GENERATIVE LATENT OPTIMIZATION FOR 3D FLOORPLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Floorplanning determines the positions and shapes of circuit blocks within a chip outline while minimizing interconnect length and satisfying geometric constraints such as non-overlap and boundary containment. 3D floorplanning extends this problem by adding cross-die block alignment, making the feasible layout space highly coupled and difficult to optimize. Existing approaches often use Reinforcement Learning for floorplanning problems which can require careful hyperparameter tuning and long training time. We instead introduce Latent Geometric Optimization (LGO), which learns a structured latent representation of complete floorplans and searches this representation after training. A Transformer VAE encodes block geometry and layout variables, an evolutionary optimizer searches the latent space, and the decoder maps the latent codes back to the geometric domain. Searching this learned layout manifold enables coordinated global changes within and across dies. On MCNC and GSRC 3D floorplanning benchmarks, our approach outperforms the best-performing baseline on 7 of the 8 tested benchmark instances, with a mean improvement of 7.5%, despite being trained on a relatively weak dataset. We further demonstrate that our model is able to effectively generalize to unseen circuits with unseen block counts. Results suggest that LGO's relative advantage grows as the number of relational constraints between objects increases.

1 INTRODUCTION

Floorplanning is an early stage of integrated-circuit (IC) design that determines the spatial organization of memory, logic, and other functional blocks. By determining the positions and shapes of these blocks, floorplanning establishes the spatial constraints for subsequent placement and routing (Adya & Markov, 2003). A good floorplan must use silicon area efficiently while supporting low power consumption and high circuit performance (Chao & Wong, 1995). Three-dimensional integration allows for further improvements in chip efficiency by stacking and vertically interconnecting multiple dies. Agnesina et al. (2023) reported a smaller footprint, improved power consumption, and performance compared with a 2D implementation of the same design. Vertical buses can require connected blocks on different dies to overlap in the vertical direction, reducing the length of the connections between them (Knechtel et al., 2015). In the floorplanning formulation studied in this paper, this is represented by a minimum projected overlap between selected blocks on different dies (Zhong et al., 2024).

Existing floorplanning and placement methods use different representations and optimization strategies. Classical methods combine simulated annealing with combinatorial layout representations, including sequence pairs (Murata et al., 1996), B*-trees (Chang et al., 2000b), and transitive closure graphs (Lin & Chang, 2001). Analytical approaches optimize continuous geometric variables and then legalize the invalid placements (Zhan et al., 2006; Li et al., 2022; Huang et al., 2023). Black-box optimization provides another alternative; for example, WireMask-BBO combines coordinate search with a greedy placement refinement stage (Shi et al., 2023). Reinforcement-learning approaches instead learn policies for sequential placement. For instance, GraphPlace uses graph representations to learn sequential macro placements (Mirhoseini et al., 2021).

Among these RL-based methods, MaskPlace uses spatial representations of placed blocks, legal positions, and incremental wirelength (Lai et al., 2022). Flexplanner extends this approach to soft-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

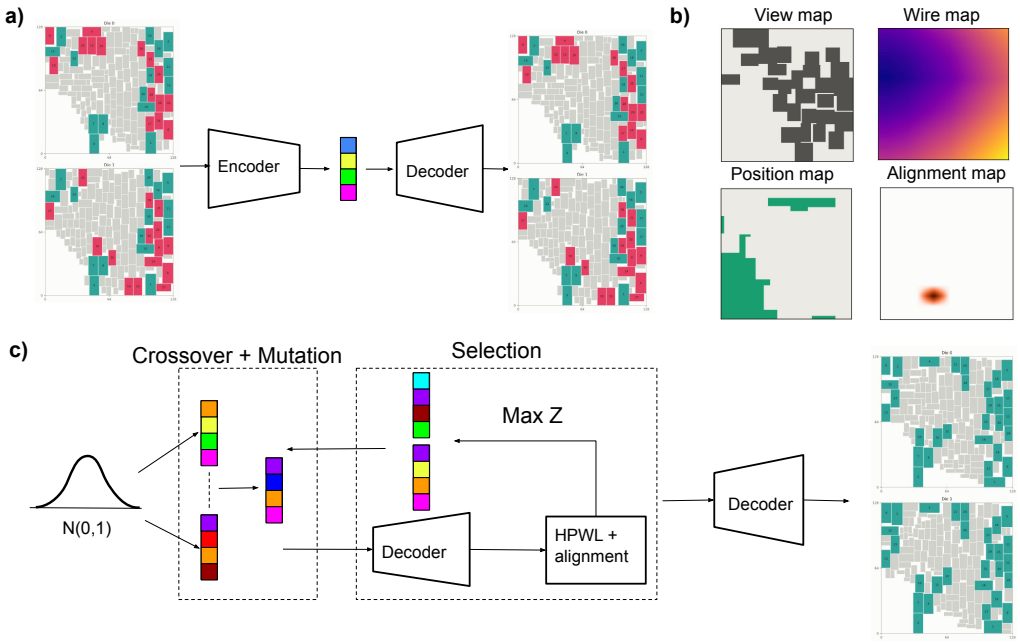


Figure 1: (**a**) Schematic diagram showing the variational autoencoder with an example of an input and the reconstructed output. (**b**) The autoregressive decoder is fed, at each decoding step t, the current spatial state. This includes a View map that shows in black where objects are placed, a wire map where a lighter color indicates lower HPWL if the current block would be placed there, a position map that highlights feasible placement cells in green, and an alignment map that shows where an object would align with its partner object. (**c**) Schematic diagram of the generation process. Parent latents are selected from high-scoring candidates and combined using crossover followed by Gaussian mutation. The resulting offspring are decoded and evaluated, after which the highest-scoring candidates survive to the next generation.

block shapes and cross-die alignment (Zhong et al., 2024). Despite these advances, RL-based methods have several practical limitations. Their results can be sensitive to random seeds and hyperparameter choices (Henderson et al., 2018). Training can also be time-consuming (Xue et al., 2024a). For 3D floorplanning, Flexplanner shows that sparse or poorly shaped rewards can lead to suboptimal solutions (Zhong et al., 2024).

Generative models provide a different approach: rather than learning a placement policy, they learn to generate candidate layouts. GraphPlanner uses a variational graph encoder–decoder to generate initial floorplans for subsequent refinement (Liu et al., 2022). Zheng et al. (2025) use a heterogeneous graph diffusion model to generate initial layouts, followed by analytical refinement. Other generative methods instead optimize the layout during generation. Lee et al. (2025) directly denoise macro coordinates and guide sampling using gradients from the overlap and wirelength objectives. MacroDiff uses diffusion to generate per-net wirelength targets, followed by a second stage that directly optimizes macro coordinates to match these targets while penalizing overlap (Yoon et al., 2025). More recent methods improve sampling efficiency and physical validity (Xie et al., 2026; Yoon et al., 2026).

These approaches operate primarily in the original solution space, where optimization acts directly on the layout variables. An alternative is to search in a learned latent space, where a change in the latent representation can affect multiple layout variables at once. In this setting, a generative model learns a lower-dimensional representation of candidate solutions, and the optimization is performed in this latent space. The decoder maps the resulting latent points back to the original solution space. Early applications of latent-space optimization use variational autoencoders (VAEs) to optimize molecular properties through gradient-based or Bayesian search (Gómez-Bombarelli et al., 2018; Kusner et al., 2017; Jin et al., 2018). Latent-space optimization has also been applied to neural

architecture search and the optimization of magnetic spin textures (Luo et al., 2018; Park et al., 2022). Closer to our approach, CVAE-Opt combines a conditional VAE, masked autoregressive decoding, and evolutionary search in the latent space for routing problems such as the traveling salesman problem (Hottung et al., 2021). To the best of our knowledge, prior IC floorplanning methods have not optimized complete layouts by searching a learned latent space.

We introduce *Latent Geometric Optimization* (LGO), a framework that searches a learned lower-dimensional representation of complete layouts. Figure 1 summarizes the approach. A Transformer VAE learns this latent space, which is then searched using an evolutionary optimizer. Each latent proposal is decoded into a floorplan with discrete block footprints and positions. During autoregressive decoding, the model is conditioned on both the latent representation and the current spatial state. This spatial state summarizes the partially constructed floorplan, including occupied and feasible regions and information about the objectives such as wire cost and cross-die alignment. Overlapping and out-of-bounds placements are masked during decoding. We evaluate LGO on the MCNC and GSRC 3D floorplanning benchmarks and find that it outperforms the best-performing baseline on 7 out of 8 benchmark instances. We also test zero-shot generalization by training on n100 and evaluating on the other seven benchmark instances, which have different block counts. Controlled 2D packing experiments further suggest that LGO's advantage over continuous optimization increases with the number of relational constraints between objects. For comparison, we evaluate LGO on a 2D irregular packing benchmark where non-overlap is the only inter-object constraint.

2 Problem Description

3D floorplanning. Let $\mathcal{B} = \{b_1, \ldots, b_n\}$ denote the circuit blocks, $\mathcal{T} = \{t_1, \ldots, t_m\}$ the fixed I/O terminals, and $\mathcal{N}$ the netlist. The blocks are distributed over a set of dies $\mathcal{D}$, each with fixed outline $\mathcal{C}(W, H) = [0, W] \times [0, H]$.

Each block b_i has width w_i, height h_i, area $a_i = w_i h_i$, bottom-left position (x_i, y_i), and a preassigned die d_i. Hard blocks have a fixed aspect ratio $AR_i = w_i/h_i$, while soft blocks satisfy

$$AR_i \in [AR_{\min}, AR_{\max}], \qquad w_i h_i = a_i.$$

The footprint of b_i is $\mathcal{P}_i = [x_i, x_i + w_i] \times [y_i, y_i + h_i]$. A feasible floorplan requires

$$\mathcal{P}_i \subseteq \mathcal{C}(W, H), \qquad \operatorname{int}(\mathcal{P}_i) \cap \operatorname{int}(\mathcal{P}_j) = \varnothing \quad \text{if } d_i = d_j. \tag{1}$$

Since d_i is fixed, the decision variables are the positions (x_i, y_i) and, for soft blocks, their aspect ratios.

For a cross-die alignment pair (b_i, b_j), the projected overlap is measured by

$$\begin{aligned} \operatorname{aln}_x(i,j) &= \left[\min(x_i + w_i, x_j + w_j) - \max(x_i, x_j)\right]_+, \\ \operatorname{aln}_y(i,j) &= \left[\min(y_i + h_i, y_j + h_j) - \max(y_i, y_j)\right]_+, \\ \operatorname{aln}(i,j) &= \min\left(1, \frac{\operatorname{aln}_x(i,j)\operatorname{aln}_y(i,j)}{\operatorname{aln}_m(i,j)}\right), \end{aligned} \tag{2}$$

where $[q]_+ = \max(0, q)$ and $\operatorname{aln}_m(i,j) > 0$ is the required overlap area. The total Higher values of $\operatorname{aln}(i,j)$ indicate better satisfaction of the required overlap..

Wirelength is approximated using half-perimeter wirelength (HPWL):

$$\mathrm{HPWL} = \sum_{e \in \mathcal{N}} \left(\max_{u \in e} x_u^c - \min_{u \in e} x_u^c + \max_{u \in e} y_u^c - \min_{u \in e} y_u^c \right), \tag{3}$$

where (x_u^c, y_u^c) is the center of a block or the fixed position of a terminal. The objective is therefore to obtain a feasible multi-die floorplan with high cross-die alignment and low HPWL.

3 Latent Geometric Optimization

We propose *Latent Geometric Optimization* (LGO), a framework for solving geometric constraint problems by searching a learned latent space. Our central idea is to search for complete layouts in a

learned latent space, instead of optimizing the object coordinates directly. A schematic diagram of our method is shown in Figure 1. The bidirectional encoder maps a layout X to a latent representation z. A decoder network maps the optimized latent representation z autoregressively back to the geometric domain. Optimization is performed in the latent space to improve z with respect to the specified objective. For each object i, the decoder first predicts the footprint (w_i, h_i) from a discrete set of options and then predicts an integer grid position (x_i, y_i), representing the lower-left corner of the block.

Spatial conditioning maps At decoding step t, the cell-prediction head is conditioned on a six-channel subset of spatial maps. Following MaskPlace (Lai et al., 2022), we represent the placement state using three spatial maps: a view map $f_v \in [0,1]^{|\mathcal{D}|\times W\times H}$, a normalized wire map $f_w^{(t)} \in [0,1]^{W\times H}$, and a position map $f_p^{(t)} \in [0,1]^{W\times H}$. The view map is instantiated once per die and $f_v^d[x,y] = 1$ when an already-placed block covers cell (x,y) of die d. The position map identifies feasible placements, with $f_p^{(t)}[x,y] = 1$ when the selected footprint of the current block fits in the die without overlap. The wire map, in turn, measures the normalized incremental HPWL associated with placing that block at (x,y). We additionally provide the normalized wire map $f_w^{(t+1)}$ of the next block as one-step lookahead. Finally, following FlexPlanner (Zhong et al., 2024), the projected-alignment map $f_a^{(t)} \in \mathbb{R}_{\geq 0}^{W\times H}$ records, at each candidate position, the projected intersection area between the current block and its alignment partner. This gives a total of six spatial maps: two die-specific view maps, the current and lookahead wire maps, one position map, and one alignment map.

These maps provide information about the current constraints and optimization objectives: the view and position maps identify occupied and legal regions, whereas the wire and alignment maps provide signals favoring low wirelength and cross-die alignment. Autoregressive decoding makes this representation possible. Each predicted placement updates the partial floorplan and net bounding boxes before the next block is decoded. It also allows direct enforcement of feasibility: whenever a feasible position exists, infeasible cell logits are masked. This reduces the search space and prevents overlapping or out-of-bounds placements from being selected.

Graph-aware attention. We expose circuit structure to both the bidirectional encoder and the autoregressive decoder through a learned additive attention bias. For every pair of block tokens (i,j), a relation tensor $R_{r,i,j}$ encodes five relations: the logarithm of the weighted number of shared nets, binary net adjacency, alignment partnership, membership of the same die, and self-identity. In layer ℓ and attention head h, the usual scaled dot-product logit is modified as

$$s_{i,j}^{(\ell,h)} = \frac{\langle q_i^{(\ell,h)}, k_j^{(\ell,h)} \rangle}{\sqrt{d_h}} + \sum_r \alpha_{\ell,h,r} R_{r,i,j}, \tag{4}$$

where the relation weights $\alpha_{\ell,h,r}$ are learned separately for every layer and head. The encoder applies this bias over all block pairs. At decoder step t, only relations $R_{r,t,j}$ with $j \leq t$ are used.

VAE training. The model is trained on floorplanning scenes obtained by running a FlexPlanner agent (Zhong et al., 2024). For a sampled latent $z = \mu_\phi(X) + \sigma_\phi(X) \odot \epsilon$, with $\epsilon \sim \mathcal{N}(0, I)$, the loss is

$$\mathcal{L}_{\text{VAE}} = \frac{1}{M} \sum_{t=1}^{M} \left[\text{CE}(p_t^{\text{cell}}, c_t) + \text{CE}(p_t^{\text{foot}}, o_t) \right] + \sum_{j=1}^{64} \max(\text{KL}_j, \lambda_{\text{fb}}), \tag{5}$$

where c_t and o_t are the teacher cell and footprint option and $\lambda_{\text{fb}} = 0.5$ nat is the free-bits threshold. Note that the HPWL and the alignment term are not present in Equation 5. The VAE learns to reconstruct floorplanning scenes, while the downstream latent optimization process finds the preferred scenes. This approach allows the user to change the objective as they deem necessary at search time, without needing to retrain the model.

Objective and population search Following Zhong et al. (2024), every generated scene is scored as follows

$$J = 1 - \text{HPWL}/\text{hpwl}_{\text{ref}} + 0.5A - 0.5O \tag{6}$$

where A is the aggregated normalized projected alignment and O the aggregated normalized overlap between objects. Since Flexplanner did not provide the hpwl_{ref} values, we approximated them in a way that allows for fair comparison between the two methods. Further details about this are in Appendix A.2. The reference values can be found in Table 10.

To optimize this objective we initialize a population of $P = 256$ latent vectors independently from $\mathcal{N}(0, I)$, then greedily decode them, and evaluate the scenes using Equation 6. In round r, the parents are sampled with replacement from the highest scoring half of the population. Each child is formed as

$$z' = \alpha z_a + (1 - \alpha) z_b + \sigma_r \epsilon, \qquad \alpha \sim \mathcal{U}(0.05, 0.95), \quad \epsilon \sim \mathcal{N}(0, I) \tag{7}$$

with cosine-annealed mutation

$$\sigma_r = 0.1 + \frac{0.6 - 0.1}{2} \left[1 + \cos\Big(\frac{\pi r}{200} \Big) \right]. \tag{8}$$

Each round generates 231 offspring and 25 samples from the prior. After decoding, the best 256 candidates from the combined parent and offspring populations survive. We run this process for 200 rounds for each of the benchmark instances. Additional methodological details are provided in Appendix A.

4 Experiments

4.1 3D Floorplanning

For 3D floorplanning, we evaluate on the MCNC [1] and GSRC [2] benchmarks against the baselines described in Appendix C. Code is available at `https://anonymous.4open.science/r/Latent-Geometric-Optimization-for-Floorplanning-506C`. The dataset was obtained by running inference using FlexPlanner's methodology on the benchmark instances. Our model was trained on a relatively weak dataset. The best training instances in the dataset had 10.58% higher HPWL and 8.97% lower alignment on average than our results; see Table 12. Yet, despite this weaker dataset, our latent search methodology is able to outperform FlexPlanner's published results on 7 out of 8 benchmarks.

For evaluation, we ran our latent search across five different seeds. All methods are evaluated on two separate metrics. In terms of HPWL, our method outperformed the baselines on 7 of the 8 instances, and on average our method improved the HPWL by 4.4% across all the benchmarks compared to the previous best method; see Table 1. With regard to the average alignment score, our method is able to outperform all other methods in 6 out of 8 instances, with an average improvement of 2.4%; see Table 2.

We approximated the objective function that Flexplanner used in order to be able to fairly compare to it, see Equation 6. Figure 2 shows, on the three benchmark instances with the most objects, the best J found over time by our latent search method. Our latent search is able to outperform on average the FlexPlanner baseline for the N100, N200, and N300 instances after 15.1s, 59.3s, and 15.8s of search, respectively. Flexplanner on matched hardware is able to produce its solutions on these instance in 4.6s, 6.2s, and 8.1s, respectively. Thus, while FlexPlanner reaches its reported solutions more quickly, our method is able to surpass their solution quality and with additional search time continuous to improve beyond the FlexPlanner results as the search budget increases. Table 3 shows that FlexPlanner obtained a mean J=0.4874 whereas our method obtained a mean J=0.5241 across the different benchmark instances, an improvement of 7.5% over the state of the art.

[1] https://vlsicad.eecs.umich.edu/BK/MCNCbench/

[2] https://vlsicad.eecs.umich.edu/BK/GSRCbench/

Table 3: HPWL and alignment score combined using Equation 6. Mean ± std over 5 runs.

Benchmark	FlexPlanner (Published)	Ours
ami33	0.4802	**0.5508 ± 0.0039**
ami49	0.4997	**0.5243 ± 0.0290**
n10	0.4658	**0.5152 ± 0.0117**
n30	0.4722	**0.5145 ± 0.0073**
n50	**0.5228**	0.4973 ± 0.0113
n100	0.5006	**0.5500 ± 0.0100**
n200	0.4734	**0.5097 ± 0.0061**
n300	0.4842	**0.5310 ± 0.0319**
Mean	0.4874	**0.5241**

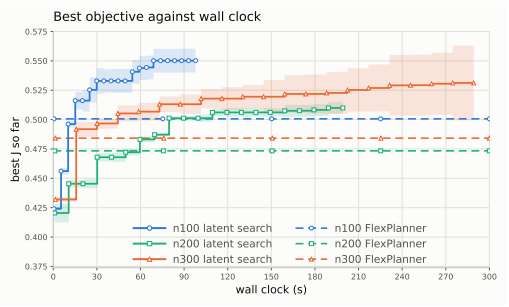


Figure 2: Best objective value found over time on n100, n200, and n300. Mean ± std over 5 runs.

Table 1: HPWL (the lower the better) comparison. The best observed results are shown in bold. Mean ± std over 5 runs. C/M means Circuit/Method. †Results reported in the FlexPlanner paper. *Reproduced FlexPlanner results, we compare to the numbers as published in their paper.

C/M	3D-B*-SA †	RL-CBL †	WireMask-BBO †	GraphPlace †	DeepPlace †	MaskPlace †	FlexPlanner*	FlexPlanner (Published) †	Ours
ami33	85,162±5,563	85,303±4,147	66,387±3,531	82,685±6,271	79,457±6,885	62,125±829	78,948±1,653	58,339±1,894	**56,080 ± 151**
ami49	1,400,787±68,043	1,338,219±98,556	1,100,891±95,467	1,455,872±80,468	1,356,203±56,434	1,128,110±90,645	1,141,221±43,458	762,712±12,878	**716,623 ± 10,776**
n10	35,230±115	34,805±1,132	33,046±36	36,520±869	34,371±817	33,648±1,096	31,915±420	29,781±75	**29,455±347**
n30	101,672±2,665	105,796±1,844	87,198±1,862	98,437±1,860	97,293±3,192	86,291±764	116,798±1,557	83,962±1,030	**82,745±286**
n50	132,421±3,123	156,113±4,079	111,878±3,371	136,980±2,219	126,910±2,836	113,145±407	116,798±1,153	**105,839±916**	109,687±1,215
n100	223,381±9,123	275,982±10,348	181,572±1,966	209,940±4,161	223,359±5,330	189,100±2,133	186,144±1,564	176,375±960	**169,834±1,716**
n200	422,060±9,101	572,649±37,831	325,453±3,488	402,650±6,117	418,348±5,134	375,250±2,939	348,223±1,678	316,199±1,080	**313,857±1,122**
n300	633,344±5,513	990,465±37,719	467,906±5,362	596,615±4,353	635,165±9,636	532,087±4,214	498,339±4,834	459,221±5,935	**426,204±4,320**
Avg.	379,257	444,917	296,791	377,462	371,388	314,969	314,798	249,053	**238,061**

Table 2: Alignment (the higher the better) comparison. The best observed results are shown in bold. Mean ± std over 5 runs. C/M means Circuit/Method. †Results reported in the FlexPlanner paper. *Reproduced FlexPlanner results.

C/M	3D-B*-SA †	RL-CBL †	WireMask-BBO †	GraphPlace †	DeepPlace †	MaskPlace †	FlexPlanner*	FlexPlanner (Published) †	Ours
ami33	0.550±0.058	0.132±0.038	0.179±0.091	0.207±0.067	0.286±0.051	0.300±0.017	0.886±0.028	0.905±0.017	**0.971±0.006**
ami49	0.438±0.099	0.107±0.043	0.222±0.082	0.265±0.063	0.180±0.056	0.218±0.052	0.798±0.036	**0.955±0.010**	0.886±0.051
n10	0.383±0.167	0.241±0.076	0.211±0.004	0.197±0.049	0.235±0.080	0.354±0.066	0.712±0.093	0.917±0.012	**0.994±0.004**
n30	0.537±0.159	0.108±0.040	0.288±0.051	0.233±0.039	0.287±0.074	0.511±0.125	0.707±0.051	0.920±0.024	**0.976±0.013**
n50	0.626±0.158	0.048±0.016	0.290±0.053	0.378±0.120	0.343±0.064	0.764±0.002	0.875±0.036	0.970±0.004	**0.989±0.005**
n100	0.131±0.051	0.016±0.008	0.195±0.034	0.279±0.050	0.332±0.073	0.575±0.046	0.876±0.019	0.961±0.017	**0.987±0.006**
n200	0.033±0.025	0.013±0.009	0.182±0.031	0.360±0.060	0.387±0.039	0.534±0.041	0.798±0.057	0.923±0.020	**0.981±0.010**
n300	0.009±0.009	0.005±0.005	0.205±0.038	0.383±0.023	0.399±0.031	0.533±0.023	0.697±0.058	**0.965±0.010**	0.915±0.061
Avg.	0.338	0.084	0.222	0.288	0.306	0.474	0.794	0.940	**0.962**

4.2 Zero-shot Transferability

We evaluate the zero-shot transferability of our method by three models, using three random seeds, on the n100 benchmark instance and evaluating it on the other seven unseen benchmark instances. Table 4 shows the ratio between the n100 model and our model trained jointly trained all benchmark instances. The model stays competitive on the other benchmark instances despite only being trained on one instance.

Table 4: Zero-shot transferability, training on circuit n100. Mean results over three training runs. *Ratio* is the zero-shot value divided by our per-circuit model. Bold ratios indicate better zero-shot performance.

Metric/Circuit		ami33	ami49	n10	n30	n50	n200	n300
Alignment (↑)	value	0.957	0.957	0.977	0.952	0.951	0.963	0.965
	ratio	0.986	**1.080**	0.983	0.975	0.962	0.982	**1.055**
HPWL (↓)	value	64,906	829,433	30,608	86,644	106,698	320,919	442,221
	ratio	1.157	1.157	1.039	1.047	**0.973**	1.023	1.038

4.3 ABLATION

We evaluate each component through an ablation study. We train each model using three random seeds on the n100 dataset. We then evaluate on both the N100 and the unseen during training N50 benchmark instances. Results are shown in Table 5. Notably, the wire map, alignment map, and graph attention have a relatively large effect on the n50 benchmark instance. This suggests that these components are necessary for generalization to unseen benchmark instances.

Table 5: Ablation of model inputs. Entries report the objective J.

Circuit / metric	w/o view map	w/o wire map	w/o position map	w/o alignment map	w/o graph attention	full
n100 (J)	$0.5261 \pm .006$	$0.5102 \pm .005$	$0.5283 \pm .009$	$0.4901 \pm .004$	$0.5263 \pm .005$	$\mathbf{0.5428 \pm 0.0121}$
n50 unseen (J)	$0.5150 \pm .008$	$0.4461 \pm .008$	$0.5072 \pm .009$	$0.4859 \pm .010$	$0.4623 \pm .008$	$\mathbf{0.5213 \pm 0.0087}$

4.4 EFFECT OF ADDITIONAL RELATIONAL CONSTRAINTS

We hypothesize that our approach becomes more competitive relative to other methods as the number relational constraints between objects increases since latent perturbations allow us to explore global changes, while our decoder enforces placement legality.

We introduce a simple controlled two-dimensional packing benchmark. The goal is to minimize the container area, while satisfying all constraints. We measure the utilization, which is the the total area of the rectangles divided by the area of their smallest axis-aligned bounding box and we vary along two axes: the number of constraint relations and the number of objects ($N \in 49, 100, 144$). We introduce three constraint families: proximity, relative ordering, and edge-alignment. These constraints are illustrated in Figure 3. For an instance containing N objects, we define four constraint-density levels: C_0 contains no additional relational constraints, while C_1, C_2, and C_3 contain approximatly $(N/4)$, $(N/2)$, and $(3N/4)$ relations, respectively. Further details and additional results are provided in Appendix E.

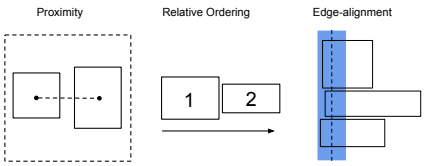
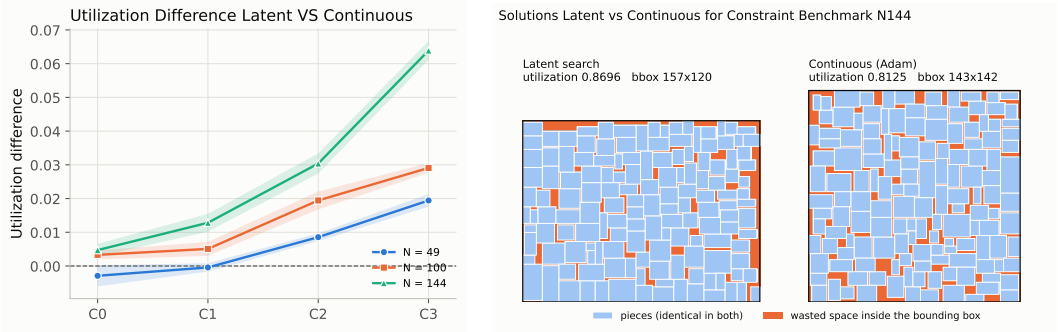


Figure 3: Illustration of the three types of constraints. The proximity constraint requires two object centers to be within a certain distance of each other. The relative ordering constraint requires that an object is to the left of another object. The edge-alignment constraint requires the bottom-left coordinate of an object to align with another object.

We compare LGO with a continuous coordinate space baseline. The purpose of this experiment is not to establish that our method is able to universally outperform continuous optimization; rather, we want to measure the utilization difference $\Delta U = U_{LGO} - U_{cont}$ as we vary the axes. As shown in Figure 4a, ΔU increases from C_0 to C_3 for every object count, with the largest increase for $N = 144$. This trend is consistent with our hypothesis.

Practical floorplanning problems can impose additional requirements that have not been evaluated in this paper, including fixed or pre-placed modules, edge or region constraints, mandatory grouping, and thermal limits (Young et al., 2002; Mallappa et al., 2024; Guan et al., 2023). Our results suggest that LGO may be particularly useful when several such requirements interact, although validation of this extension remains future work.

Irregular strip packing In order to evaluate how our method performs against established baselines on a problem with a relatively low number of relational constraints we evaluate on the public Dental benchmark. We additionally implement a continuous baseline that is similar in nature to the baseline in our previous experiment. The Dental benchmark is a 2D irregular shape strip packing problem where the only constraint is that objects are not allowed to overlap. All methods are evaluated on 128 unseen test instances. Methodological details are provided in Appendix F.



(a) Utilization difference.

(b) N144 constraint benchmark solutions.

Figure 4: (a) Utilization difference between latent search vs continuous search. As the number of constraints increases, so does the relative performance of the latent search compared to that of the continuous search. (b) Example solutions to the constraint benchmark by the latent search and the continuous baseline.

We found that the Dental benchmark requires a relatively fine resolution. We set up an experiment where we evaluated on three random dental benchmark instances. The search was given a fixed time budget of 240 seconds. As we increase the spatial resolution, the model is able to make more precise placements; however, this larger CNN head comes at the cost of fewer search rounds.

In order to allow operating at a finer resolution, we made a modification to the architecture by replacing the CNN head with a linear head, where the model only predicts discrete y-axis positions and rotation for the objects. The objects then automatically slide to the leftmost positions where there is no overlap. Figure 6 shows the utilization changes with the output resolution size for three instances from the Dental dataset. The mean results for the CNN head, the linear head, and the other baselines on all 128 benchmark instances can be found in Table 6. Our linear head approach is able to perform competitively against the continuous optimization baseline, while being able to outperform all other methods in terms of utilization.

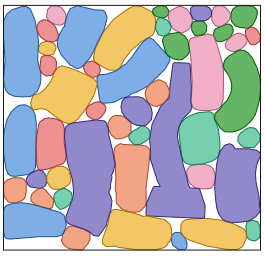


Figure 5: Solution to a Dental benchmark instance.

Table 6: Mean utilization and runtime on the Dental dataset. Note: For 9 out of 128 instances, GFPack++ was not able to generate any overlap-free scene, and the continuous optimization was not able to do this for one scene, even after a dedicated overlap cleanup phase. For these scenes, we set the utilization to 0 when calculating the mean utilization. †Results reported in GFPack++ paper (Xue et al., 2024b).

Algorithm	Mean Utilization (%)	Time (s)
XAtlas†	70.86	1.25
NFP†	66.30	8.22
SVGnest†	73.64	96.73
GFPack++ ($b = 512$)	71.48	10.69
Ours (CNN Head)	72.01	240
Ours (Linear Head)	79.53	240
Continuous Optimization	81.12	182.3

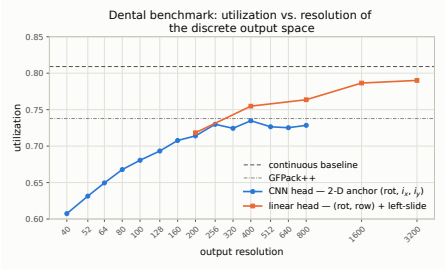


Figure 6: Utilization vs resolution results on the dental benchmark, mean over three test instances. The x-axis is the number of rows G_y. The CNN-head has a resolution of $G_x \times G_y$ where $G_x \approx 1.6G_y$. The y-axis ranges from 0.60 to 0.85. Both methods discretized the rotation into 64 bins.

5 CONCLUSION

We introduce Latent Geometric Optimization (LGO), a test-time optimization framework that searches for a complete layout in a learned latent space rather than directly modifying individual geometric variables. A Transformer VAE learns a global representation of floorplans, while an evolutionary optimizer searches this representation, allowing for coordinated changes across strongly coupled layout variables. On the MCNC and GSRC 3D floorplanning benchmarks, LGO outperforms the previous state of the art on 7 out of 8 instances and improves the combined objective of HPWL and alignment by 7.5% on average. It also generalizes well to unseen circuits and block counts. Results suggest that the method is particularly useful for problems with many relational constraints, where its ability to make coordinated global changes across multiple blocks may be especially beneficial.

AI USE STATEMENT

The authors wrote the manuscript and code. Generative AI was used as a tool, to assist with debugging, identify relevant literature that the authors subsequently read, and to refine sentence wording.

REFERENCES

Saurabh N. Adya and Igor L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6):1120–1135, 2003. doi: 10.1109/TVLSI.2003.817546.

Anthony Agnesina, Moritz Brunion, Jinwoo Kim, Alberto Garcia-Ortiz, Dragomir Milojevic, Francky Catthoor, Gioele Mirabelli, Manu Komalan, and Sung Kyu Lim. Power, performance, area, and cost analysis of face-to-face-bonded 3-D ICs. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 13(3):300–314, 2023. doi: 10.1109/TCPMT.2023.3264520.

Mohammad Amini, Zhanguang Zhang, Surya Penmetsa, Yingxue Zhang, Jianye Hao, and Wulong Liu. Generalizable floorplanner through corner block list representation and hypergraph embedding. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, pp. 2692–2702, New York, NY, USA, 2022. Association for Computing Machinery. doi: 10.1145/3534678.3539220.

Julia A Bennell and Xiang Song. A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums. *Computers & Operations Research*, 35(1):267–281, 2008.

Edmund Burke, Robert Hellier, Graham Kendall, and Glenn Whitwell. A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations research*, 54(3): 587–601, 2006.

Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th annual design automation conference*, pp. 458–463, 2000a.

Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the 37th Design Automation Conference*, pp. 458–463, 2000b. doi: 10.1145/337292.337541.

Kai-Yuan Chao and D. F. Wong. Floorplanning for low power designs. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pp. 45–48. IEEE, 1995. doi: 10.1109/ISCAS.1995.521447.

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.

Wesley Cox, Lyndon While, and Mark Reynolds. A review of methods to compute minkowski operations for geometric overlap detection. *IEEE Transactions on Visualization and Computer Graphics*, 27(8):3377–3396, 2021. doi: 10.1109/TVCG.2020.2976922.

Paul Falkenstern, Yuan Xie, Yao-Wen Chang, and Yu Wang. Three-dimensional integrated circuits (3D IC) floorplan and Power/Ground network co-synthesis. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 169–174. IEEE, 2010. doi: 10.1109/ASPDAC.2010.5419899.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. doi: 10.1021/acscentsci.7b00572.

Wenbo Guan, Xiaoyan Tang, Hongliang Lu, Yuming Zhang, and Yimen Zhang. Thermal-aware fixed-outline 3-D IC floorplanning: An end-to-end learning-based approach. *IEEE Transactions on Very Large Scale Integration Systems*, 31(12):1882–1895, 2023. doi: 10.1109/TVLSI.2023.3321532.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 3207–3214, 2018. doi: 10.1609/aaai.v32i1.11694.

André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2021.

Fuxing Huang, Duanxiang Liu, Xingquan Li, Bei Yu, and Wenxing Zhu. Handling orientation and aspect ratio of modules in electrostatics-based large scale fixed-outline floorplanning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023. doi: 10.1109/ICCAD57390.2023.10323841.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2323–2332. PMLR, 2018.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Johann Knechtel, Evangeline F. Y. Young, and Jens Lienig. Planning massive interconnects in 3-D chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1808–1821, 2015. doi: 10.1109/TCAD.2015.2432141.

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1945–1954. PMLR, 2017.

Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast chip placement via reinforced visual representation learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24019–24030. Curran Associates, Inc., 2022. doi: 10.52202/068431-1744.

Vint Lee, Minh Nguyen, Leena Elzeiny, Chun Deng, Pieter Abbeel, and John Wawrzynek. Chip placement with diffusion models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 33499–33514. PMLR, 2025.

Ximeng Li, Keyu Peng, Fuxing Huang, and Wenxing Zhu. PeF: Poisson's equation based large-scale fixed-outline floorplanning. *arXiv preprint arXiv:2210.03293*, 2022.

Jai-Ming Lin and Yao-Wen Chang. TCG: A transitive closure graph-based representation for non-slicing floorplans. In *Proceedings of the 38th Design Automation Conference*, pp. 764–769, 2001. doi: 10.1145/378239.379062.

Jai-Ming Lin, Yao-Wen Chang, and Shih-Ping Lin. Corner sequence-a p-admissible floorplan representation with a worst case linear-time packing scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):679–686, 2003.

Yiting Liu, Ziyi Ju, Zhengming Li, Mingzhi Dong, Hai Zhou, Jia Wang, Fan Yang, Xuan Zeng, and Li Shang. GraphPlanner: Floorplanning with graph neural network. *ACM Transactions on Design Automation of Electronic Systems*, 28(2):21:1–21:24, 2022. doi: 10.1145/3555804.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. Local optimization for robust signed distance field collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(1), 2020. doi: 10.1145/3384538.

Uday Mallappa, Hesham Mostafa, Mikhail Galkin, Mariano Phielipp, and Somdeb Majumdar. FloorSet: A VLSI floorplanning dataset with design constraints of real-world SoCs. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 118:1–118:9, 2024. doi: 10.1145/3676536.3676814.

L. Martínez, R. Andrade, E. G. Birgin, and J. M. Martínez. PACKMOL: A package for building initial configurations for molecular dynamics simulations. *Journal of Computational Chemistry*, 30(13):2157–2164, 2009. doi: 10.1002/jcc.21224.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021. doi: 10.1038/s41586-021-03544-w.

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.

Ralph H. J. M. Otten. Automatic floorplan design. In *Proceedings of the 19th Design Automation Conference*, pp. 261–267, 1982. doi: 10.1109/DAC.1982.1585510.

S. M. Park, H. G. Yoon, D. B. Lee, J. W. Choi, H. Y. Kwon, and C. Won. Optimization of physical quantities in the autoencoder latent space. *Scientific Reports*, 12:9003, 2022. doi: 10.1038/s41598-022-13007-5.

Jack Qiao. Svgnest. `https://github.com/Jack000/SVGnest`, 2017. GitHub repository.

Yunqi Shi, Ke Xue, Lei Song, and Chao Qian. Macro placement by Wire-Mask-Guided black-box optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 6825–6843. Curran Associates, Inc., 2023. doi: 10.52202/075280-0299.

Salvatore Torquato and Yang Jiao. Dense packings of the Platonic and Archimedean solids. *Nature*, 460(7257):876–879, 2009. doi: 10.1038/nature08239.

Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

Lijun Wei, Wee-Chong Oon, Wenbin Zhu, and Andrew Lim. A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research*, 215(2): 337–346, 2011. doi: 10.1016/j.ejor.2011.06.022.

D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pp. 101–107, 1986. doi: 10.1109/DAC.1986.1586075.

Peng Xie, Ke Xue, Yunqi Shi, Ruo-Tong Chen, Chengrui Gao, Siyuan Xu, Chenjian Ding, Mingxuan Yuan, and Chao Qian. Flowplace: Flow matching for chip placement. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, 2026. doi: 10.1145/3770743.3804343.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. doi: 10.52202/079017-4462.

Tianyang Xue, Lin Lu, Yang Liu, Mingdong Wu, Hao Dong, Yanbin Zhang, Renmin Han, and Baoquan Chen. Gfpack++: Improving 2d irregular packing by learning gradient field with attention. *arXiv preprint arXiv:2406.07579*, 2024b.

Jongho Yoon, Jinsung Jeon, and Seokhyeong Kang. Late breaking results: A geometric diffusion model for macro placement generation. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–2, 2025. doi: 10.1109/DAC63849.2025.11132593.

Jongho Yoon, Jinsung Jeon, and Seokhyeong Kang. Physics-guided geometric diffusion for macro placement generation. In *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence*, pp. 6609–6616, 2026. doi: 10.24963/ijcai.2026/735.

Evangeline F. Y. Young, Chris C. N. Chu, and M. L. Ho. A unified method to handle different kinds of placement constraints in floorplan design. In *Proceedings of the Seventh Asia and South Pacific Design Automation Conference*, pp. 661–667, 2002. doi: 10.1109/ASPDAC.2002.995011.

Jonathan Young. Xatlas. `https://github.com/jpcy/xatlas`, 2022. GitHub repository.

Yong Zhan, Yan Feng, and Sachin S. Sapatnekar. A fixed-die floorplanning algorithm using an analytical approach. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pp. 771–776, 2006. doi: 10.1145/1118299.1118477.

Xinglin Zheng, Hao Gu, Keyu Peng, Youwen Wang, Wenxing Zhu, and Ziran Zhu. Late breaking results: Customized diffusion model empowered by heterogeneous graph network for effective floorplanning. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–2, 2025. doi: 10.1109/DAC63849.2025.11133070.

Ruizhe Zhong, Xingbo Du, Shixiong Kai, Zhentao Tang, Siyuan Xu, Jianye Hao, Mingxuan Yuan, and Junchi Yan. Flexplanner: flexible 3d floorplanning via deep reinforcement learning in hybrid action space with multi-modality representation. *Advances in Neural Information Processing Systems*, 37:49252–49278, 2024.

A Additional Methodological Details

A.1 Architecture

The configurations for the model trained on all eight circuits can be found in Table 7 and the configurations for the model only trained on the N100 dataset can be found in Table 8.

Encoder Every block is described by eight normalized values: x, y, width, height, area ratio, die index, a binary alignment-partner indicator, and the nominal area relative to the largest block. Each of these values is linearly projected and then added together with the positional embeddings. The encoder is implemented using a standard Transformer architecture consisting of layer normalization (Xiong et al., 2020) and multi-head self-attention blocks.

Decoder At step t, the decoder token is the sum of the current block embedding, a learned positional embedding, a state embedding, and a projection of z. The block embedding is constructed from the 12 static features in the first part of Table 9. The state embedding is constructed from the second part of the table. It summarizes the preceding placement, identifies the current decision, and records placement progress on every die.

Table 7: Selected VAE configuration for Floorplanning benchmark trained on the eight circuit dataset.

Hyperparameter	Value
Latent dimension	64
Transformer width	256
Attention heads	8
Encoder / decoder layers	4 / 6
Feed-forward width	1,024
Spatial channels	6
Output resolution	128×128
Free bits per latent dimension	0.5 nat
KL weight	1.0
Optimizer	AdamW (β_1=0.9, β_2=0.95, wd 0.01)
Peak learning rate	3×10^{-4}
Linear warmup steps	500
LR schedule	cosine decay to a 10% floor over 6,000 steps
Training steps	6,000
Selected checkpoint	step 2,400 (max validation prior-J)

Table 8: Selected VAE configuration for Floorplanning benchmark trained on the N100 circuit dataset.

Hyperparameter	Value
Latent dimension	64
Transformer width	256
Attention heads	8
Encoder / decoder layers	4 / 6
Feed-forward width	1,024
Spatial channels	6
Output resolution	128×128
Free bits per latent dimension	0.5 nat
KL weight	1.0
Optimizer	AdamW (β_1=0.9, β_2=0.95, wd 0.01)
Peak learning rate	3×10^{-4}
Linear warmup steps	250
LR schedule	cosine to a 10% floor over a 6,000-step horizon
Training steps	1000
Selected checkpoint	step 800

Table 9: Features used to construct the decoder's block and state embeddings. Integer-valued geometry is normalized and presented to the network as real-valued inputs.

Feature	**Encoded value**	**Type**	**Interpretation**
Block embedding: 12 static features			
Nominal area	a_i/A_G	Real, normalized	Fraction of one die occupied by the block.
Initial width	$w_i^{(0)}/G_x$	Real, normalized	Initial block width as a fraction of die width.
Initial height	$h_i^{(0)}/G_y$	Real, normalized	Initial block height as a fraction of die height.
Assigned die	d_i	Categorical scalar	Die index stored as one scalar rather than one-hot encoded.

Continued on next page

Table 9 (continued)

Feature	Encoded value	Type	Interpretation
Has alignment partner	$\mathbf{1}[p(i) \neq \varnothing]$	Binary scalar	Indicates whether the block has an alignment partner.
Partner area	$\mathbf{1}[p(i) \neq \varnothing] a_{p(i)}/A_G$	Real, normalized	Partner area. Zero when no partner exists.
Net degree	$\lvert\mathcal{N}(i)\rvert/32$	Count, normalized	Number of nets incident to the block.
Decode-order position	π_i/M	Ordinal, normalized	Relative placement position in $[0, 1)$.
Circuit size	$M/300$	Real, normalized	Number of blocks. Constant within a circuit.
Assigned-die utilization	$\frac{\sum_{j:d_j=d_i} a_j}{A_G}$	Real, normalized	Total nominal area assigned to the same die.
Logarithmic area	$\max\{\log(a_i/A_G), -8\}/8$	Real, log-scaled	Log-scaled area, clipped to avoid extreme values.
Within-die area rank	$r_i/\max(1, n_{d_i})$	Ordinal, normalized	Relative area rank. $r_i = 0$ denotes the largest block on its die.
State embedding: $9 + 2D$ *dynamic and step-specific features*			
Previous position	$\left(x_{i_{t-1}}/G_x,\, y_{i_{t-1}}/G_y\right)$	Real 2-vector, normalized	Position selected at the preceding step. Zero at $t = 0$.
Previous footprint	$\left(w_{i_{t-1}}/G_x,\, h_{i_{t-1}}/G_y\right)$	Real 2-vector, normalized	Realized footprint selected at the preceding step. Zero at $t = 0$.
Previous die	$d_{i_{t-1}}$	Categorical scalar	Die of the preceding block. Zero at $t = 0$.
Previous realized area	$\widetilde{a}_{i_{t-1}}/a_{\max}$	Real, normalized	Realized area of the preceding block. Zero at $t = 0$.
Current decode step	t/M	Ordinal, normalized	Progress through the global placement sequence.
Current die	d_{i_t}	Categorical scalar	Assigned die of the block currently being placed.
Current nominal area	$a_{i_t}/a_{\max}$	Real, normalized	Nominal area of the current block relative to the largest block.
Per-die blocks processed	$(q_{t,d}/n_d)_{d=0}^{D-1}$	Real D-vector, normalized	Fraction of each die's block queue processed before step t.
Per-die area filled	$\left(\frac{\sum_{s<t:\, d_{i_s}=d} \widetilde{a}_{i_s}}{A_G}\right)_{d=0}^{D-1}$	Real D-vector, normalized	Fraction of each die occupied by previously placed blocks.

The static features are concatenated into $f_i \in \mathbb{R}^{12}$ and transformed by a shared MLP, whereas the state features are concatenated into $s_t \in \mathbb{R}^{9+2D}$ and linearly projected:

$$e_i^{\text{block}} = W_2\ \text{GELU}(W_1 f_i + b_1) + b_2, \qquad \mathbb{R}^{12} \to \mathbb{R}^{128} \to \mathbb{R}^{256}, \tag{9}$$

$$e_t^{\text{state}} = W_s s_t + b_s, \qquad \mathbb{R}^{9+2D} \to \mathbb{R}^{256}. \tag{10}$$

For the 3D Floorplanning experiments, $s_t \in \mathbb{R}^{13}$. The complete decoder token is

$$u_t = e_{i_t}^{\text{block}} + e_t^{\text{pos}} + e_t^{\text{state}} + (W_z z + b_z). \tag{11}$$

Six causal Transformer blocks process the token sequence and use key/value caches during free decoding.

Decoding order The decode order is deterministic. The alignment pairs are sorted by decreasing overlap area and are placed consecutively. For each pair, the one on the lower-numbered die is placed first. Unpaired blocks are placed in the queue based on decreasing area.

Footprint head For every block, the footprint head produces one logit for each legal candidate footprint. The (w, h) candidates are represented by $(w/128, h/128, \log(w/h), wh/a_i)$ and mapped to a 256-dimensional footprint embedding. The contextualized token embedding is mapped through a learned linear layer to a 256-dimensional query. The logit of each candidate is computed from the scaled dot product between its footprint embedding and this query. Invalid logits are masked. If no option is valid, the decoder uses a least-overlap fallback option.

Cell head The chosen footprint embedding is added to the contextualized token embedding. A fully connected layer then projects the 256-dimensional vector onto a $16 \times 8 \times 8$ feature grid, which is progressively upsampled by four convolutional layers into six 128×128 global feature maps. A separate convolutional tower produces the spatial maps. Fusing both towers produces one logit per grid cell.

A.2 Benchmark objective reference values

The main method we are comparing to is Flexplanner (Zhong et al., 2024). The authors report alignment, HPWL, and overlap in separate tables, but never combine them into a single number. In order to fairly compare the performance of both methods, we adopt the reward they use to train their RL agent and then compare the models on this reward. FlexPlanner's terminal reward (Alg. 2) is $r_T = w_a \cdot \mathrm{aln} - w_o \cdot o - w_l \cdot \mathrm{HPWL}$ with published weights $w_a = w_o = 0.5$ and $w_l = 1.0$ (Table 8), which matches our objective Equation 6. We adopt the HPWL normalization from Flexplanner, only differing in how the constant is fixed. Flexplanner estimates $\mathrm{hpwl}_{\mathrm{ref}}$ per circuit by rolling out the untrained policy and taking the mean HPWL, so the scale of the reward varies from run to run. We instead use a fixed constant, chosen using FlexPlanner's published HPWL values, which makes the objective reproducible across runs and identical for both methods. The reference values can be found in Table 10.

Table 10: Per-circuit wirelength reference constants $\mathrm{hpwl}_{\mathrm{ref}}$ which are used to normalise the objective J. Each constant is fixed just above the FlexPlanner's published circuit's HPWL, so that $\mathrm{HPWL}/\mathrm{HPWL}_{\mathrm{ref}} \approx 1$. The same constant is applied to every method on a given circuit. Importantly, J can never be compared across circuits.

Suite	Benchmark	$\mathrm{HPWL}_{\mathrm{ref}}$	FlexPlanner HPWL	Ratio
MCNC	ami33	60,000	58,339	0.972
	ami49	780,000	762,712	0.978
GSRC	n10	30,000	29,781	0.993
	n30	85,000	83,962	0.988
	n50	110,000	105,839	0.962
	n100	180,000	176,375	0.980
	n200	320,000	316,199	0.988
	n300	460,000	459,221	0.998

A.3 Hardware

All training and inference were performed on a single A100 80GB GPU.

B Benchmark statistics and training dataset

B.1 Benchmark statistics

All 3D floorplanning methods are evaluated on the public MCNC and GSRC benchmarks. The statistics of these benchmarks are shown in Table 11.

Table 11: Statistics of the MCNC and GSRC benchmarks. Alignment denotes the number of blocks with an alignment partner.

circuit	block	I/O port	net	alignment
ami33	33	40	121	20
ami49	49	22	396	20
n10	10	69	118	10
n30	30	212	349	20
n50	50	209	485	30
n100	100	334	885	60
n200	200	564	1585	60
n300	300	569	1893	60

B.2 FLOORPLANNING DATASET

The model was trained on a dataset that we obtained from running the FlexPlanner agent on the benchmark instances. It is important to note that we do not have access to the weights the authors of FlexPlanner used. Therefore, we had to train the agent from scratch. However, after carefully training the FlexPlanner agent, using the exact hyperparameters mentioned in their paper, its performance lagged behind the published performance.

For each benchmark instance we created 12,000 overlap-free samples by sampling each action from the policy's output distributions rather than taking the argmax, so that repeated rollouts of the same instance yield diverse layouts. The total dataset contained 96,000 samples across eight benchmark instances. Table 12 shows for each benchmark instance the HPWL and the alignment of the sample with the greatest J objective value in the dataset, see Equation 6, together with how our search method was able to improve beyond these results.

Table 12: For each benchmark instance, the training sample with the highest objective value J in the dataset. The difference in HPWL and alignment is compared to our search method trained on this dataset.

Benchmark	Best HPWL in dataset	HPWL difference vs. ours	Best alignment in dataset	Alignment difference vs. ours
ami33	63,696	+13.58%	0.90985	+6.30%
ami49	833,735	+16.34%	0.84775	+4.32%
n10	30,097	+2.18%	0.84766	+14.72%
n30	102,055	+23.34%	0.81564	+16.43%
n50	113,989	+3.92%	0.93997	+4.96%
n100	178,727	+5.24%	0.90939	+7.86%
n200	337,619	+7.57%	0.91522	+6.71%
n300	479,510	+12.51%	0.81891	+10.50%
Mean	–	+10.58%	–	+8.97%

C 3D FLOORPLANNING BASELINES

In Section 4.1 we compare our method to seven baselines. Zhong et al. (2024) adapted these methods to the 3D floorplanning setting to allow for fair comparison.

3D-B*-SA (Falkenstern et al., 2010) Using B*-trees (Chang et al., 2000a) and simulated annealing (Van Laarhoven & Aarts, 1987) this heuristic approach optimizes floorplanning instances. At each iteration, a new state is generated by perturbing the current B*-tree. After the search process is completed, the resulting B*-tree is decoded into the corresponding floorplan. In order to adapt the method to 3D floorplanning, an alignment score is added into the cost function.

RL-CBL (Amini et al., 2022) combines Reinforcement Learning with a heuristic-based Corner Block List (CBL) method (Lin et al., 2003). A policy network determines how the current floorplan should be perturbed at each step. For the 3D floorplanning setting, the alignment score is incorporated into the reward. However, RL-CBL cannot handle soft blocks with variable aspect ratios.

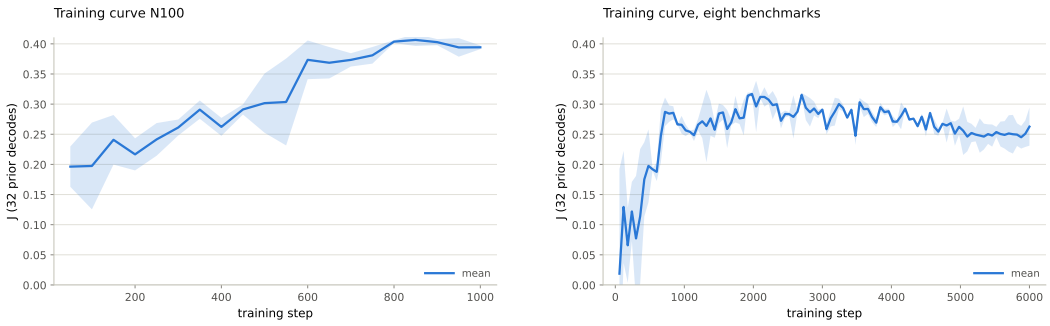
WireMask-BBO (Shi et al., 2023) is a black-box optimization framework that employs a wire-mask-guided (Lai et al., 2022) greedy procedure for objective evaluation. At each placement step, a position mask first eliminates invalid locations that would cause overlap between blocks or boundary violations. Among the remaining valid locations, the position that produces the smallest increase in HPWL is selected. For the 3D floorplanning setting, an additional alignment mask is added to help ensure alignment. Similar to RL-CBL, however, Wiremask-BBO does not support soft blocks with variable aspect ratios.

GraphPlace (Mirhoseini et al., 2021), DeepPlace (Cheng & Yan, 2021), MaskPlace (Lai et al., 2022) are three Reinforcement Learning approaches for 2D floorplanning. They do not consider variable aspect ratios for soft blocks. For the 3D floorplanning task they are extended by providing an alignment mask and incorporating the alignment into the reward function.

FlexPlanner (Zhong et al., 2024) is a Reinforcement Learning method specifically designed for 3D floorplanning, it directly optimizes block position, layer, and aspect ratio. It uses a multi-model state representation combining vision, graph, and sequence information. Alignment and position masks are applied to filter out invalid placements.

D TRAINING CURVES

Training the N100 only model for 1000 steps took on average 25 minutes. Whereas training the all circuit model for 6000 steps took on average 153 minutes. For evaluation, we selected the n100 checkpoint at step 800 and the model trained on all eight circuits at step 2,400. Resulting in an effective average training time of 19 minutes and 61 minutes. Figure 7 shows the mean and standard deviation of the training curves. For each dataset, we trained three separate models using different random seeds.



(a) (b)

Figure 7: (a) Mean training curve for three models trained exclusively on the N100 dataset. The y-axis shows the mean objective value for 32 prior decodes. (b) Mean training curve for three models trained on all eight datasets. The y-axis shows the mean objective value for 32 prior decodes averaged across all 8 benchmark instances.

E CONSTRAINED RECTANGLE-PACKING BENCHMARK

Each instance consists of $N \in \{49, 100, 144\}$ axis-aligned rectangles. A rectangle i has an integer width and height $s_i = (w_i, h_i)$ and is placed using integer lower-left corner $p_i = (x_i, y_i)$. The rectangles may not overlap. The rectangles have total area k^2 with $k = 128$ and are allowed to be placed inside a $G \times G$ container with $G = \lceil 1.5k \rceil = 192$.

The objective is to maximize the utilization of the smallest axis-aligned bounding box. The width and the height of the bounding box are $W = \max_i(x_i + w_i) - \min_i x_i$ and $H = \max_i(y_i + h_i) - \min_i y_i$, respectively. The utilization is therefore

$$U(p) = \frac{k^2}{W(p)H(p)} \tag{12}$$

a higher utilization corresponds to a more compact arrangement of the rectangles.

Proximity Two rectangles i and j are considered within distance when the centers satisfy $\max(|x_j - x_i|, |y_j - y_i|) \leq d$, with $d = 32$. Proximity is pairwise, but a rectangle may be paired with several others.

Relative ordering Rectangle j is constrained to lie to the right rectangle i by requiring $x_j \geq x_i + w_i$, where $x_i + w_i$ is the right edge of a rectangle i. The ordering groups are arranged in chains containing at most four rectangles.

Edge-alignment Rectangles can be both aligned horizontally and vertically. Two rectangles are x-aligned when $|x_j - x_i| \leq \tau$, and are y-aligned when $|y_j - y_i| \leq \tau$, with $\tau = 4$. The edge-alignment groups contain at most four rectangles. One of the rectangles is chosen as the reference and the others must align with it.

A layout is deemed feasible when all constraints are satisfied.

Instance construction We generate a solution first and then hide it from the solver. Following the standard slicing-floorplan model (Otten, 1982; Wong & Liu, 1986), we recursively split a rectangle using horizontal or vertical guillotine cuts until N rectangles remain. This produces a layout $p^\star$ with no gaps or overlaps, meaning that $U(p^\star) = 1$.

All constraints are generated from this layout. A proximity constraint is only added if two rectangles have a distance of at least $0.5d$. An ordering constraint is only added if the left-to-right order holds in the layout. An edge-alignment constraint is only added between the rectangles if they share the same left-edge or bottom-edge coordinate.

Let $q = \lfloor N/12 \rfloor$. The four constraint levels are

level	proximity	ordering	edge-alignment	total
$C0$	0	0	0	0
$C1$	q	q	q	$3q$
$C2$	$2q$	$2q$	$2q$	$6q$
$C3$	$3q$	$3q$	$3q$	$9q$

For each benchmark instance the same rectangles and hidden optimum are used at all four levels, only number of constraint relations change.

Continuous baseline The continuous baseline minimizes a single differentiable surrogate with Adam (Kingma, 2014). Maximizing U is the same as minimizing the bounding-box area, so the objective is

$$\mathcal{L}(p) = W_\beta H_\beta \; + \; \lambda_{\text{ov}}\, O(p) \; + \; \lambda_c\, V(p). \tag{13}$$

The first term is the bounding-box area, which we smooth by using log-sum-exp at temperature β,

$$W_\beta = \beta \log \sum_i e^{(x_i + w_i)/\beta} + \beta \log \sum_i e^{-x_i/\beta}, \tag{14}$$

this is also done for H_β. The second term is the pairwise overlap area. Two rectangles overlap in a rectangle whose sides are the one-dimensional overlaps of their projections

$$O(p) = \sum_{i<j} \big[\, \min(x_i{+}w_i, x_j{+}w_j) - \max(x_i, x_j) \big]_+ \big[\, \min(y_i{+}h_i, y_j{+}h_j) - \max(y_i, y_j) \big]_+, \tag{15}$$

The third term is the sum of the constraint. Each constraint is turned into a squared hinge that is exactly zero when the relation holds and grows quadratically outside it. For example, $\big[\, |x_j - x_i| - \tau \,\big]_+^2$ for x-alignment and $\big[\, x_i + w_i - x_j \,\big]_+^2$ for ordering.

Initially all objects are placed randomly within the area $G \times G$. Over the T optimization steps the weights $\lambda_{\text{ov}}, \lambda_c$ are annealed up while the learning rate is annealed down. The reported results are from the best of 20,480 restarts per benchmark instance.

LGO We adapt the LGO method to this benchmark. We omit the wire mask and the alignment mask, but we keep the view map and the position map. We also add three new maps that denote the proximity, the edge alignment, and the relative ordering. Specifically, for each constraint type $q \in \text{prox}, \text{edge}, \text{ord}$, we define a map $f_q^{(t)} \in [0,1]^{W \times H}$ with $f_q^{(t)}[x,y] = \mathbf{1}C_q^{(t)}(x,y)$, where $C_q^{(t)}(x,y)$ denotes that placing the current block at ((x,y)) satisfies, respectively, the prescribed proximity, edge-alignment, or relative-ordering constraint. For each benchmark instance, we carry out the population search for 200 rounds optimizing for the objective

$$Q(p) = \big(\text{sat}_{\text{prox}} + \text{sat}_{\text{ord}} + \text{sat}_{\text{align}} + U(p)\big) \tag{16}$$

where sat_f is the fraction of family f's relations that hold.

Training set Training layouts are produced by using the continuous baseline. To set a conservative utilization cap for the training set, we first took the best continuous-baseline result from 20,480 runs per benchmark instance. We then set the global cap to 0.02 below the worst of all the results on the benchmark instances. This yielded a cap of $U = 0.7816$, meaning that there was no sample in the training set with a greater utilization. For each N, we produce a training set of 1,200 samples and 8 unseen benchmark instances

E.1 Absolute utilization on the constrained-packing benchmark

Figure 8 shows the absolute utilization achieved by LGO and the continuous baseline, while Figure 4a shows the difference between these two methods. For both methods, utilization decreases as the number of relational constraints increases. However, the LGO is more robust to this increasing of the relational constraints.

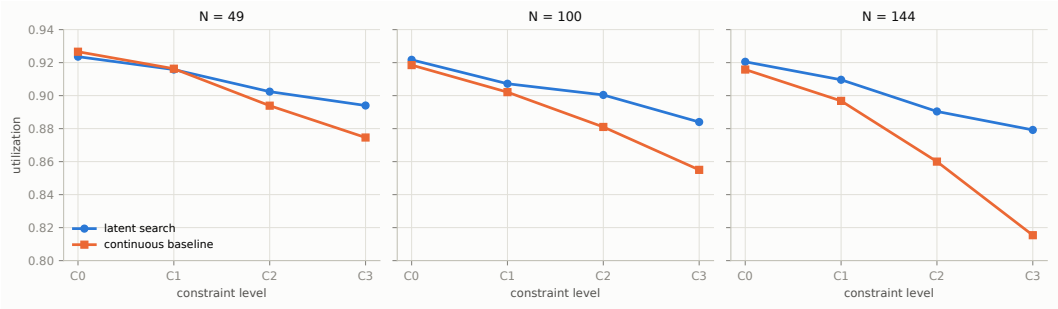


Figure 8: Absolute utilization across constraint levels. Each panel uses a fixed number of objects N.

E.2 Constraint type isolation

In order to validate whether the results shown in Figure 4a are due to one particular type of constraint being difficult for our continuous baseline, we isolate the three different constraints. We create a new dataset using the continuous method. Again we cap the training set utilization, in this case to $0.8214 - 0.02 = 0.8014$. Table 13 shows the mean performance of our latent method versus the continuous baseline on benchmark instances with only one of the three constraint families, each containing 72 constraint relations, and a composite containing all three constraints, where each constraint family contains 24 relations.

Our latent optimization method is able to outperform the continuous baseline for the proximity only and alignment only benchmark instances. However, the latent and continuous methods perform very similarly in terms of the ordering constraint. This might be explained by the fact that the ordering

relation requires only one thing: that a rectangle j is placed to the right of i. Meaning that j can be placed anywhere on the y-axis and arbitrarily far along x. Contrast this with the proximity constraint and alignment constraint which are much more restrictive in the places a rectangle can be placed while still satisfying the constraint.

Table 13: Constraint type isolation at $N = 144$. For each constraint type there are 72 constraint relations: the three constraint isolation family settings concentrate them in one family, the composite splits them $24/24/24$ across all three. The continuous baseline uses $20{,}480$ restarts, whereas the latent optimization method runs for 200 rounds. cont. feasible is the fraction of continuous restarts that land exactly constraint-feasible.

Constraint type	latent	continuous	cont. feasible
proximity only	**0.8970**	0.8349	0.434
ordering only	0.8905	**0.8910**	0.782
alignment only	**0.8999**	0.8214	0.570
composite ($C2$)	0.8906	0.8613	0.371

F 2D STRIP PACKING

Going from 3D floorplanning to two-dimensional irregular strip packing, we remove all inter-object constraint relations except non-overlap. Strip packing keeps the same high-level feasibility conditions of Equation 1, i.e. objects must remain in the container and may not overlap, but changes the geometry and optimization objective. There is a single container, so d_i and the alignment term disappear. Likewise, there is no netlist which means we remove the HPWL term. The outline is no longer fixed: only the height H is given while the W is minimized. Lastly, there are no soft aspect ratios, but each object is allowed to rotate, making the decision variable (x_i, y_i, θ_i). In contrast to the axis-aligned rectangles in 3D floorplanning, the objects are convex polygons.

F.1 BASELINES

XAtlas (Young, 2022) places objects using a brute-force placement strategy. The no-fit polygon (NFP) method (Bennell & Song, 2008) uses Minkowski sums (Cox et al., 2021) to characterize feasible placements between irregular shapes. SVGnest (Qiao, 2017) combines NFP-based placements with a genetic algorithm. GFPack++ (Xue et al., 2024b) is an attention-based diffusion model for 2D irregular packing that learns gradient fields in order to move and rotate polygons jointly. It uses attention-based geometry and relation encoding to model interactions between shapes and boundaries.

Continuous baseline The continuous baseline minimizes a single differentiable surrogate with Adam, jointly optimizing the poses of all N objects and the container W. Each object i has a pose (x_i, y_i, θ_i). Non-overlap is enforced using a weighted signed-distance-field (SDF) collision penalty (Macklin et al., 2020). Containment is enforced by a soft wall penalty, which we adapted from (Martínez et al., 2009), applied to all four walls. H is fixed, as the definition of strip-packing requires. By contrast, because the x-bound of the wall penalty is W itself, the term is differentiable in the container as well as in the poses: every object leaning on the wall pushes it outward. An unconditional pressure term $\lambda_{\text{area}} WH / \sum_i A_i$, where A_i is the area of object i, rewards a smaller W, and the optimization settles where the two gradients balance. This is the differentiable counterpart of the adaptive shrinking cell of (Torquato & Jiao, 2009). Optimization uses a cosine-decayed learning rate (Loshchilov & Hutter, 2017) and all poses are initialized uniformly at random.

F.2 MODIFICATIONS FOR 2D STRIP PACKING

LGO with CNN head In order to adapt the model to the dental benchmark we omit the wire mask and the alignment mask, while keeping the view map and the position map. We do not include the footprint head, but we do keep the CNN cell head, as described in Appendix A. We found that the Dental benchmark required a relatively fine resolution. At this resolution, however, search becomes computationally prohibitive and is unable to find satisfactory solutions within a practical

time budget. For evaluation, we chose a resolution of 640×400 and discretized the rotation in 64 bins.

LGO with linear head Instead of predicting x_i, y_i, θ_i, the linear head only predicts y_i, r_i. The column is not chosen by the model at all, but supplied using the Skyline rule (Wei et al., 2011). A piece is slid to the left until contact. This also means that feasibility is guaranteed. One important limitation of this method in its current form is that the order in which the objects are placed is fixed. We place the polygons in decreasing order of area. One possible solution would be to encode the ordering in z, however, we leave this for further research. For evaluation, we chose a resolution 3200 and discretized the rotation in 64 bins.

F.3 Training datasets

Table 14 shows the min, mean, max and standard deviation of two datasets we used for training. Xue et al. (2024b) created a training set by running the SVGnest algorithm on $196{,}000$ unique combinations of 440 different polygon shapes. Both the GFPack++ model and our LGO CNN head model were trained on this dataset.

An additional dataset was created to accommodate the LGO linear head method, since this model requires that objects in the training set are placed in order of decreasing polygon area. Similarly to the SVGnest dataset, we take unique combinations of the 440 different polygon shapes, while ensuring that these combinations do not appear in the test instances. To generate the dataset we supply the objects in decreasing polygon area to a randomized greedy bottom-left-fill teacher (Burke et al., 2006).

Table 14: Utilization of the training datasets used. In order to create a dataset for training the linear head we use a heuristic approach to generate a dataset. Both training sets have a strip height of $H = 1200$.

Dataset	Samples	Min	Mean	Max	SD
SVGnest	196,113	0.5647	0.7424	0.7919	0.0149
Heuristic	96,000	0.6105	0.7354	0.7892	0.0168