarks. These published baselines have $1.35\times$ to $4.2\times$ the gHPWL of DREAMPlace 4.3.0 on every benchmark, showing that prior learning-based approaches have struggled to outperform the most recent state-of-the-art analytical placer. Qualitative placement examples are shown in Figure 4. We ablate each design choice of VeoPlace in Section 5.3.

Table 4: VeoPlace (VP) with ChiPFormer (CF). VP uses Gemini 2.5 Flash. Mean ± SE of best gHPWL ($\times 10^7$, lower is better) across three random seeds with 2,000 rollouts each. Percentages compare VP+CF to the better CF baseline. Best results are bolded.

	VP+CF	CF	
Benchmark	(C=1)	No-FT	FT
adaptec1	15.97±0.80 (+12.2%)	**14.23±0.77**	14.99±0.55
adaptec2	**11.82±0.89** (−13.8%)	15.03±0.59	13.71±0.33
adaptec3	**24.10±0.21** (−6.0%)	25.64±0.43	26.31±0.97
adaptec4	**19.63±0.20** (−8.7%)	22.65±1.69	21.50±0.50
ibm01	**0.31±0.01** (−22.5%)	0.44±0.04	0.40±0.01
ibm02	**0.59±0.01** (−32.2%)	0.91±0.02	0.87±0.06
ibm03	**0.80±0.02** (−16.7%)	1.16±0.10	0.96±0.02
ibm04	**0.84±0.00** (−12.5%)	1.15±0.06	0.96±0.02
ariane133	**0.34±0.02** (−24.4%)	0.46±0.00	0.45±0.02
ariane136	**0.34±0.02** (−26.1%)	0.46±0.01	0.48±0.01

Learning-Based Placers. This comparison isolates the value of VLM guidance as a test-time adaptation mechanism for ChiPFormer. Under the same 2,000-rollout budget, VeoPlace improves over both repeatedly sampling the fixed ChiPFormer policy (No-FT) and online fine-tuning of ChiPFormer itself (FT) on 9 of 10 benchmarks (Table 4). The largest gains occur on `ibm02` (32%) and `ariane136` (26%), showing that VLM-guided spatial proposals can provide stronger adaptation than updating the low-level policy weights.

5.3 Q2: Which design choices matter?

We ablate the prompt strategy, input modality, VLM, anchor weight λ_A, context selection strategy, and context length C. We also compare against two evolutionary search controls that replace the VLM. The input modality, VLM, and search control experiments use all eight Superblue designs, and the others use `superblue1` (full plots in Figure 8).

Prompt Strategy. We compare our default prompt against a **Greedy** variant that encourages minor refinements and an **Exploratory** variant that pushes for novel placements. On `superblue1`, the default prompt performs best, achieving 37.47 ± 0.11 versus 37.51 ± 0.09 for Greedy and 37.64 ± 0.11 for Exploratory (Table 17).

Input Modality. We remove either the chip canvas image or the textual context from each in-context example, or shuffle the images across placements, while keeping top-stratified selection and C=25

fixed. All four prompts improve over DREAMPlace on all eight Superblue designs and differ from each other by under 0.8% (Table 18), so the gain does not depend on a single input channel.

VLM Choice. With the open-weight Qwen3-VL 8B Instruct or Qwen3-VL 30B A3B Instruct served locally with vLLM in place of Gemini, VP+DP still improves over DREAMPlace on all eight designs and matches Gemini 3.6 Flash (Table 5).

Table 5: VP+DP with open-weight VLMs on Superblue: gHPWL ($\times 10^6$, mean $\pm$ standard error over five seeds, lower is better). All runs use 2,000 rollouts. Bold is best and underlined is second best. Both Qwen models are served locally with vLLM and use top-stratified history with C=25 and the global anchor weight λ_A=0.01.

VLM (VP+DP)	sb1	sb3	sb4	sb5	sb7	sb10	sb16	sb18
DP 4.3.0	376.63±0.13	453.75±0.33	290.92±0.26	400.88±0.30	544.45±0.35	714.47±1.18	366.34±0.12	219.24±0.07
Gemini 3.6 Flash	**372.85±0.34**	438.14±0.42	286.82±0.55	**397.90±0.41**	**538.76±0.86**	**683.25±1.22**	364.43±0.09	217.50±1.09
Qwen3-VL 8B Instruct	374.52±0.41	437.04±0.93	284.03±0.75	400.50±0.26	539.85±0.38	686.85±0.51	364.46±0.38	**217.25±0.28**
Qwen3-VL 30B A3B Instruct	375.54±0.59	**432.98±1.35**	**283.40±0.84**	400.29±0.34	539.58±0.74	686.25±1.26	**364.40±0.39**	217.59±0.32

Evolutionary Search Controls. We perform ablations to test whether the improvement over DREAMPlace comes from our search loop or from the VLM. In the first ablation, we replace the VLM's proposed regions with random regions. In the second ablation, we replace VLM guidance by anchoring each rollout to the best placement found so far. Both ablations run 4,000 rollouts, twice the 2,000 used by VeoPlace. VeoPlace outperforms both ablations on all eight Superblue designs (Table 16).

Anchor Weight. The anchor weight, λ_A, controls the strength of the auxiliary VLM anchor term relative to DREAMPlace's native wirelength and density terms. As shown in Figure 8a, lower anchor weights perform better overall, while $\lambda_A \in \{0.1, 1.0\}$ degrades performance. High anchor weights overconstrain macro locations and prevent standard cells from flowing freely around them. The main Superblue results use λ_A=0.01 on every benchmark (Appendix C.3.1).

Context Selection Strategy. We compare FIFO, Random, Best, Diverse, and Top Stratified strategies for selecting the C in-context examples from the history buffer (definitions in Appendix D.6). As shown in Figure 8c, each strategy continues to improve over rollouts, suggesting VeoPlace is robust to this design choice. Notably, "Best" performs worst, indicating that greedily selecting high-performing placements can trap the evolutionary loop in local minima.

Invalid Suggestion Rate. For learning-based placers, a suggested region can become invalid when it overlaps macros that were already placed earlier in the autoregressive sequence; in that case, the policy samples from its unconstrained distribution. For analytical placers, invalid suggestions correspond to malformed, out-of-bounds, or overlapping proposed macro boxes, and the corresponding anchor term is omitted. As shown in Figure 8d, all strategies converge to $\sim$20% invalid suggestions.

Context Length. We test $C \in \{1, 10, 25\}$ examples. As shown in Figure 8b, C=25 achieves lower gHPWL, suggesting that more in-context examples help the VLM better understand patterns associated with high-quality macro placements.

5.4 Q3: How does VeoPlace perform on downstream PPA metrics?

Because gHPWL is a placement-stage proxy, we also measure post-route power, performance, and area (PPA). We use four designs with open Nangate45 collateral (Ariane-133, Ariane-136, BlackParrot, and BP-Quad), and push every method's macro placement through the identical OpenROAD flow up to detailed routing. VP+DP has the shortest routed wirelength on Ariane-136, BlackParrot, and BP-Quad. On all four designs, VP+DP also has the smallest area and the best or tied TNS and number of violating pins (Tables 6 and 15).

5.5 Q4: What runtime and API-cost overhead does VLM guidance introduce?

Any iterative macro-placement method must repeatedly invoke a placer to position standard cells around macros and evaluate candidate layouts, so the additional cost introduced by VeoPlace comes from VLM inference. In both experimental tracks, each 2,000-rollout run uses 125 Gemini calls: one query per batch of eight guided rollouts, paired with eight unguided rollouts. Gemini 3.6 Flash averages 62 seconds per call, for about 2.1 hours of benchmark-independent latency per run.

Table 6: Post-route PPA on four Nangate45 designs after the full OpenROAD flow. The columns are routed wirelength (WL, 10^6 μm), worst negative slack (WNS, ns), total negative slack (TNS, 10^3 ns), and power (mW). Table 15 reports the full set of metrics. VP+DP uses Gemini 3.5 Flash. Bold is best and underlined is second best. Error terms are standard deviations over three seeds. An asterisk (*) marks a mean over two seeds, because the third seed failed to route in OpenROAD. Plain values are single runs.

Method	Metric	Ariane-133	Ariane-136	BlackParrot	BP-Quad
Hier-RTLMP	Routed WL	6.91	7.46	6.11	52.56
	WNS	-0.411	**0.432**	-2.991	-150.056
	TNS	-0.719	**0.000**	**-0.003**	-38511.600
	Power	**254.3**	162.1	147.0	**2008.7**
DP 4.3.0	Routed WL	**5.93±0.19**	6.43±0.05	5.71±0.02	45.91±0.16
	WNS	-0.368±0.041	0.268±0.055	-2.990±0.028	-149.712±0.087
	TNS	-0.645±0.061	**0.000**	**-0.003**	-38497.433±4.661
	Power	254.4±1.1	**153.6±0.1**	147.6±0.3	2018.1±6.6
WireMask-EA	Routed WL	7.51±0.79*	7.27±0.62	6.88±0.22	58.22
	WNS	-0.820±0.440*	0.114±0.087	**-2.938±0.049**	-150.507
	TNS	-2.002±0.827*	**0.000**	**-0.003**	-38547.600
	Power	258.7±0.4*	154.9±0.8	151.3±5.8	2058.5
EGPlace	Routed WL	7.12±0.04*	7.13±0.45*	7.32±0.41	58.44±1.63
	WNS	-0.688±0.039*	-0.062±0.005*	-3.025±0.072	-150.504±0.308
	TNS	-1.431±0.160*	-0.004±0.001*	**-0.003**	-38552.300±11.772
	Power	258.6±0.9*	155.9±1.7*	153.0±5.0	2073.1±22.3
LaMPlace	Routed WL	7.49±1.21	7.59±0.51	7.33±0.29	63.92±2.14*
	WNS	-0.496±0.171	0.317±0.070	-3.028±0.065	-150.498±0.006*
	TNS	-1.259±0.767	**0.000**	**-0.003**	-38547.200±3.323*
	Power	256.9±2.2	160.7±2.8	153.3±2.7	2051.5±9.3*
VP+DP 3.5 Flash	Routed WL	5.98±0.09	**6.35±0.04**	**5.58±0.02**	**44.57±0.93**
	WNS	**-0.339±0.026**	0.277±0.121	-2.965±0.014	**-149.585±0.029**
	TNS	**-0.549±0.039**	**0.000**	**-0.003**	**-38492.100±5.450**
	Power	255.3±0.5	153.7±0.5	**146.4±0.5**	2012.5±7.8

Table 8 reports the resulting wall-clock overhead for one run on a single A100 GPU: 2%–7% over DREAMPlace, so DREAMPlace accounts for 93%–98% of a run. VeoPlace requires no VLM training, and under conservative token-count estimates, the API-cost overhead is about $7 per 2,000-rollout experiment (see Appendix B.5.2).

6 Limitations & Future Work

The search loop optimizes only gHPWL. We evaluate post-route PPA only after the search, so the loop does not use PPA metrics. This evaluation covers four Nangate45 designs (Table 6). The full physical-design flow is too slow to run inside the loop (Appendix D.3). VeoPlace depends on VLM calls, which add the latency and API cost reported in Section 5.5. VeoPlace also runs with open-weight Qwen3-VL models served locally (Table 5). The VLM guides at most 256 macros per design. High-quality chips ultimately require optimizing additional PPA metrics such as timing and power. Because VeoPlace relies on natural language prompting rather than a fixed loss function, these metrics can be incorporated directly into the VLM prompt and evolutionary scoring loop when available. Future work will add more macros to the prompt and extend VeoPlace to multi-objective PPA optimization.

7 Conclusion

We proposed VeoPlace, an evolutionary framework that leverages vision-language models to enhance chip floorplanning. Through a structured prompting approach that requires no fine-tuning of the VLM, VeoPlace improves strong learning-based and analytical placers, providing a blueprint for integrating multimodal foundation models into complex engineering workflows as high-level strategic guides for computer-aided design.

Acknowledgments

This work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence at Harvard University. Part of the computations in this paper were run on the FASRC Cannon cluster supported by the FAS Division of Science Research Computing Group at Harvard University.

References

[1] Saurabh N Adya and Igor L Markov. Consistent placement of macro-blocks using floorplanning and standard-cell placement. In *Proceedings of the 2002 international symposium on Physical design*, pages 12–17, 2002.

[2] Saurabh N Adya, Shubhyant Chaturvedi, Jarrod A Roy, David A Papa, and Igor L Markov. Unification of partitioning, placement and floorplanning. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 550–557. IEEE, 2004.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[5] Erick Cantú-Paz et al. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[7] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, 2008.

[8] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2018.

[9] Ruoyu Cheng and Junchi Yan. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 34:16508–16519, 2021.

[10] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[11] Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Mingxuan Yuan, Jianye Hao, Yongdong Zhang, and Feng Wu. Reinforcement learning within tree search for fast macro placement. In *Forty-first International Conference on Machine Learning*, 2024.

[12] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, et al. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

[13] Aric Hagberg, Pieter J Swart, and Daniel A Schult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008.

[14] Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. Evolving code with a large language model. *Genetic Programming and Evolvable Machines*, 25(2):21, 2024.

[15] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[16] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022.

[17] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[18] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 921–926. IEEE, 2015.

[19] Yao Lai, Yao Mu, and Ping Luo. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems*, 35:24019–24030, 2022.

[20] Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, and Ping Luo. Chipformer: Transferable chip placement via offline decision transformer. In *International Conference on Machine Learning*, pages 18346–18364. PMLR, 2023.

[21] Kuang-Huei Lee, Ofir Nachum, Mengjiao Sherry Yang, Lisa Lee, Daniel Freeman, Sergio Guadarrama, Ian Fischer, Winnie Xu, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35:27921–27936, 2022.

[22] Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*, 2025.

[23] Vint Lee, Chun Deng, Leena Elzeiny, Pieter Abbeel, and John Wawrzynek. Chip placement with diffusion. *arXiv preprint arXiv:2407.12282*, 2024.

[24] Jacky Liang, Fei Xia, Wenhao Yu, Andy Zeng, Montserrat Gonzalez Arenas, Maria Attarian, Maria Bauza, Matthew Bennice, Alex Bewley, Adil Dostmohamed, et al. Learning to learn faster from human feedback with language model predictive control. *arXiv preprint arXiv:2402.11450*, 2024.

[25] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, Las Vegas NV USA, June 2019. ACM. ISBN 978-1-4503-6725-7. doi: 10.1145/3316781.3317803. URL `https://dl.acm.org/doi/10.1145/3316781.3317803`.

[26] Vadim Liventsev, Anastasiia Grishina, Aki Härmä, and Leon Moonen. Fully autonomous programming with large language models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1146–1155, 2023.

[27] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. eplace: Electrostatics-based placement using fast fourier transform and nesterov's method. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(2):1–34, 2015.

[28] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, and Azade Nova. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[29] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996. doi: 10.1109/43.552084.

[30] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. The ispd2005 placement contest and benchmark suite. In *Proceedings of the 2005 international symposium on Physical design*, pages 216–220, 2005.

[31] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, et al. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. In *International Conference on Machine Learning*, pages 37321–37341. PMLR, 2024.

[32] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. Technical report, Google Deepmind, 2025.

[33] OpenHW Group. CVA6 RISC-V CPU. `https://github.com/openhwgroup/cva6`, 2024.

[34] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2 edition, 2003.

[35] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[36] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[37] Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. Macro placement by wire-mask-guided black-box optimization. *Advances in Neural Information Processing Systems*, 36:6825–6843, 2023.

[38] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pages 894–906. PMLR, 2022.

[39] T Singha, HS Dutta, and M De. Optimization of floor-planning using genetic algorithm. *Procedia Technology*, 4:825–829, 2012.

[40] Reiko Tanese. *Distributed genetic algorithms for function optimization*. University of Michigan, 1989.

[41] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[42] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[43] Yunyue Wei, Vincent Zhuang, Saraswati Soedarmadji, and Yanan Sui. Scalable bayesian optimization via focalized sparse gaussian processes. *arXiv preprint arXiv:2412.20375*, 2024.

[44] Neil H. E. Weste and David Money Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 4 edition, 2011.

[45] DF Wong and CL Liu. A new algorithm for floorplan design. In *23rd ACM/IEEE Design Automation Conference*, pages 101–107. IEEE, 1986.

[46] Ke Xue, Ruo-Tong Chen, Xi Lin, Yunqi Shi, Shixiong Kai, Siyuan Xu, and Chao Qian. Reinforcement learning policy as macro regulator rather than macro placer. *Advances in Neural Information Processing Systems*, 37:140565–140588, 2024.

[47] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

[48] Xufeng Yao, Jiaxi Jiang, Yuxuan Zhao, Peiyu Liao, Yibo Lin, and Bei Yu. Evolution of optimization algorithms for global placement via large language models. *arXiv preprint arXiv:2504.17801*, 2025.

Appendix

A Additional Details

A.1 Benchmark Statistics

Table 7: Benchmark statistics for all evaluated circuits. IBM benchmarks do not distinguish macros from standard cells; their macro counts come from DREAMPlace, which treats a movable object as a macro when its area exceeds 10 times the mean movable-cell area and its height spans more than two standard-cell rows.

Benchmark	# Objects	# Macros	# Nets	# Pins
ISPD 2005				
adaptec1	211K	63	221K	944K
adaptec2	255K	127	266K	1.1M
adaptec3	452K	58	467K	1.9M
adaptec4	496K	69	516K	1.9M
ICCAD 2004				
ibm01	13K	246	14K	51K
ibm02	20K	271	20K	81K
ibm03	23K	290	27K	94K
ibm04	28K	295	32K	106K
Ariane (Nangate45)				
ariane133	—	133	—	—
ariane136	—	136	—	—
ICCAD 2015 Superblue				
superblue1	1,215K	424	1,216K	3.8M
superblue3	1,219K	565	1,225K	3.9M
superblue4	802K	300	803K	2.5M
superblue5	1,090K	770	1,101K	3.2M
superblue7	1,938K	441	1,934K	6.4M
superblue10	984K	1,629	1,898K	5.6M
superblue16	986K	99	1,000K	3.0M
superblue18	772K	201	772K	2.6M

A.2 Visual Examples

A.2.1 Analytical Placer Comparison

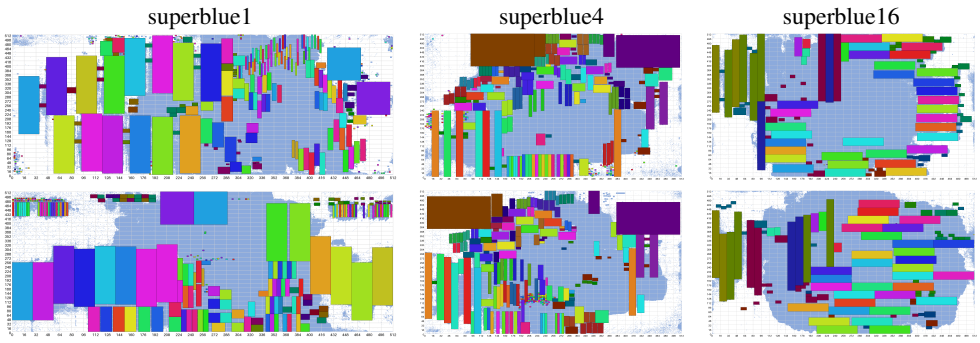
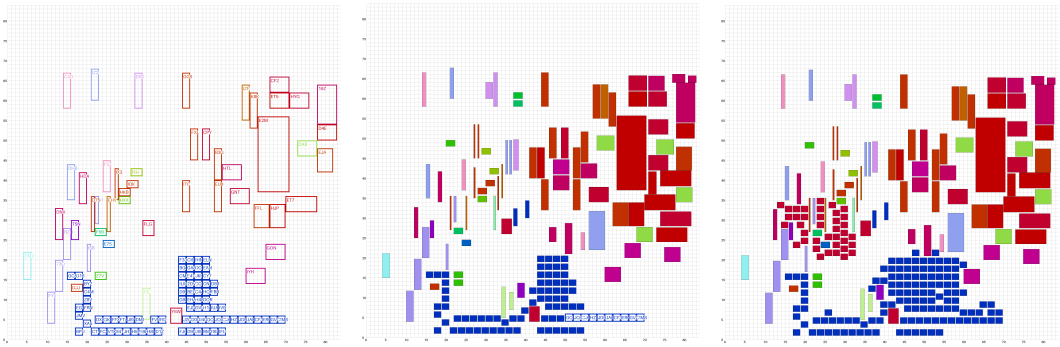


Figure 4: Visual comparison of placements on selected Superblue benchmarks. **Top row:** DREAMPlace 4.3.0. **Bottom row:** VeoPlace-guided DREAMPlace. Blue clouds are individual standard cells; colored rectangles are macros. VeoPlace guidance improves gHPWL on all eight Superblue benchmarks.

A.2.2 VLM-Guided Placement

To provide a qualitative illustration of our method, Figure 5 visualizes a complete VLM-guided rollout on the `adaptec4` benchmark. This example highlights the hierarchical division of labor central to VeoPlace: the VLM provides a high-level spatial strategy, and the low-level policy executes the precise placement decisions within that strategy. The figure shows the initial VLM proposals ($t = 0$), the state mid-placement ($t = T/2$), and the final floorplan ($t = T$) that results from this guided process.

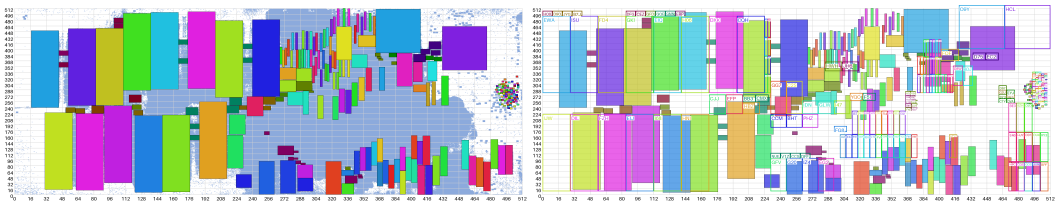


(a) $t = 0$ (b) $t = \frac{T}{2}$ (c) $t = T$

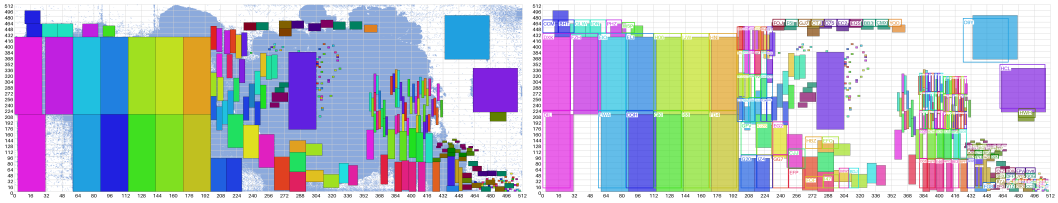
Figure 5: VeoPlace's VLM-guided placement on adaptec4. (a) VLM proposes initial regions ($t = 0$); policy is unconstrained for macros without valid suggestions. (b) Mid-placement ($t = T/2$). (c) Final placement ($t = T$), with the policy operating within VLM constraints.

A.2.3 Anchor Weight Comparison

To illustrate the effect of anchor weight on placement quality, Figure 6 shows the final placements for `superblue1` at different anchor weights. With low anchor weights ($\lambda_A = 0.001$), DREAMPlace has sufficient flexibility to route standard cells around the VLM-suggested macro positions. Higher anchor weights ($\lambda_A = 1.0$) overly constrain the optimization, resulting in suboptimal standard cell placement.



(a) $\lambda_A = 0.001$ (best)



(b) $\lambda_A = 1.0$ (too restrictive)

Figure 6: Visual comparison of placements at different anchor weights on `superblue1`. Lower anchor weights allow the analytical placer more flexibility to optimize standard cell positions around the macro placements.

A.3 Macro Coloring

We employ a color-coding strategy to implicitly convey functional relationships between macros to the VLM. This process involves several steps:

First, we construct a **macro-connectivity graph** from the original netlist G. In this graph, nodes (excluding standard cells) represent the macros to be placed. An undirected, weighted edge is created between any two macros if they share one or more nets in G. The weight of such an edge is proportional to the number of nets these two macros commonly share. This construction effectively flattens the hypergraph structure of the netlist into a standard graph, where indirect connections through nets are represented as weighted between macros.

This macro-connectivity graph is embedded into a low-dimensional space (specifically, an 8-dimensional space in our implementation for k-means) using a spring-based graph layout algorithm. Such algorithms, like the one implemented in the NetworkX [13] Python library, position macros in the embedding space such that those with stronger connections in the graph are located closer to one another in space.

With macros represented as points in this embedding space, we apply k-means clustering to group them. To determine a suitable number of clusters, k, we iterate through a predefined range of potential k values (e.g., from 2 to 30). For each k, we perform k-means clustering and evaluate the resulting cluster separation using the Silhouette score [36]. The value of k (and its corresponding clustering) that yields the highest Silhouette score is selected as optimal.

Finally, macros are assigned colors based on their cluster membership: all macros within the same cluster receive the same unique color. Any macros that are not part of the main connectivity graph (e.g., isolated macros not sharing nets with other considered macros, if any) are assigned a default gray color. The coloring is used to guide VLM suggestions only: it provides the VLM with an approximate visual encoding of netlist connectivity so it can identify which macros are more closely related. The colors do not constrain the final placement.

B Experimental Setup

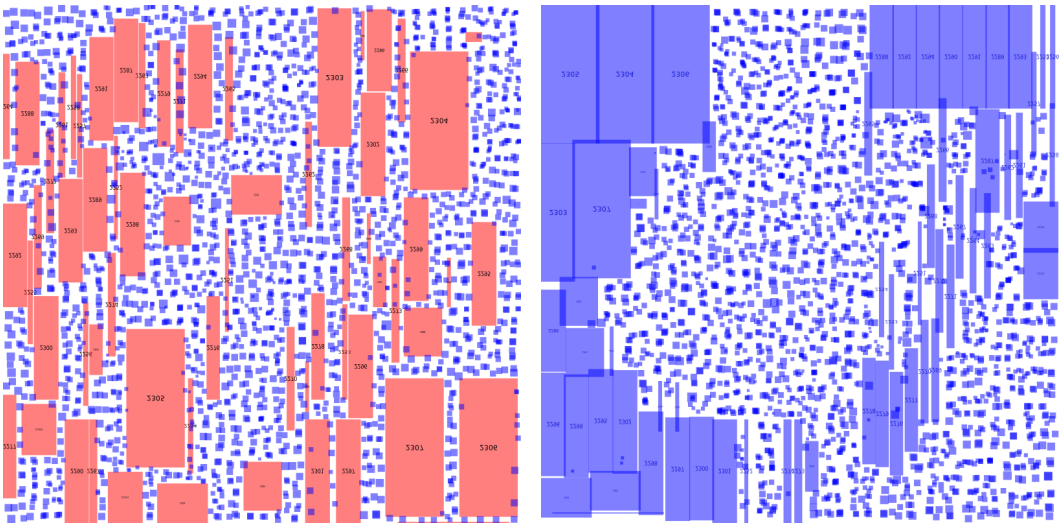
B.1 Justification for Fixed Macros During Standard Cell Placement

This section applies only to the learning-based evaluation, where the object being evaluated is the macro placement produced by ChiPFormer or VeoPlace (Section 5.1.2). It does not apply to

the analytical-track experiments, where DREAMPlace jointly optimizes macro and standard-cell positions with VLM soft-anchor guidance (Section 5.1.1).

In the learning-based evaluation, we report results with fixed macro placements, where the analytical placer only positions the standard cells around the macros (a two-stage flow). This differs from some prior work that allows the analytical placer to move all cells, including macros (a three-stage flow). We find that allowing macros to be movable during the analytical placement stage results in substantial displacements from their original VeoPlace-generated locations. This confounds the evaluation, as the final placement is no longer representative of VeoPlace's learned spatial reasoning but rather the output of the analytical placer.

Figure 7 provides a clear visual demonstration of this effect on the `adaptec1` benchmark. The significant displacement of macros is evident, justifying our decision to use a fixed-macro evaluation to ensure we are measuring the direct efficacy of VeoPlace.



(a) Our two-stage flow (VeoPlace with fixed macros). (b) Alternative three-stage flow (movable macros).

Figure 7: Visual comparison demonstrating why a fixed-macro flow is essential for a fair evaluation of VeoPlace. (a) The final layout from our two-stage flow, where macro placements generated by VeoPlace are **fixed**. (b) The result of an alternative three-stage flow, which takes the **exact same initial placement** from (a) as input but allows DREAMPlace to move all cells. The analytical placer's drastic rearrangement of macros in (b) shows that it effectively ignores the initial solution, making its final metrics an invalid measure of VeoPlace's contribution. Our fixed-macro approach ensures a direct and unconfounded evaluation. The flipped macro IDs are a DREAMPlace rendering artifact. We did not modify the DREAMPlace code.

B.2 Model Size

The ChiPFormer decision transformer model contains approximately 3 million trainable parameters.

B.3 Pre-training

The pre-training phase was conducted on servers equipped with 4× NVIDIA A100 40GB GPUs.

B.4 Rollout and Inference

Generating ChiPFormer macro-placement candidates is lightweight, requiring only forward passes through the trained 3M-parameter model on a single A100 GPU or equivalent. Each fine-tuning update trains the 3M-parameter model for 10 epochs on the 256-episode buffer and takes seconds, while the eight DREAMPlace evaluations in the same block take minutes, so fine-tuning is under 1% of a run's wall clock. Reported gHPWL values are computed using the fixed-macro DREAMPlace evaluation flow described in Section 5.1.2, which places standard cells and legalizes the full design.

B.5 VLM Integration

For VLM integration, we used Google's Gemini API: Gemini 2.5 Flash for the learning-based track and the `superblue1` ablations, Gemini 3.6 Flash for the Superblue results, and Gemini 3.5 Flash for the post-route PPA table. Our experiments are organized into VeoPlace iterations, where each iteration consists of 8 unguided rollouts, one VLM query, and 8 guided rollouts.

Over a 2,000-rollout experiment, this results in exactly 125 calls to the VLM. Each call to Gemini returns 8 candidate generations, with each generation being a complete set of suggested regions for the selected macros.

B.5.1 Wall-Clock Time

For the analytical-track Superblue experiments, the VLM overhead is dominated by Gemini response latency. Each 2,000-rollout run makes 125 queries. Gemini 3.6 Flash averages 62 seconds per query, for about 2.1 hours of added latency per run. Table 8 reports the per-rollout DREAMPlace latency and total wall-clock time on a single A100 GPU.

Because the VLM latency is a fixed cost per run, its relative overhead decreases on larger designs, dropping from 7% on `superblue18` to 2% on `superblue10`.

Table 8: Wall-clock overhead for VeoPlace-guided DREAMPlace on Superblue. DP rollout is one vanilla DREAMPlace run; DP/VP+DP report total time for one 2,000-rollout run on one A100, with VP+DP including 125 Gemini 3.6 Flash queries at the measured 62 s per query.

Benchmark	DP rollout (s)	DP (h)	VP+DP (h)	Overhead
superblue1	83	46	48	+5%
superblue3	168	93	95	+2%
superblue4	127	70	72	+3%
superblue5	95	53	55	+4%
superblue7	122	68	70	+3%
superblue10	190	106	108	+2%
superblue16	98	55	57	+4%
superblue18	52	29	31	+7%

B.5.2 API Cost

Using Gemini 2.5 Flash standard paid-tier pricing and conservative overestimates of 100K input tokens and 10K output tokens per query, the estimated cost per query is

$$0.30 \times \frac{100{,}000}{1{,}000{,}000} + 2.50 \times \frac{10{,}000}{1{,}000{,}000} = 0.055$$

USD. Across 125 queries, this yields an estimated API cost of

$$125 \times 0.055 = 6.875$$

USD, i.e., about $7 per 2,000-rollout experiment. Because the token counts above are intentionally overestimated, the true API cost should be lower.

C Hyperparameters

C.1 Gemini

We use the public Gemini API endpoint for our experiments. Table 9 shows these hyperparameters.

Table 9: Gemini API hyperparameters.

Parameter	Value
Temperature	0.7
Top-k	64
Top-p	0.95
Candidates	8

C.2 Standard Cell Grouping Parameters

As discussed in Section 5, for the learning-based benchmarks (adaptec/ibm/ariane), we group the hundreds of thousands of standard cells into a smaller set of clusters to make training and inference tractable. This grouping is only used to compute the proxy wirelength reward during rollouts; all reported results are gHPWL computed by DREAMPlace with fixed macros.

We use the open-source codebase from Google's Circuit Training project for this task, which implements the grouping methodology first introduced by Mirhoseini et al. [28]. To ensure a consistent basis for comparison, we applied the same set of grouping hyperparameters across all learning-based benchmarks. The specific values for these parameters are detailed in Table 10.

Table 10: Hyperparameters for the Standard Cell Grouping Algorithm.

Parameter	**Value**	**Description**
Number of Groups	2000	The fixed number of clusters to group all standard cells into.
Cell Area Utilization	1.25	A target for the density of cells within a cluster.
Enable Group Breakup	True	A boolean flag that allows the algorithm to split larger groups.

C.2.1 Pretraining

Circuit Tokens For pretraining the circuit token representation component using the Variational Graph Auto-Encoder (VGAE), we used the following hyperparameters:

- Hidden layer dimensions: [32, 32]
- Learning rate: 0.01
- Training epochs: 800

Transformer Following ChiPFormer [20], we use a reward-conditioned transformer with the following hyperparameters:

- Number of transformer layers: 6
- Number of attention heads: 8
- Embedding dimension: 128
- Pretraining epochs: 1,500
- Batch size: 256
- Learning rate: 10^{-3}
- Weight decay: 10^{-2}

C.2.2 Rollout Settings

Fine-tuning The FT baseline follows the ChiPFormer online fine-tuning settings:

- Update schedule: after every 8 rollouts
- Epochs per update: 10
- Batch size: 64

- Replay buffer: the 256 best episodes
- Optimizer: AdamW with cosine annealing

The No-FT baseline is the same code path with the update disabled.

Returns-to-go We configured specific target returns-to-go for each benchmark netlist to guide the generated placements. Since our objective is to minimize wirelength, we define the reward as its negative value. Following the methodology of Decision Transformers, we set these values to ambitious targets, encouraging the model to generate high-quality placements with very low wirelengths. Table 11 shows the target returns-to-go values used for each benchmark circuit in our experiments.

Table 11: Target Returns-to-Go for Different Benchmark Netlists

Netlist	Return-to-go
adaptec1	-2.86E+06
adaptec2	-2.91E+06
adaptec3	-5.90E+06
adaptec4	-6.37E+06
ibm01	-7.00E+04
ibm02	-1.67E+05
ibm03	-2.52E+05
ibm04	-2.86E+05
ariane133	-2.00E+05
ariane136	-2.00E+05

C.3 DREAMPlace

All experiments utilize DREAMPlace version `4.3.0`.

C.3.1 VLM-Guided Placement (Superblue)

For the VLM-guided DREAMPlace experiments on the Superblue benchmarks, we use anchor constraints to incorporate VLM suggestions with a single anchor weight λ_A=0.01 on every benchmark. We chose it on superblue1 by comparing placements at $\lambda_A \in \{1, 0.1, 0.01, 0.001\}$ and keeping the largest weight whose placements follow the VLM suggestions while standard cells still flow freely around the macros.

Table 12 details the DREAMPlace hyperparameters for all benchmarks. All benchmarks use the Nesterov optimizer with a learning rate of 0.01.

D Additional Experiments

D.1 Congestion

We report a congestion proxy (RUDY) for the Superblue benchmarks to verify that VeoPlace's wirelength gains do not come at the expense of routability. The differences are small and mixed across benchmarks.

Table 12: DREAMPlace 4.3.0 Hyperparameters for All Benchmarks

Benchmark	Target Density	Stop Overflow	Density Weight	Num Bins (X)	Num Bins (Y)	Iterations
ICCAD 2015 Superblue						
superblue1	1.00	0.10	8×10^{-5}	1024	1024	1000
superblue3	1.00	0.10	8×10^{-5}	2048	2048	1000
superblue4	1.00	0.10	8×10^{-5}	512	512	1000
superblue5	1.00	0.10	8×10^{-5}	1024	1024	1000
superblue7	1.00	0.10	8×10^{-5}	512	512	1000
superblue10	1.00	0.10	8×10^{-5}	1024	1024	1000
superblue16	1.00	0.10	8×10^{-5}	1024	1024	1000
superblue18	1.00	0.10	8×10^{-5}	512	512	1000
ISPD 2005						
adaptec1	1.00	0.07	8×10^{-5}	512	512	1000
adaptec2	1.00	0.07	8×10^{-5}	1024	1024	1000
adaptec3	1.00	0.07	8×10^{-5}	1024	1024	1000
adaptec4	1.00	0.07	8×10^{-5}	1024	1024	1000
ICCAD 2004						
ibm01	1.00	0.07	8×10^{-5}	512	512	1000
ibm02	1.00	0.07	8×10^{-5}	512	512	1000
ibm03	1.00	0.07	8×10^{-5}	512	512	1000
ibm04	1.00	0.07	8×10^{-5}	512	512	1000
Ariane (Nangate45)						
ariane133	1.00	0.07	8×10^{-5}	512	512	1000
ariane136	1.00	0.07	8×10^{-5}	512	512	1000

Table 13: Congestion proxy (RUDY, lower is better) for VeoPlace-guided DREAMPlace vs. DREAMPlace 4.3.0 on Superblue. VP+DP uses Gemini 2.5 Flash; mean $\pm$ standard error across three seeds. Differences are small and mixed (4–4 split), indicating that VeoPlace's gHPWL improvements do not materially change routability.

Benchmark	VP+DP 2.5 Flash	DP 4.3.0
superblue1	0.93±0.01	**0.92±0.00**
superblue3	1.06±0.02	**1.04±0.00**
superblue4	**0.91±0.02**	0.93±0.00
superblue5	0.85±0.02	**0.80±0.00**
superblue7	**1.03±0.00**	1.06±0.00
superblue10	**1.08±0.02**	1.09±0.01
superblue16	1.13±0.02	**1.09±0.02**
superblue18	**1.02±0.01**	1.10±0.01

D.2 Seed-Paired DREAMPlace Comparison

Each VP+DP run in Table 3 shares its random seed with its DREAMPlace 4.3.0 baseline, so the comparison is paired by construction. Table 14 lists the per-seed differences. VP+DP improves on its seed-paired baseline in 38 of the 40 pairs; the two exceptions are superblue18 seeds 4 and 5.

Table 14: Per-seed paired gHPWL differences ($\times 10^6$), DREAMPlace 4.3.0 minus VP+DP, on Superblue. Positive means VP+DP is better. Green marks pairs where VP+DP is better, and red marks pairs where DREAMPlace 4.3.0 is better. Each column pairs the two runs that share that seed. The per-design means are the Table 3 gaps.

Benchmark	**Seed 1**	**Seed 2**	**Seed 3**	**Seed 4**	**Seed 5**
superblue1	4.26	4.37	2.41	4.01	3.86
superblue3	15.93	13.53	17.84	16.00	14.72
superblue4	2.47	3.91	4.36	4.55	5.18
superblue5	4.93	3.39	2.27	2.51	1.80
superblue7	6.39	9.14	6.59	3.09	3.25
superblue10	33.92	32.16	27.10	32.07	30.86
superblue16	2.00	1.64	1.83	2.13	1.96
superblue18	3.58	3.34	3.86	−0.99	−1.10

D.3 Placement Metrics and Downstream PPA

Macro placement is an early physical-design stage, so our evaluation follows the standard placement protocol of optimizing gHPWL after standard-cell placement and legalization, while also checking density or routability proxies [8, 25, 28, 20, 19]. Full sign-off metrics such as WNS, TNS, DRV count, and power are determined by the complete downstream physical-design flow, including detailed placement, buffering, clock-tree synthesis, routing, and timing optimization. Running this flow can take weeks in realistic design settings, making it impractical as an inner-loop objective for evolutionary search. It would also conflate macro placement quality with downstream implementation choices, and many academic placement benchmarks do not provide the technology libraries and sign-off collateral needed for timing and power analysis.

Wirelength is the canonical objective for placement algorithms and directly targets physical drivers of downstream PPA. Dynamic power scales with switched capacitance, $P_{\mathrm{dyn}} \propto \alpha C V_{dd}^2 f$, and wire capacitance is a major component of load capacitance in modern VLSI processes [34, 44]. Timing is similarly affected by interconnect RC delay, which grows with wire resistance and capacitance. Thus, reducing gHPWL addresses a central source of delay and power at the placement stage.

Because shorter wirelength can still coincide with worse routability, Table 13 reports RUDY congestion on the Superblue benchmarks. The differences are small and mixed across designs, indicating that VeoPlace's wirelength improvements do not materially degrade this congestion proxy. If timing, power, or routing metrics are available in a deployment flow, they can also be included in the VLM prompt and used as additional scores in the same evolutionary loop.

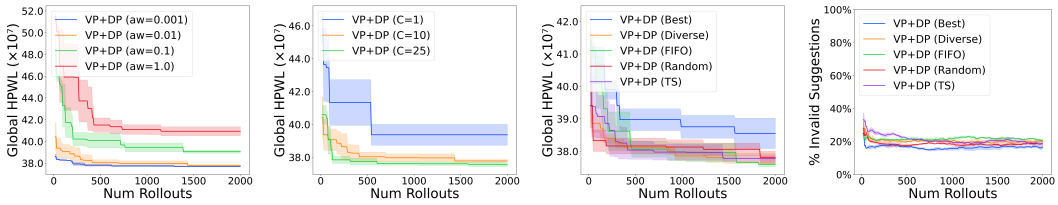
We run the full flow on four designs with open Nangate45 collateral (Ariane-133, Ariane-136, BlackParrot, BP-Quad), pushing every method's macro placement through the identical OpenROAD flow through detailed routing (Table 15). VP+DP has the shortest routed wirelength on Ariane-136, BlackParrot, and BP-Quad and the smallest area on all four designs, with best or tied TNS and violating pins on every design.

Table 15: Post-route PPA on four Nangate45 designs after the full OpenROAD flow. The metrics are routed wirelength (WL, 10^6 μm), worst negative slack (WNS, ns), total negative slack (TNS, 10^3 ns), number of violating pins (NVP), power (mW), area (10^3 μm^2), and DRC violations. VP+DP uses Gemini 3.5 Flash. Bold is best and underlined is second best. Error terms are standard deviations over three seeds. An asterisk (*) marks a mean over two seeds, because the third seed failed to route in OpenROAD. Plain values are single runs.

Method	Metric	Ariane-133	Ariane-136	BlackParrot	BP-Quad
Hier-RTLMP	Routed WL	6.91	7.46	6.11	52.56
	WNS	-0.411	**0.432**	-2.991	-150.056
	TNS	-0.719	**0.000**	**-0.003**	-38511.600
	NVP	2894	**0**	**1**	**259747**
	Power	**254.3**	162.1	147.0	**2008.7**
	Area	727.7	748.6	645.2	5402.6
	DRC	**0**	**0**	**0**	**0**
DP 4.3.0	Routed WL	**5.93±0.19**	6.43±0.05	5.71±0.02	45.91±0.16
	WNS	-0.368±0.041	0.268±0.055	-2.990±0.028	-149.712±0.087
	TNS	-0.645±0.061	**0.000**	**-0.003**	-38497.433±4.661
	NVP	2838±127	**0**	**1**	**259747**
	Power	254.4±1.1	**153.6±0.1**	147.6±0.3	2018.1±6.6
	Area	726.9±1.4	747.9±0.2	643.5±0.0	5355.6±0.7
	DRC	**0**	**0**	**0**	**0**
WireMask-EA	Routed WL	7.51±0.79*	7.27±0.62	6.88±0.22	58.22
	WNS	-0.820±0.440*	0.114±0.087	**-2.938±0.049**	-150.507
	TNS	-2.002±0.827*	**0.000**	**-0.003**	-38547.600
	NVP	4326±963*	**0**	**1**	**259747**
	Power	258.7±0.4*	154.9±0.8	151.3±5.8	2058.5
	Area	730.5±1.5*	751.6±1.9	647.5±1.0	5454.7
	DRC	**0***	**0**	**0**	**0**
EGPlace	Routed WL	7.12±0.04*	7.13±0.45*	7.32±0.41	58.44±1.63
	WNS	-0.688±0.039*	-0.062±0.005*	-3.025±0.072	-150.504±0.308
	TNS	-1.431±0.160*	-0.004±0.001*	**-0.003**	-38552.300±11.772
	NVP	3321±62*	142±28*	**1**	**259747**
	Power	258.6±0.9*	155.9±1.7*	153.0±5.0	2073.1±22.3
	Area	731.1±0.9*	750.8±1.8*	648.6±1.6	5461.6±15.9
	DRC	**0***	**0***	**0**	**0**
LaMPlace	Routed WL	7.49±1.21	7.59±0.51	7.33±0.29	63.92±2.14*
	WNS	-0.496±0.171	0.317±0.070	-3.028±0.065	-150.498±0.006*
	TNS	-1.259±0.767	**0.000**	**-0.003**	-38547.200±3.323*
	NVP	3896±1424	**0**	**1**	**259747***
	Power	256.9±2.2	160.7±2.8	153.3±2.7	2051.5±9.3*
	Area	732.0±6.9	751.2±1.3	648.7±1.1	5512.0±16.3*
	DRC	**0**	**0**	**0**	**0***
VP+DP 3.5 Flash	Routed WL	5.98±0.09	**6.35±0.04**	**5.58±0.02**	**44.57±0.93**
	WNS	**-0.339±0.026**	0.277±0.121	-2.965±0.014	**-149.585±0.029**
	TNS	**-0.549±0.039**	**0.000**	**-0.003**	**-38492.100±5.450**
	NVP	**2523±74**	**0**	**1**	**259747**
	Power	255.3±0.5	153.7±0.5	**146.4±0.5**	2012.5±7.8
	Area	**726.2±0.8**	**747.7±0.1**	**643.3±0.2**	**5343.9±6.2**
	DRC	**0**	**0**	**0**	**0**

D.4 Design Choice Ablations

Figure 8 shows the full ablation plots discussed in Section 5.3.



(a) Anchor weight λ_A (b) Context length C (c) Context selection strategy (d) Invalid suggestion rate

Figure 8: Ablations on `superblue1` with VeoPlace-guided DREAMPlace (Gemini 2.5 Flash, three seeds). Defaults: top stratified (TS), C=10, λ_A=0.01; main Superblue results use C=25. Lower anchor weights and longer context perform best; all context strategies improve over rollouts, with invalid suggestions converging to $\sim$20%.

D.5 Evolutionary Search Controls

The random-regions control replaces the VLM's proposals with regions drawn at random, matching the VLM arm in number of proposals per iteration, region format, and anchoring pipeline. The best-so-far-anchor control anchors each rollout to the best placement found so far with a soft spring at λ_A=0.01, a local perturbation of the best solution. Both controls use the same search loop and global λ_A=0.01 as VeoPlace and run 4,000 rollouts, twice the VeoPlace budget, on all eight Superblue designs with five seeds. VP+DP beats both controls on all eight designs (Table 16): random regions sharply degrade placement quality, while best-so-far anchoring performs comparably to DREAMPlace alone.

Table 16: VP+DP (Gemini 3.6 Flash) vs. DREAMPlace 4.3.0 (DP) and two evolutionary search controls. All columns report gHPWL ($\times 10^6$, lower is better), mean $\pm$ standard error across five seeds. DP 4.3.0 and both controls run 4,000 rollouts. VP+DP runs 2,000 rollouts. Best results are bolded.

Method (rollouts)	superblue1	superblue3	superblue4	superblue5	superblue7	superblue10	superblue16	superblue18
DP 4.3.0 (4,000)	376.61±0.12	453.64±0.27	290.90±0.25	399.94±0.25	544.36±0.31	714.05±1.30	366.21±0.07	219.13±0.05
VP+DP 3.6 Flash (2,000)	**372.85±0.34**	**438.14±0.42**	**286.82±0.55**	**397.90±0.41**	**538.76±0.86**	**683.25±1.22**	**364.43±0.09**	**217.50±1.09**
Random regions (4,000)	496.75±3.99	584.81±2.88	352.41±0.34	599.84±2.41	712.07±2.53	748.64±2.23	406.88±1.27	236.62±0.49
Best-so-far anchor (4,000)	382.79±0.72	451.09±1.20	289.09±0.43	406.20±0.74	544.93±0.28	702.34±2.69	366.55±0.11	221.35±0.31

D.6 Context Selection Strategies

We compare five strategies for selecting the C in-context examples from the history buffer:

Most Recent (FIFO). Select the C most recently generated placements from the population. This strategy implements pure evolutionary search, where the VLM observes a temporally ordered sequence of recent attempts.

Random. Randomly sample C placements from the population buffer. This provides a baseline that makes no assumptions about which examples are most valuable for evolutionary search.

Best Performing. Select the C placements with the lowest gHPWL from the population to encourage the VLM to replicate successful patterns.

Diverse. Represent each placement as a vector in $\mathbb{R}^{2T}$ by concatenating the (x_i, y_i) coordinates of its T macros. We then perform K-means clustering with C clusters on the population and select the placement with the best gHPWL from each cluster. This promotes geometric diversity across the selected examples while still favoring high-quality designs.

Top Stratified. Represent placements as coordinate vectors and cluster them by geometric similarity, as in the diverse strategy. We then focus on a single promising cluster by ranking all clusters by their best gHPWL and sampling one using a rank-based softmax distribution (with temperature $\tau = 0.43$). From the selected cluster, we choose the top C performing layouts, supplementing from the nearest

clusters by centroid distance if the selected cluster contains fewer than C members. This strategy provides a set of geometrically similar examples that represent variations of a particular design pattern.

D.7 Prompt Ablation

To isolate the impact of the prompt's high-level strategic guidance, we conducted an ablation study to test the sensitivity of the VLM's performance to its core instructions. A key question is whether the VLM performs best when asked to explore novel design configurations, to greedily refine known high-quality solutions, or to follow a more balanced default instruction.

To investigate this, we created two strategic variants of our main prompt: a **Greedy** prompt that explicitly instructs the VLM to make only minor modifications to the best-performing examples provided in-context, and an **Exploratory** prompt that encourages the VLM to disregard prior examples and generate creative, novel placements. Figure 9 shows the exact strategy-specific prompt text used by these three variants.

The results of this ablation, presented in Table 17 for the `superblue1` benchmark, show that prompt intent matters, but the strongest performance comes from the **default** prompt rather than from either extreme. The **Greedy** prompt is slightly worse, and the **Exploratory** prompt degrades further. This suggests that, for guiding DREAMPlace, the VLM works best when given balanced instructions that preserve useful inductive bias from prior examples without over-constraining the search.

Figure 9: Strategy-specific prompt text extracted from the implemented prompt templates. The **Greedy** prompt adds stay-close instructions, the **Default** prompt keeps the neutral scaffold, and the **Exploratory** prompt adds explicit novelty-seeking guidance.

Greedy Prompt	Default Prompt	Exploratory Prompt
Preamble addition: "Use the previous examples as a strong prior. Stay very close to the best known historical arrangements and prefer small, low-risk refinements over novel departures." *Strategy section:* "Refinement on Best Archetype: Propose exactly one small, low-risk improvement to the best known pattern. Stay very close to the successful historical arrangement and explain why this minimal change should help."	*Preamble addition:* none. *Strategy section:* "Improvement on Best Archetype: Propose exactly one concrete change to the best pattern (e.g., shift position, adjust spacing, try different canvas region); if evidence is weak, label it 'exploratory' but still commit."	*Preamble addition:* "Use the previous examples as context only, not as a constraint. Favor novel, diverse, and deliberately exploratory region proposals when they could plausibly lead to a better floorplan." *Strategy section:* "Deliberate Departure from Best Archetype: Propose exactly one concrete way your plan intentionally differs from the best known pattern. Explain why this novel configuration could outperform it; if evidence is weak, label it an 'exploratory hypothesis' but still commit."

Table 17: Ablation study on the VLM prompt's strategic guidance (TS, C=25) on the `superblue1` benchmark. We report VeoPlace with DREAMPlace (VP+DP) using Gemini 2.5 Flash, mean $\pm$ standard error across three seeds. The **Default** prompt yields the best objective ($\times 10^7$, lower is better). The best result is **bolded**.

Benchmark	Method	Greedy	Default	Exploratory
superblue1	VP+DP 2.5 Flash	37.51 $\pm$ 0.09	**37.47 $\pm$ 0.11**	37.64 $\pm$ 0.11
	DP 4.3.0 (Baseline)		37.95 $\pm$ 0.16	

D.8 Input Modality Ablation

We test how much each input channel contributes by removing either the chip canvas image or the textual context (the macro coordinates in text) from each in-context example, with top-stratified selection and $C{=}25$ fixed, on all eight Superblue designs with five seeds. The shuffled arm keeps the canvas images but pairs each one with a placement it did not come from. All four prompts improve over DREAMPlace 4.3.0 on all eight designs and differ from each other by under 0.8% (Table 18), so the gain does not depend on a single input channel. The shuffled arm matches the full prompt on every design, because the textual history already carries the exact macro coordinates.

Table 18: Input modality ablation on Superblue. The values are gHPWL ($\times 10^6$, mean $\pm$ standard error over five seeds, lower is better). VP+DP uses Gemini 3.6 Flash. 'Full' is the default multimodal prompt with both image and text. 'NoImg' removes the canvas image from every in-context example. 'NoTxt' removes the textual context from every in-context example. 'Shuffled' keeps the images but pairs each one with a placement it did not come from. DREAMPlace 4.3.0 runs 2,000 rollouts. The best result per design is in bold.

Benchmark	DP 4.3.0	VP+DP Full	VP+DP NoImg	VP+DP NoTxt	VP+DP Shuffled
superblue1	376.63±0.13	372.85±0.34	374.18±0.36	**372.73±0.39**	373.54±0.38
superblue3	453.75±0.33	438.14±0.42	438.31±0.73	**436.81±0.22**	438.01±0.72
superblue4	290.92±0.26	286.82±0.55	285.88±0.22	**284.64±0.59**	285.91±0.21
superblue5	400.88±0.30	**397.90±0.41**	397.91±0.53	397.99±0.96	398.42±0.39
superblue7	544.45±0.35	538.76±0.86	**537.35±0.57**	541.32±0.57	537.64±0.90
superblue10	714.47±1.18	683.25±1.22	684.01±1.31	682.72±0.51	**681.75±1.06**
superblue16	366.34±0.12	364.43±0.09	**363.48±0.39**	364.43±0.14	364.10±0.29
superblue18	219.24±0.07	217.50±1.09	217.77±0.87	**217.04±0.88**	217.61±1.07

E Prompt Details

E.1 Example Prompt

We show a representative prompt from the `superblue1` DREAMPlace track. Repeated macro rows are omitted for readability, but the prompt structure and wording match the prompts used in our experiments.

Prompt example: Superblue1

You are guiding a low-level macro placer for computer chip floorplanning. Your primary goal is to create a chip floorplan that minimizes wirelength (total interconnect length). Your task is to suggest rectangular regions for placing macros on the chip canvas, which has been divided into a grid. The low-level macro placer will choose the exact placement location within your suggested regions. If there is a macro in the netlist that you are not providing a suggestion for, the low-level macro placer will place that macro by itself.

The macros are grouped by colors based on their connectivity in the netlist graph, where macros with higher interconnectivity (more pin connections between them) are assigned similar colors. After macro placement, an analytical placer automatically positions the standard cells around them, so your macro placement decisions directly affect standard-cell placement quality.

This is a global optimization task where you need to consider:

- The impact of your suggested regions on macros that will be placed in the future.
- The overall arrangement of the selected macros that best supports a floorplan that minimizes wirelength (total interconnect length).

MACRO NAMES AND PROPERTIES FOR THIS NETLIST:

Macro	Color	WxH
I68	#80e020	28 x 212
DIL	#e020e0	28 x 212
IJW	#c0e020	28 x 212
H06	#a0e020	28 x 212
FZH	#c020e0	28 x 212
DXK	#e020e0	28 x 212

Macro	Color	WxH
EIQ	#20c0e0	28 x 212
ISU	#6020e0	28 x 212
FD4	#c0c020	28 x 212
HNI	#e0a020	28 x 212
CQH	#2020e0	28 x 212
EIJ	#2080e0	28 x 212

IMPORTANT PLACEMENT RULES:

1. The chip canvas is 512×512.
2. Coordinate system:
   - Origin `(0,0)` is at the bottom-left corner.
   - Top-left corner is `(0,512)`.
   - Bottom-right corner is `(512,0)`.
   - Top-right corner is `(512,512)`.
3. Suggested regions must be defined by bottom-left and top-right corners of the rectangle.
4. Suggested regions must not overlap with each other.
5. Suggestions are needed for these selected macros (name, size, color).

PLACEMENT QUALITY METRICS:

- Macro overlap must be zero (overlapping placements are invalid).

METRIC DIRECTIONS (REFERENCE ONLY):

- Lower wirelength is better.
- Lower congestion is better.
- Lower density is better.
- Higher WNS and TNS is better.

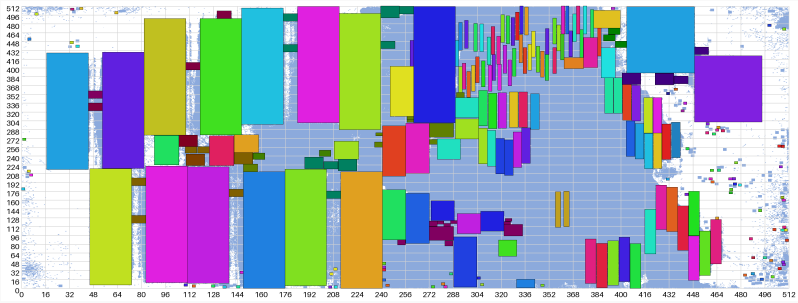
PREVIOUS PLACEMENT EPISODES:

In each episode's canvas image, the selected macros are outlined in red and labeled with their names. Blue texture represents standard cells placed by the analytical placer. Dark gray rectangles are fixed I/O terminals.

Episode #1

Position of Selected Macros:

- CQH: (262,300) to (290,512)
- EIJ: (148,0) to (176,212)
- DXK: (184,300) to (212,512)
- EIQ: (147,297) to (175,509)
- DIL: (83,10) to (111,222)
- ISU: (54,217) to (82,429)
- H06: (212,287) to (240,499)
- IJW: (46,6) to (74,218)
- I68: (176,4) to (204,216)
- HNI: (213,0) to (241,212)
- GKI: (119,278) to (147,490)
- FZH: (111,9) to (139,221)



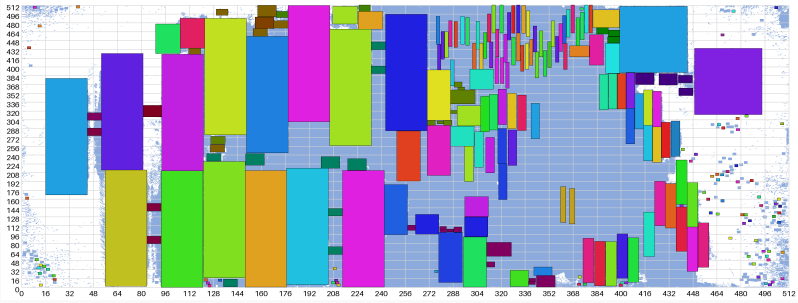
Final Metrics:

- Wirelength: 3.93×10^{8}
- Congestion: 9.36×10^{-1}
- Density: 1.04
- Macro overlap: 0

Episode #2

Position of Selected Macros:

- CQH: (243,284) to (271,496)
- EIJ: (150,244) to (178,456)
- DXK: (214,0) to (242,212)
- EIQ: (177,4) to (205,216)
- DIL: (178,300) to (206,512)
- ISU: (53,213) to (81,425)
- H06: (206,257) to (234,469)
- IJW: (122,277) to (150,489)
- I68: (121,17) to (149,229)
- HNI: (149,0) to (177,212)
- GKI: (93,0) to (121,212)
- FZH: (94,211) to (122,423)



Final Metrics:

- Wirelength: 4.20×10^{8}
- Congestion: 9.59×10^{-1}
- Density: 1.04
- Macro overlap: 0

IMPORTANT OUTPUT FORMAT:

1. All coordinates must be integers between 0 and 512.
2. All regions must have non-zero width and height ($x_2 > x_1$ and $y_2 > y_1$).
3. The orientation of macros cannot be changed. Do not try to rotate macros.

4. All regions must be large enough to fit the macro while still being within the bounds of the canvas. For example, if a macro size is 3.1×4.2, the region must be at least 4×5.
5. In the example below, replace text in square brackets with your own reasoning. Do not copy the text inside the brackets.
6. Follow this example format exactly (without the dashed lines).

HISTORY SUMMARY

- **Placement Archetypes by Color Group:**
  - [Identify 2–3 distinct placement strategies that appeared across episodes for major color groups]
  - [For each archetype: one representative episode + coordinates + wirelength]
  - [Note which strategy performed best and why]
- **Largest Macros Analysis:**
  - [Compare placement of largest macros in best vs worst episodes]
  - [Include coordinates and metrics only when there's clear performance difference]
- **Spatial Patterns:**
  - [Any adjacency, spacing, or positioning patterns that consistently helped performance]
  - [If no consistent pattern found, explain why patterns varied]
- **Standard Cell Distribution:**
  - [Observe light blue standard cells in canvas images - note where they cluster or get blocked]
  - [Which macro arrangements allowed standard cells to spread evenly vs get congested?]
- **Key Principles (at least 3):**
  - [Concise placement rules derived from analysis above]
  - [Cite evidence from history; if evidence is weak, mark as "hypothesis"]

STRATEGY

- **Overall Plan:** [2–3 sentences describing your placement approach]
- **Improvement on Best Archetype:** [Propose exactly one concrete change to the best pattern (e.g., shift position, adjust spacing, try different canvas region); if evidence is weak, label it "exploratory" but still commit]
- **Color Group Organization:** [How different color groups will be distributed across the canvas]
- **Wirelength Rationale:** [Why this arrangement minimizes wirelength]

REGIONS

- [List regions in DECREASING SIZE ORDER (largest macros first) to avoid overlaps]
- [Format: `MACRO_NAME: (x1,y1) to (x2,y2)`]

Example format:

- `ABC: (120,200) to (150,220)`
- `DEF: (45,310) to (65,326)`
- `RST: (280,150) to (288,162)`

Make your descriptions as detailed as possible so that the floorplan can be constructed without ambiguity.

E.2 Example Gemini Response

Abridged response excerpt from the same prompt.

HISTORY SUMMARY

- **Placement Archetypes by Color Group:**
  - **Large Macros** (28×212)**:** The dominant archetype for these macros is dense vertical stacking in columns, typically occupying the left side of the chip. In Episode #3 (WL: 3.92×10^8), these macros formed a very compact, multi-column block on the left. This pattern consistently yielded

better wirelength. In contrast, Episode #2 (WL: 4.20×10^8) showed more horizontal dispersion of these macros, leading to larger gaps and higher wirelength.

  - **Top-Right Block (HCL, O9Y -** 46×121**):** These two macros, often with distinct colors, were consistently placed together in the top-right corner. In Episode #3, O9Y (397,390) to (443,511) and HCL (452,316) to (498,437) were placed compactly, suggesting they are either I/O related or have high interconnectivity with each other and minimal with the main macro block.
  - **Medium/Small Macros:** These macros were generally used to fill the remaining spaces, often forming smaller clusters or extending the larger blocks.

- **Largest Macros Analysis:** The placement of the largest macros (28×212) was critical. In the best episode (#3, WL: 3.92×10^8), these macros were arranged into a very tight, rectangular block spanning almost the entire height of the left half of the chip. This dense packing minimized the overall footprint of the largest components, reducing their internal interconnect wirelength and creating a large, contiguous area for standard cells.

- **Key Principles:**
  1. **Maximize Macro Block Density:** Group the largest, most connected macros into a single, compact, and geometrically regular block to minimize inter-macro wirelength.
  2. **Utilize Chip Edges/Corners:** Place distinct macro groups, like HCL/O9Y, in corners to efficiently use peripheral space while maintaining a clear central area.
  3. **Fill Gaps with Smaller Macros by Color:** Systematically fill remaining spaces with smaller macros, prioritizing color-based clustering to localize connections and maintain overall floorplan density.

STRATEGY

- **Overall Plan:** My strategy is to create a highly dense and rectangular block of the largest macros (28×212) on the left side of the chip, spanning most of its height. The distinct HCL and O9Y macros will be placed compactly in the top-right corner. The remaining medium and small macros will then be systematically placed in contiguous blocks in the central and bottom-right regions, prioritizing color-based adjacency to minimize local interconnect.
- **Improvement on Best Archetype:** I will refine the "dense block" principle by ensuring the 28×212 macros are arranged in perfectly aligned vertical columns, minimizing any staggering or small gaps between them (exploratory). This aims for a more uniform macro-standard cell interface, which could improve routing predictability.
- **Wirelength Rationale:** This arrangement minimizes wirelength by physically co-locating highly connected macros, inferred by color and grouping. The dense packing of large macros ensures their internal connections are short.

REGIONS

- O9Y: (466,391) to (512,512)
- HCL: (420,391) to (466,512)
- FOF: (380,470) to (398,503)
- D8W: (375,391) to (385,446)
- BDJ: (385,391) to (395,446)
- EWA: (0,0) to (28,212)
- ISU: (0,212) to (28,424)
- IJW: (28,0) to (56,212)
- DIL: (28,212) to (56,424)
- FZH: (56,0) to (84,212)
- GKI: (56,212) to (84,424)
- I68: (84,212) to (112,424)
- EIJ: (84,0) to (112,212)
- HNI: (112,0) to (140,212)
- H06: (112,212) to (140,424)

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state that VeoPlace improves placement quality on open-source benchmarks using a VLM without fine-tuning, with specific win counts and peak improvement percentages that are substantiated by the experimental results in Section 5.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 states that the search loop optimizes only gHPWL and that post-route PPA is evaluated only after the search. It also covers the VLM latency and API cost, the 256-macro limit, and future work on more macros and multi-objective PPA optimization.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical results. Our contribution is an empirical framework evaluated through experiments.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We specify the VLM used for each experiment (Gemini 2.5, 3.5, or 3.6 Flash, and Qwen3-VL), all hyperparameters (context length, grid resolution, anchor weight, rollout count), random seeds, and benchmark sources. Full prompt templates and hyperparameter details are provided in the appendix. Code is included in the supplementary material.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is included in the supplementary material. All benchmarks used (ICCAD 2015 Superblue, ISPD 2005, ICCAD 2004, Ariane) are publicly available. Instructions for reproducing the experiments are provided with the code.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies the VLM, grid resolution, context length, rollout count, and random seeds. Full hyperparameter tables for both DREAMPlace and ChiPFormer settings are provided in the appendix.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: All main results report mean and standard error across random seeds (three or five, stated per table) with 2,000 rollouts each. The source of variability (random seed / initialization) and the error metric (standard error of the mean) are stated in each table caption.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report per-rollout wall-clock time and total runtime on a single A100 GPU for each Superblue benchmark (Table 8), along with VLM API cost estimates (Appendix B.5.2). The overhead ranges from 2%–7% over base DREAMPlace.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. We use publicly available benchmarks, a commercially available VLM API, and do not involve human subjects.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work improves macro placement in chip design, a specialized EDA task with no direct negative societal implications. It does not introduce new capabilities for surveillance, disinformation, or other harmful applications.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not release pre-trained models or scraped datasets. VeoPlace uses an existing commercial VLM API and publicly available EDA benchmarks, posing no risk for misuse.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the original papers for all benchmarks (ICCAD 2015, ISPD 2005, ICCAD 2004, Ariane), DREAMPlace, and ChiPFormer. URLs to public repositories are provided where applicable.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We release the VeoPlace code in the supplementary material with documentation for reproducing experiments.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or research with human subjects.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human subjects research.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The core method uses a Vision-Language Model (Gemini Flash models and Qwen3-VL) as a high-level planner for macro placement. The VLM, its role, and its usage are described in detail throughout Section 4.