Peer Review File

Manuscript Title: A Graph Placement Methodology for Fast Chip Design

Reviewer Comments & Author Rebuttals

Reviewer Reports on the Initial Version:

Reviewer #1

=Summary of the key results=
Chip design is a labor intensive process that requires highly trained expert personnel to spend thousands of hours for each chip.

This paper proposes a new architecture for automatically generating chip designs using deep reinforcement learning (RL). Compared to prior work on design optimization the main differences are that the approach can deal with non-differentiable cost functions and this new architecture allows the agent to at least partially generalize across different chip architectures, rather than having to be trained from scratch for each new design.

=Originality and significance: if not novel, please include reference=
Clearly this paper addresses a hugely relevant issue and shows impressive performance. In particular, the architecture reduces the training time from 48 hours to less than 10 hours for equivalent performance.

In terms of significance, it would be great if the paper was more specific about the advantages of the method. The most convincing evidence is comparison to actual *production design* of a previous chip (that was presumably state-of-the-art with baseline methods) and show that using this new method (even with more training time), it's possible to obtain a far superior performance.

This would clearly demonstrate superhuman performance, which as far as I can tell, is currently missing from the paper. Relatedly, this statement ""Our method has been used in production to design the next generation of Google TPU." is somewhat unclear. How was the method used? Did it provide the final layout or did it provide a starting point for the further optimization by humans? If it provided the final starting point, how long was it trained for in total with how many resources?

Furthermore, it also is unclear to what extent the amount of compute used between the method and different baselines is comparable. As described in 6.3, the method takes advantage of a large number of simplifications and approximations - are those also available to the method? It would also be great if the paper could be more specific about the compute resources required both for training the network and for finetuning it.

While I understand that calculating optimal policies for these problem settings is NP-hard, it would still be good to provide some intuitions about how far from optimal the solutions found are likely to be. In particular, I would be extremely interested to know whether better solutions can be obtained by training a lot longer, even on one specific chip design. If so, why hasn't this been tried? And if not, what is the reason?

Lastly, it would be great to understand the limitations of generalization for this chip design process and what components of the architecture really matter. Which relevant benchmarks and ablations were compared to within deep RL?

=Data & methodology: validity of approach, quality of data, quality of presentation=
The overall approach and presentation is sound. The only comment I have is about the fundamental framing of the problem. I believe that the underlying problem is more naturally formalized as a contextual bandit problem, with an extremely high dimensional, combinatorial action space (one action dimension for each macro).
While it is entirely reasonable to transform the high dimensional action space into a sequential (MDP) setting to reduce the challenges associated with this large action space, it would be great to mention that this is what is happening.

This approach is not too dissimilar to how the "Greedy MDP" (https://arxiv.org/pdf/1609.02993.pdf) deals with the large action space associated with multi-agent learning (one action dimension for each agent). I'd be extremely surprised if this approach hasn't been taken for bandits with high dimensional action spaces previously, but unfortunately I don't have a good reference at this point.
Either way it would be great to clarify the formalism.

Appropriate use of statistics and treatment of uncertainties
As far as I can tell, none of the results provided include uncertainty or significance measures. There might be good reasons for this, e.g. that individual runs are too expensive to try multiple seeds. Similarly, little insight was provided regarding the hyperparameter selection process.

=Suggested improvements: experiments, data for possible revision=
As mentioned above this problem setting is naturally formalized as a contextual bandit, while the MDP formalism seems more convenient for practical optimization.
Some related work is missing, as described below.
It would be great to obtain superhuman results on an actual previous production chip, rather than one that was designed purely as a human baseline.
Baselines and ablations for the architecture used are missing.
What are the limitations in terms of generalization of the method used? What is the limiting factor in terms of final performance?
Some writing should be updated, see below.
It would be great to address all of the questions in this review.

=References: appropriate credit to previous work?=
It would be great to list other examples of representing contextual bandits with large combinatorial action spaces as sequential problems. I would also expect more literature on RL for combinatorial optimization to be cited.

In particular, the following claim, "While there have been a number of attempts to use machine learning for combinatorial optimization … our approach is the first to leverage past experience to generate higher quality results and reduce training time," is inaccurate:
One example that does exactly this (generalizing to unseen problem settings) is "Exploratory Combinatorial Optimization with Reinforcement Learning" (T. Barrett et al, AAAI, 2020, https://arxiv.org/pdf/1909.04063.pdf). While I am not aware of other references, I'd be surprised if this paper is the only example missed.

With this in mind, I recommend that the authors focus on the practical relevance of the method and provide more insight into why the architecture chosen works and what the limitations of the method are.

=Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions=
The paper is well written. There are a couple of places that are somewhat misleading:
In the introduction the authors mention that ‘partition based’ approaches sacrifice global optimality. However, 6.3 implies that the method also relies on partition based approaches:
“We group millions of standard cells into a few thousand clus ters using hMETIS [26], a partitioning technique based on 453 the normalized minimum cut objective‟

Similarly, I find it odd that as much focus is placed on the architecture that generalizes across different chip layouts by processing the chip context. Reducing a process that takes six months of human work to an automated process that can be trained within 48 hours seems like a much bigger step than further reducing this training time to 6 hours through a complicated architecture that generalizes across chip layouts.
Yet, the paper focuses a lot more on the reduction of compute from 48 to ~6.

Reviewer #2

A. Summary of the key results

The paper describes a VLSI placement algorithm that casts chip floorplanning as a reinforcement learning problem. A VLSI netlist is cast as graph, and placed onto the chip, with the usual wirelength and wiring congestion metrics as reward metrics.
The algorithm was evaluated on an open source RISC-V processor core from academia, the Ariane, as well as TPU blocks, and the paper indicates that it was used in production to design the next-generation TPU chip. It was compared against RePlAce, a leading academic placement algorithm, as well as a manual placements, and shown to have better timing than RePlAce in 2 test cases, and generating feasible placements in all tests cases whereas RePlAce failed in 3 cases. When compared against manual placements, it has different results for different blocks in terms of timing, area and power.

B. Originality and significance: if not novel, please include reference

The algorithm is novel, and the problem addressed is a significant practical one that can have huge impact.

C. Data & methodology: validity of approach, quality of data, quality of presentation

The presentation of the evaluation methodology can be improved with more details and clarity. The experiments are not explained in sufficient detail to be repeatable for comparison. While I understand that TPU designs are proprietary, the methodology for the open source Araine netlist should be more clearly described so subsequent research can clearly compare against these numbers. How is training and test set separated and chosen? What are the small, medium, large datasets specifically? What are the hyper parameters and how are they chosen?

The evaluation can be strengthened with comparisons against the academic benchmark suites for placement, such as those from ISPD, ICCAD and DAC, which is de rigeur standard in the academic community. For instance, the ReplAce paper compared its results against several other algorithms using the above academic datasets. Nevertheless, I appreciate the practicality of the data sets used for evaluation here for their scale. It is tough, however, to glean insights on why the proposed algorithm does better on certain metrics and not others, when the evaluations against RePlAce and manual are on TPU blocks that are not open, with little information.

It will help if some hypothesis can be presented for the results: For instance, why is TNS so much smaller for the proposed work vs. the manual placement for block 4? When wirelength and wiring density are comparable?

The use of blocks from the same TPU design as training dataset may also have helped in the learning process, as the algorithm was trained with blocks placed by the same person, and hence could be tuned to the heuristics used by that person. Given the homogeneous nature of the TPU, where there are many identical tiles, it is unclear if the same effectiveness can be realized when the placer is trained with different designs.

D. Appropriate use of statistics and treatment of uncertainties

Yes, this is reasonable

E. Conclusions: robustness, validity, reliability

This is a solid paper with a novel algorithm that has been applied to practical product. The impact is substantial.

However, a key contribution of the paper is its application to a large netlist that is an actual production chip, the TPU. Yet, due to confidentiality reasons, this cannot be revealed (even a single snapshot of the generated placement had to be fuzzified) and hence, the validity of this is difficult to verify.

F. Suggested improvements: experiments, data for possible revision

See above for suggested improvements to provide more clarity into the evaluations and enable repeatability and fair comparisons of subsequent research. This will heighten the impact of this work, so follow on academic research can quantitatively compare to this work.

G. References: appropriate credit to previous work?

There have been recent research efforts applying machine learning to VLSI placement. The paper cited a DATE 2019 paper [49]. Another closely related work that was missed and should be discussed is "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement", Lin et al, DAC 2019. It is an open source placement tool that maps RePlAce as deep neural networks on GPUs.

H. Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions

The paper is very well written, with very clear description of the research landscape and the research gap that this work addresses.

Reviewer #3

A. Summary of the key results.
B. Originality and significance.
C. Data & methodology: validity of approach, quality of data, quality of presentation.
D. Appropriate use of statistics and treatment of uncertainties.
E. Conclusions: robustness, validity, reliability.

F. Suggested improvements: experiments, data for possible revision.
G. References: appropriate credit to previous work?
H. Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions.

(Points A, B.) The reviewed paper aims to present a deep reinforcement learning approach to macro placement, a key aspect of physical floorplanning at the block level of modern system-on-chip physical implementation. Macro placement is an important, high-stakes problem, and any progress toward automation is welcome. The many authors have clearly devoted enormous effort and computational resources toward achieving the reported outcomes. The authors claim superiority of results produced by their deep RL network in hours, relative to the results of months of effort by human experts.

The authors' descriptions of methods and experimental validations require substantial improvements for clarity, and for reproducibility and verifiability. Particularly since the approach is so obviously effortful, and its stated impacts and value so great, the clarity and reproducibility of the description are all the more important to provide. Several comments (related to Points C, D, E, F and H) follow.

1. Clarity and precision of exposition.

1.a. The terms "in production" (Lines 315, 786) or "manufacturable" (Lines 6, 102) have well-understood meaning in the IC design / EDA domain. It would be very helpful for the authors to clarify their achievement using accepted definitions, e.g., "Our floorplan solutions are in the product tapeout of a recent-generation TPU chip" or similar. If the authors' floorplan solutions from the fine-tuned policy undergo adjustments and transformations before being used in a production chip (i.e., final floorplan ".def"), those must be explained clearly and in a quantified way.

1.b. The term "netlist graph" is used a number of times in the paper, along with graph-related terms such as "adjacency matrix". However, the VLSI "netlist" is actually a hypergraph (cf. use of hMETIS (h = hypergraph), Lines 406, 433). The transformation of hypergraph to graph is significant and is the subject of many works in the literature. Clarification of specifics (weights, thresholding, directionality, star / clique / other "net model", etc.) should be provided to enable reproducibility.

1.c. As noted below, without any "flowchart" or "pseudocode", and missing many details, the experiments are challenging for readers to follow.

2. Experimental methods and reporting.

2.a. A flowchart or procedural "pseudocode" of the experimental setup and process must be provided. This is customary in domains such as VLSI CAD / EDA. This reviewer is unclear, for example, on whether the authors' macro placements are modified in any way subsequent to the one-shot, or after the fine-tuned, macro placement processes. Such an important aspect must be clearer to readers.

2.b. The compute resources at each step (training, fine-tuning, etc.) must be specified, as is also customary in works that present the results of optimization. (Put another way: In optimization, results improve with additional resources. Therefore, computational and schedule resources used to obtain results need to be stated.)

2.c. Usage of the commercial place-and-route tool needs several important clarifications. (Even if "Cadence" or "Synopsys" is not mentioned, it is necessary to qualitatively indicate the use or non-use of standard steps (e.g., post-placement optimization) or standard options (e.g., dont_touch, low-power effort).) Put another way: Readers must be able to understand how credit is due to the deep-RL macro placement versus the commercial tool that was subsequently applied. (i) Importantly, the "Area" column of Table 1 shows that the netlist is being changed, which suggests that optimization steps are being performed in the commercial tool. The authors should clarify why attribution of timing or power outcomes is meaningful in this light. The customary way to do this is to explain how many sizing or buffer/inverter insertions were made by the commercial tool. Pre- and post-tool timing is important as well; see Comment 2.d.iv below. (ii) The provenance of the (mixed-size) input gate-level netlist needs to be clarified. One obvious clarification to make: was any physical synthesis option used? (iii) the choice of max density threshold 0.6 (Line 139) relative to usage in production for next-generation TPU chips is worth clarifying as well. This reviewer is unclear on the final utilizations after the commercial tool finalizes the testcase layouts.

2.d. The "RePlAce" mixed-size placer is presumably retrieved from the RePlAce GitHub repository. (i) To permit basic understanding, the authors should specify version (commit) and options used, whether the same clustered netlist was provided to RePlAce (as it was to SA, Line 827?), whether post-processing of macro placements (if any) was equally applied, and other basic details. Again, the customary flowchart (showing "A vs. B") would help. (ii) Timing slack and congestion are used to "gray out" three out of five RePlAce results in Table 1. The authors need to describe how these metrics were assessed (i.e., at what point in the commercial tool flow, based on what early/trial or final routing). (iii) Whether the timing-driven capability of RePlAce was used needs to be stated clearly. (iv) A sentence or two to confirm the final metrics of Timing, Power, Wirelength after the commercial tool run, from the 15 rows of Table 1, will help the reader understand that the "graying out" filters are indeed meaningful. See Comment 2.e, next.

2.e. It is well-known that tools have inherent noise or even chaotic behavior, where a small change to inputs results in large changes to outputs. Therefore, in the VLSI CAD literature, and in chip design practice, results are typically compiled after "denoising". The authors should clarify whether their experimental methodology used denoising, and if so, at what junctures.

2.f. Section 8.2 provides a comparison with Simulated Annealing (SA). While the authors note a number of hyperparameters explored, this is "qualitative", with no specifics. (i) The computational resources (e.g., number of states explored, Line 834, per testcase) should be mentioned. (ii) Important details of any SA implementation would include the neighborhood structure (move set), how feasibility is maintained (or, allowed to be violated, with what penalty), move acceptance rates seen (a confirmation of SA well-tuning), etc.

3. Ablation studies.

3.a. The authors report a final result of automated manufacturable, production-worthy macro placements in latest technology nodes (Line 221), with up to a few hundreds of macros and millions of standard cells (Lines 421, 432, 817, etc.). Especially given the effort required to achieve this result, this reviewer believes that "ablation studies" (effort or complexity vs. benefit) and/or application of "Occam's Razor" will add great value to the present reporting. It would be helpful if the authors provided insight into the aspects of their architecture and methodology that contribute the most, at what incremental costs, to the final system and methodology that is described in this paper.

4. Minor.

4.a. (Point G.) Errors in the references (citations) should be fixed.

4.b. Terms such as "normalized" minimum cut objective (Line 407) are confusing to this reviewer. The use of "netlist graph" was also noted, above. How Figure 6 is known to be quite close to the "optimal" arrangement is unclear.

4.c. Line 220: The Ariane RISC-V CPU does not have so many macros as in the authors' presentation. This is confusing to this reviewer. See, e.g., Slide 21 of https://riscv.org/wp-content/uploads/2018/05/14.15-14.40-FlorianZaruba_riscv_workshop-1.pdf . Was the Ariane design transformed for convenience in some way? If the 100+ macros are of identical size as suggested by Figure 6, could the authors comment on variation of results across multiple topological orders that could have been used for tie-breaking (Section 6.3.3, Lines 465-)?

4.d. Clarifying information (Comment 1) should be easy to add without changing the overall length of the paper. "Method" does not add reproducibility beyond the earlier exposition, and the presentation is repetitive. Examples: [Equations (1-2) discussion, Equations (6-7) discussion.], [Lines 405-, Lines 430-].

4.e. Line 67: This reviewer believes that 1000^1000 should be replaced by 1000!

Author Rebuttals to Initial Comments:

Referee #1 (Remarks to the Author):

=Summary of the key results=

Chip design is a labor intensive process that requires highly trained expert personnel to spend thousands of hours for each chip.

This paper proposes a new architecture for automatically generating chip designs using deep reinforcement learning (RL). Compared to prior work on design optimization the main differences are that the approach can deal with non-differentiable cost functions and this new architecture allows the agent to at least partially generalize across different chip architectures, rather than having to be trained from scratch for each new design.
> Thank you for your insightful review! We have used your feedback to greatly improve the clarity of our manuscript and to provide valuable new insights into the RL architecture and problem formulation.

=Originality and significance: if not novel, please include reference=

Clearly this paper addresses a hugely relevant issue and shows impressive performance. In particular, the architecture reduces the training time from 48 hours to less than 10 hours for equivalent performance.

> As you suggest later in this review, the most impactful result is that we can train an agent capable of automatically generating production-quality chip floorplans, and can do so far faster than human designers. By developing and pretraining an RL architecture that generalizes across chips, we were able to further reduce training time from 48 hours to less than 6 hours and improve the quality of the result. Furthermore, our zero-shot mode is capable of generating decent placements in subseconds (as shown in Figure 6), which has implications for future design space exploration in earlier stages of chip design.

In terms of significance, it would be great if the paper was more specific about the advantages of the method. The most convincing evidence is comparison to actual *production design* of a previous chip (that was presumably state-of-the-art with baseline methods) and show that using this new method (even with more training time), it's possible to obtain a far superior performance. This would clearly demonstrate superhuman performance, which as far as I can tell, is currently missing from the paper.
> We agree that comparison with the actual production design would represent the most compelling evidence for the effectiveness of our method, and in fact this is exactly what we show in Table 1. We've updated the manuscript to make this more clear (Section 4.4), and have also added these details below.

The "manual" placement we compare against is the actual production design of the previous TPU chip (TPU-v4). This baseline was generated by the TPU physical design team, which is composed of professional hardware engineers who are experts in generating chip layouts and who are responsible for the layout of this particular chip. These human experts also made use of the strongest baseline algorithms to aid them over the course of several months.

To be more specific about the advantages of our method, below we describe key properties that allow it to generate high-quality placements:

(1) Our method is capable of directly optimizing for non-smooth, non-differentiable cost functions, such as routing congestion. In each iteration, we directly measure the congestion (as defined in Section 6.4.2), and use it as part of our reward function. Because the policy can explore the entire search space and jointly (and directly) optimizes for wirelength, routing congestion, and density, the RL policy implicitly learns the tradeoff between these important metrics, and does not need to rely on heuristics. This results in better global optimization and lower congestion, as can be seen in Table 1. To provide additional evidence for this hypothesis, we have added a new ablation study (Table 5), which shows that we can control the tradeoff between wirelength and routing congestion using the congestion weight in our reward function.

(2) Unlike the prior state-of-the-art (RePlAce), our method maintains legalization (e.g. prohibits overlapping macros, avoids blockages such as clockstraps, and prevents other violations that would render a placement invalid) as it places the nodes, rather than as a penalty or a post-processing step. As a result, it does not suffer from performance degradation as a result of post-processing legalization. Our approach, rather than penalizing illegal placements, avoids them altogether. We maintain legalization by employing a feasibility mask at each optimization step as described in Section 6.4.3. In fact, RePlAce legalization failed on several blocks, so we chose to omit those blocks from our study.

(3) Unlike prior approaches, our method has the ability to learn from experience and improve

over time. As shown in Figures 4, 5 and 9, as the policy is exposed to more data, it becomes both better and faster at placing previously unseen blocks.

Relatedly, this statement ""Our method has been used in production to design the next generation of Google TPU." is somewhat unclear. How was the method used? Did it provide the final layout or did it provide a starting point for the further optimization by humans? If it provided the final starting point, how long was it trained for in total with how many resources?
> No manual adjustments were performed on any of the results reported in the manuscript, except of course the manual placements themselves. The experimental results in the paper are on the previous TPU (TPU-v4) (updated Section 4.4), but our method was also used in production to design the next TPU (TPU-v5) (added Section 8.3).

In the production flow, we use exactly the same RL method described in the manuscript, as well as the same EDA workflow to place standard cells. Although the RL placements were already comparable to manual designs, we performed an additional fine-tuning step with simulated annealing to further boost performance (only in the production workflow), which helped with macro orientation (as rotations are not part of the RL action space). Adding this fine-tuning step improved wirelength by an average of 1.07% (stddev=.04%), slightly reduced timing (average 1.18ns reduction in TNS, stddev=2.4ns), and negligibly affected congestion (less than 0.02% reduction in vertical or horizontal congestion in all cases). The SA fine-tuning had the same action space and hyperparameters as described in Section 8.8. The resulting end-to-end runtime was 8 hours on average. Since that production launch in TPU-v5, we have removed SA from our production workflow and replaced it with a greedy postprocessing step that can optimize macro orientation in less than 5 minutes, which significantly reduced our end-to-end runtime without degrading quality.

We fully automate the placement process through PlaceOpt, at which point the design is sent to a third party for post-placement optimization, which includes detailed routing, clock tree synthesis, and post-clock optimization. This is a standard practice for many hardware teams, and physical designers spend months iterating with commercial EDA tools to produce designs that meet the strict requirements for post-PlaceOpt. The same post-placement optimization is performed on both manual and ML-generated designs.

Thanks for your suggestion to clarify how our method was used in production. We have added a new section to the manuscript with these details (Section 8.3).

In terms of resource usage, for pre-training, we used the same number of workers as training blocks in our dataset (e.g. for the largest training set with 20 blocks, we pre-trained with 20 workers) and the pre-training runtime was 48 hours. To generate the fine-tuning results in Table 1, our method ran on 16 workers for up to 6 hours, but the runtime was often significantly lower due to early stopping. For both pre-training and finetuning, a worker consists of an Nvidia Volta GPU and 10 CPUs each with 2GB of RAM. For zero-shot mode (applying a pre-trained policy to a new netlist with no fine-tuning), we can generate a placement in less than a second on a single GPU. We've updated the manuscript to clarify our resource usage (Section 4).

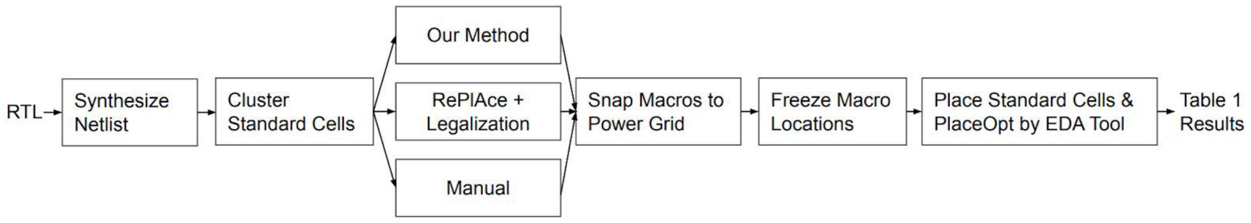
Furthermore, it also is unclear to what extent the amount of compute used between the method and different baselines is comparable. As described in 6.3, the method takes advantage of a large number of simplifications and approximations - are those also available to the method? It would also be great if the paper could be more specific about the compute resources required both for training the network and for finetuning it.

> See above for details on the computational resource usage of our method.

The RePlAce baseline is faster and uses less compute (from a few minutes to an hour on a CPU Intel(R) Xeon(R) W-2135 CPU @ 3.70GHz), but as shown in Table 1, the quality of result is usually much lower and in many cases unusable. Note that after RePlAce converges there is no clear way to leverage additional time and compute to improve the quality of result. [We added details of the computational resource usage of our method and of RePlAce to Section 4].

In Section 8.8, we also compare against simulated annealing (SA), a classic placement algorithm. In order to perform a fair comparison, we ran multiple SA experiments sweeping different hyper-parameters, including max temperature (1e-5, 3e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4, 1e-3), max episode length (5e4, 1e5), and random seed (5 different random seeds), and report the best result in Table 7. The SA baseline uses more compute (80 SA workers * 2 CPUs per SA worker * 18 hours of runtime = 2880 CPU-hours) than our method (16 RL workers * (1 GPU + 10 CPUs per RL worker) * 6 hours = 1920 CPU-hours). Here, we are treating the cost of one GPU as roughly 10 times that of a CPU. If we had stopped SA after 12 hours (and if we didn't use early stopping in RL), then the two methods would have used equivalent compute, but the SA results after 12 hours were not even close to competitive. Even with additional time (18 hours of SA vs. 6 hours of RL), SA struggles to produce high-quality placements compared to our approach, generating placements with 14.4% higher wirelength and 24.1% higher congestion on average.

In all evaluations, we ensured that each method had the same experimental setup, including the same inputs (same approximations and simplifications) and the same evaluation settings (Added details to Section 8.1).



As shown in the figure above, all methods (including SA, though it is not depicted in this figure) have access to the same clustered netlist. After each method completes the placement, the macros are snapped to the nearest power grid and their locations are frozen, and the EDA tool performs standard cell placement. The evaluation settings are drawn directly from our production flow, and we use the same settings to evaluate each method.

Clustering standard cells allowed our method to more effectively optimize the placement of macros. We therefore gave RePlAce access to this approximation (the clustered standard cells) and found that its performance also improved, so we reported results of RePlAce on the netlist with clustered standard cells.

We have added a new Section 8 entitled “Further Evaluation and Experimental Setup” to include all of these details.

While I understand that calculating optimal policies for these problem settings is NP-hard, it would still be good to provide some intuitions about how far from optimal the solutions found are likely to be. In particular, I would be extremely interested to know whether better solutions can be obtained by training a lot longer, even on one specific chip design. If so, why hasn’t this been tried? And if not, what is the reason?

> We have tried running our RL policies for longer, and as shown in Figure 4, the RL policy fully converges after ~40 hours and does not benefit from additional optimization time. In practice (in both the paper and our production flow), we employ a standard early stopping mechanism to stop training once the RL policy converges. However, given that architectural choices and the size of the pretraining dataset greatly affect performance (as discussed in response to your next question), we cannot conclude that the performance of our method (even run over a much longer time horizon) provides an upper bound on optimal performance. You may find it interesting that, as shown in Figures 5 and 9, as we pretrain the current architecture on a larger dataset, the policy takes longer to converge and converges to a better final quality.

Lastly, it would be great to understand the limitations of generalization for this chip design process and what components of the architecture really matter. Which relevant benchmarks and ablations were compared to within deep RL?

> Early in the course of developing this method, we explored the use of various RL optimization algorithms (including REINFORCE, DQN, DDQN, PPO, and others), and found that PPO worked best for our task. In terms of the architecture, we compared against GCN and GraphSage, but found that our novel graph neural network (Edge-GNN) performed and generalized better than these baselines, perhaps because it is more focused on the features of the edges rather than the nodes. This result is intuitive as many important features of a chip layout are more related to edges than nodes (e.g. the total wirelength or the longest path, which relates to timing performance).

In response to your question, we ran an ablation study on the impact of our edge-based graph embeddings (Edge-GNN) vs. node-based graph embeddings (GCN). In Figure 8, we show zero-shot performance as a function of training step (average performance on the five test blocks in Table 1), which is a measure of the ability of the model to generalize. Compared to our Edge-GNN, GCN performs significantly worse. We’ve added this result to the manuscript (Figure 8). Thank you for the helpful suggestion!

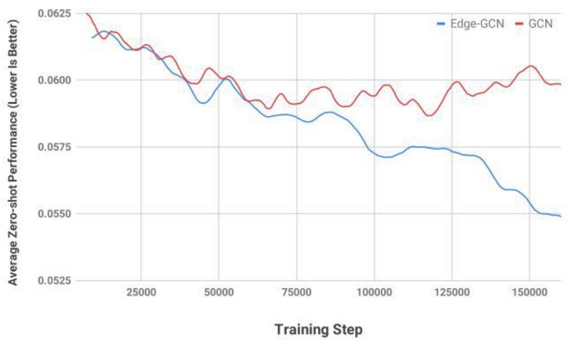
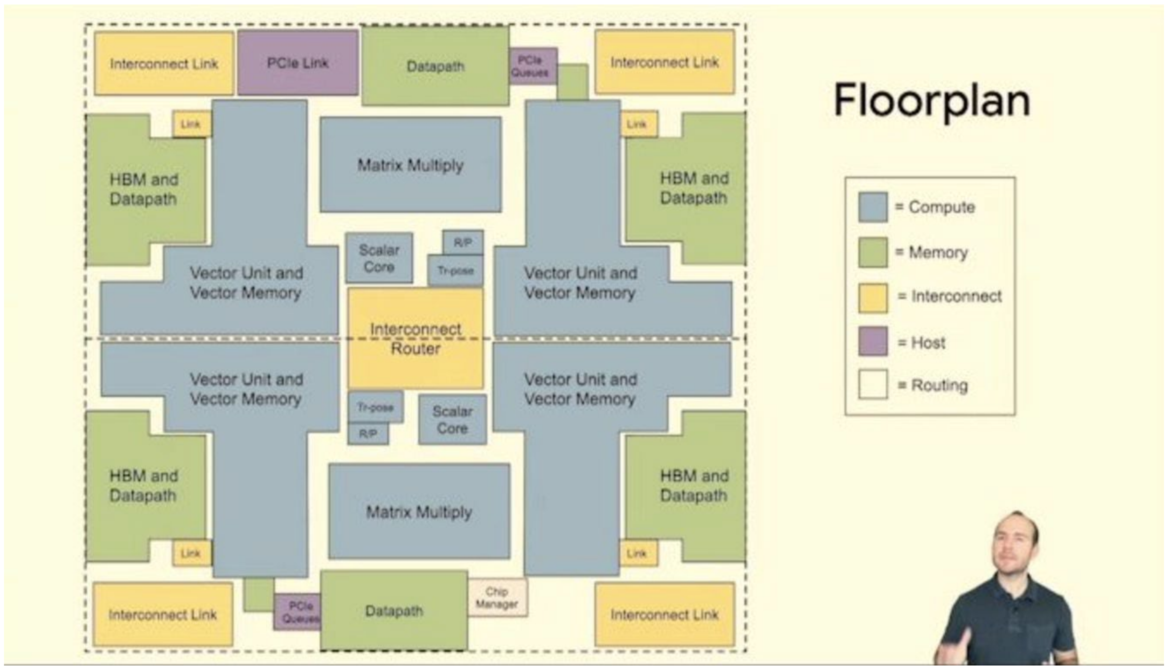


Figure 8: The zero-shot performance of our Edge-GNN vs. GCN [35]. The agent with an Edge-GNN architecture is more robust to over-fitting and yields higher quality results, as measured by average zero-shot performance, on the test blocks in Table 1.

In terms of ability to generalize, our method has produced optimized placements on every chip that we've tried it on so far, including TPU blocks, open source RISC-V blocks, blocks in Pixel phones, and other accelerator blocks. Furthermore, the TPU blocks on which we train and evaluate are quite diverse and include a wide range of saturations (ratio of total area of macros to that of the canvas), macro counts, and functionalities (e.g. on-chip and inter-chip network blocks, computation cores, memory controllers, data transport buffers and logic, and various interface controllers). For more insight into the heterogeneity of TPU blocks, we show the top-level floorplan for TPU-v2 (composed of many blocks for which macros must be placed), which was recently presented at Hot Chips 2020: https://www.anandtech.com/show/16005/hot-chips-2020-live-blog-google-tpuv2-and-tpuv3-230p m-pt



However, in terms of limitations, we do not currently support non-rectangular macros (e.g. other rectilinear polygons), but we are in the process of adding support for these irregularly shaped macros (requires updating our mask). Additionally, due to memory constraints, our current method would not scale to 10,000s of macros (but we are not aware of any chip blocks with more than 1000 macros). Further innovation (e.g. model parallelism) would be needed if these larger-scale chips emerge in the future.

=Data & methodology: validity of approach, quality of data, quality of presentation=
The overall approach and presentation is sound. The only comment I have is about the fundamental framing of the problem. I believe that the underlying problem is more naturally formalized as a contextual bandit problem, with an extremely high dimensional, combinatorial action space (one action dimension for each macro).
While it is entirely reasonable to transform the high dimensional action space into a sequential (MDP) setting to reduce the challenges associated with this large action space, it would be great to mention that this is what is happening.

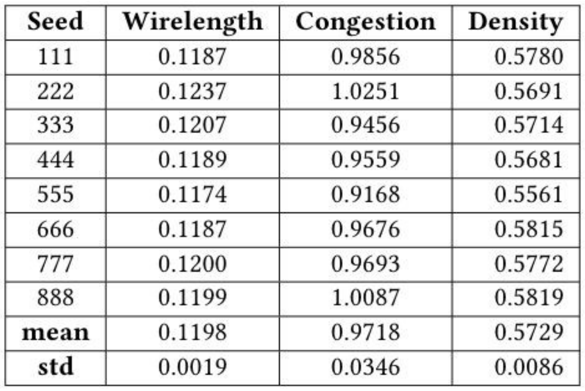
This approach is not too dissimilar to how the "Greedy MDP" (https://arxiv.org/pdf/1609.02993.pdf) deals with the large action space associated with multi-agent learning (one action dimension for each agent). I'd be extremely surprised if this approach hasn't been taken for bandits with high dimensional action spaces previously, but unfortunately I don't have a good reference at this point.
Either way it would be great to clarify the formalism.

> Thank you for this insightful comment! We agree that it is worth stating explicitly that we have reduced a high-dimensional contextual bandit problem to a sequential MDP. We have updated the manuscript to reflect this (Section 2), and added the reference you suggested ("Greedy MDP"), as well as [1], [2], and [3], which are previous examples of this reduction. As you probably guessed, the reason that we chose to perform this reduction was the extremely large action space and the significant impact that prior actions have on the feasibility of those remaining (in terms of routing congestion and cell density). In practice, this means that our method benefits greatly from the use of a mask to limit its search space, which would be difficult to employ if we formulated the problem as contextual bandits.

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. Arxiv, 2016.
[2] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean. ICML 2017.
[3] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean. A Hierarchical Model for Device Placement. ICLR 2018.

Appropriate use of statistics and treatment of uncertainties
As far as I can tell, none of the results provided include uncertainty or significance measures. There might be good reasons for this, e.g. that individual runs are too expensive to try multiple seeds. Similarly, little insight was provided regarding the hyperparameter selection process.
> Yes, you are correct that each full evaluation run is extremely expensive (takes up to 72 hours with an EDA license that costs ~$MM/year), which is why we can't afford to provide error bars in all cases. However, in response to your comment, we did perform sensitivity analysis on the impact of random seed when fine-tuning a policy on a new block. At a high-level, we found that the choice of random seed had negligible impact; over 8 runs (with different random seeds), standard deviation in the overall cost was .0022. We've included the full results below, and have also added them to the manuscript in Table

6.



Seed	Wirelength	Congestion	Density
111	0.1187	0.9856	0.5780
222	0.1237	1.0251	0.5691
333	0.1207	0.9456	0.5714
444	0.1189	0.9559	0.5681
555	0.1174	0.9168	0.5561
666	0.1187	0.9676	0.5815
777	0.1200	0.9693	0.5772
888	0.1199	1.0087	0.5819
mean	0.1198	0.9718	0.5729
std	0.0019	0.0346	0.0086

As described in Section 3, we grounded the selection of policy architecture in the supervised task of predicting proxy wirelength and routing congestion. We've updated the manuscript to clarify this process and our motivation for it (Section 6.7). We've also added a section with details of our hyperparameter choices (Section 8.1). In terms of optimization hyperparameters, we used common defaults (e.g. Adam optimizer) and performed minimal hyperparameter tuning.

We used the same hyperparameters for all experiments reported in the paper, and we do not perform hyperparameter tuning when fine-tuning on test blocks.

=Suggested improvements: experiments, data for possible revision=

As mentioned above this problem setting is naturally formalized as a contextual bandit, while the MDP formalism seems more convenient for practical optimization.

> Yes, agreed and discussed above!

Some related work is missing, as described below.

> We've added the suggested citations and highlighted a few more as described below.

It would be great to obtain superhuman results on an actual previous production chip, rather than one that was designed purely as a human baseline.

> As described above, our "manual" baseline was the actual previous production chip (TPU-v4) designed by the TPU physical design team, and was not a straw(hu)man designed purely to serve as a human baseline.

Baselines and ablations for the architecture used are missing.

What are the limitations in terms of generalization of the method used? What is the limiting factor in terms of final performance?

> As we discussed above, our method does not yet support non-rectangular macros, and although it performed well on all of the blocks we've tested it on so far (including TPU blocks, the open-source Ariane core, and blocks of Pixel phone chips), it may not work well on netlists with 10,000s of macros, which in practice we have not encountered.

To summarize the discussion above on baselines and ablations, we did run early experiments on other RL optimization algorithms (e.g. DQN, DDQN, REINFORCE) and architectures (GraphSage and GCN), but found that PPO combined with our novel edge-based graph neural network (Edge-GNN) performed best. Thanks to your suggestions, we also added new ablation studies, including one which examined the impact of random seed and one which examined the impact of edge-based (vs. node-based) graph embeddings.

Some writing should be updated, see below.
It would be great to address all of the questions in this review.

=References: appropriate credit to previous work?=
It would be great to list other examples of representing contextual bandits with large combinatorial action spaces as sequential problems. I would also expect more literature on RL for combinatorial optimization to be cited.

In particular, the following claim, “While there have been a number of attempts to use machine learning for combinatorial optimization … our approach is the first to leverage past experience to generate higher quality results and reduce training time,“ is inaccurate:

One example that does exactly this (generalizing to unseen problem settings) is “Exploratory Combinatorial Optimization with Reinforcement Learning” (T. Barrett et al, AAAI, 2020 https://arxiv.org/pdf/1909.04063.pdf). While I am not aware of other references, I’d be surprised if this paper is the only example missed.
> We agree that there has been previous work showing the ability of RL policies to generalize, and in fact we cited a number of works on RL for combinatorial optimization: [1], [2], [3], [4], and [5]. We also cited [6] and [7], which show domain adaptation of RL on a combinatorial optimization problem. The paper you mentioned is also highly relevant, so we’ve added a citation to that as well. Thanks for the suggestion!

To clarify, our claim is not that we were the first to develop an RL policy capable of generalization in solving combinatorial optimization problems, but rather that we were able to both reduce training time and converge to a better solution than if we had trained from scratch (see Figure 3). We thought that this (improved quality) was an interesting result, as most prior work focuses on how pretraining reduces training time or allows the method to scale to larger (more real world) examples. Nevertheless, we’ve updated the manuscript to clarify our claims, which are as follows: to our knowledge, we are the first deep RL method to be used in production to solve a combinatorial optimization problem, namely designing the layout of the latest generation of Google TPU (Updated Section 7).

[1] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. arXiv 2016.
[2] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean. ICML 2017.
[3] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, Le Song, Learning Combinatorial Optimization Algorithms over Graphs, NeurIPS 2017
[4] Barret Zoph, Quoc V. Le, Neural Architecture Search with Reinforcement Learning, ICLR 2017.
[5] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean. A

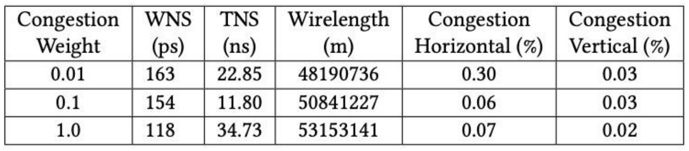
Hierarchical Model for Device Placement. ICLR 2018.
[6] Addanki, R., Venkatakrishnan, S. B., Gupta, S., Mao, H. & Alizadeh, M. Placeto, Learning generalizable device placement algorithms for distributed machine learning, NeurIPS 2019.
[7] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu, Mangpo Phitchaya Phothilimtha, Shen Wang, Anna Goldie, Azalia Mirhoseini, James Laudon. Transferable Graph Optimizers for ML Compilers. NeurIPS 2020.

With this in mind, I recommend that the authors focus on the practical relevance of the method and provide more insight into why the architecture chosen works and what the limitations of the method are.
> Thanks, we've updated the paper to better highlight the practical relevance of our method. As described earlier, we have also clarified our claims and added more insights into the advantages and limitations of our method.

To provide more insight into the why the chosen architecture and optimization algorithms work, we summarize the results of three newly added ablations studies:

- **(For reward function)** We ran a study on the impact of congestion cost on final quality of result. As expected, we show that higher congestion weight improves post-PlaceOpt congestion results, suggesting that our congestion weight allows us to control the trade-off between wirelength and routing congestion (Table 5).

Table 5: Effect of different cost trade-offs on the post-PlaceOpt performance of Block 1 in Table 1. As expected, increasing congestion weight improves both horizontal and vertical congestion up to a point, but results in wirelength degradation, due to the inherent trade-off between these two metrics.



Congestion Weight	WNS (ps)	TNS (ns)	Wirelength (m)	Congestion Horizontal (%)	Congestion Vertical (%)
0.01	163	22.85	48190736	0.30	0.03
0.1	154	11.80	50841227	0.06	0.03
1.0	118	34.73	53153141	0.07	0.02

> **(For architecture)** We ablate our novel edge-based graph neural network (Edge-GNN) and instead use a standard non-edge based GNN baseline (GCN), demonstrating that this architectural innovation improves the zero-shot performance of our method and its ability to generalize (shown in Figure 8).

> **(Impact of random seed)** We performed a sensitivity analysis on the impact of random seed when fine-tuning a policy on a new block (shown in Table 6), and observed negligible sensitivity to random seed (standard deviation of 0.0022 in overall cost).

=Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions=
The paper is well written. There are a couple of places that are somewhat misleading:
In the introduction the authors mention that 'partition based' approaches sacrifice global optimality. However, 6.3 implies that the method also relies on partition based approaches: "We group millions of standard cells into a few thousand clusters using hMETIS [26], a partitioning technique based on 453 the normalized minimum cut objective"
> The problem we are addressing is macro placement, not standard cell placement (which is a well-understood problem and is not performed manually by human experts). However, macro

placement is affected by the placement of standard cells, so we use clustering techniques to perform approximate standard cell placement in order to quickly provide feedback to the RL agent at the end of each episode. Because partitioning-based methods sacrifice quality of global solution, we do not use them for macro placement (the target task), and instead perform global optimization using RL (place all macros onto the full canvas). In contrast, partitioning-based methods break the problem into smaller and smaller subproblems (subnetlists and subcanvases) until it is possible to solve them optimally with analytic solvers. Thanks for raising this question, as it helped us to improve the clarity of the manuscript.

Similarly, I find it odd that as much focus is placed on the architecture that generalizes across different chip layouts by processing the chip context. Reducing a process that takes six months of human work to an automated process that can be trained within 48 hours seems like a much bigger step than further reducing this training time to 6 hours through a complicated architecture that generalizes across chip layouts.
Yet, the paper focuses a lot more on the reduction of compute from 48 to ~6.

> Yes, we agree that the main breakthrough in the paper is training an artificial agent that can generate production-quality layouts in hours rather than months. Although we tried to strike the right balance in the introduction, we do agree with your perspective, so we updated the introduction to make this more clear.

However, we do think that it's also important to discuss generalization, because it has major implications for the future of chip design:

1. As we expose the agent to a larger set of chip designs, it can improve at the placement task, like a human designer gaining knowledge / experience over time,
2. Generalization also enables (previously impossible) design space explorations at earlier stages of chip design (e.g. RTL or architecture design). Reducing the latency of physical design (and making it possible for this to be an inner loop) will be extremely helpful in guiding impactful decisions made at these earlier stages (and in fact, that's our next major focus!).

Referee #2 (Remarks to the Author):

A. Summary of the key results

The paper describes a VLSI placement algorithm that casts chip floorplanning as a reinforcement learning problem. A VLSI netlist is cast as graph, and placed onto the chip, with the usual wirelength and wiring congestion metrics as reward metrics.
The algorithm was evaluated on an open source RISC-V processor core from academia, the Ariane, as well as TPU blocks, and the paper indicates that it was used in production to design the next-generation TPU chip. It was compared against RePlAce, a leading academic placement algorithm, as well as a manual placements, and shown to have better timing than RePlAce in 2 test cases, and generating feasible placements in all tests cases whereas RePlAce failed in 3 cases. When compared against manual placements, it has different results for different blocks in terms of timing, area and power.

> Thank you for your thoughtful review! Your suggestions have helped us to greatly improve the clarity of our manuscript, and to provide valuable new insights into the performance of our method, new ablation studies, and additional details of our method and evaluation for reproducibility.

B. Originality and significance: if not novel, please include reference

The algorithm is novel, and the problem addressed is a significant practical one that can have huge impact.

C. Data & methodology: validity of approach, quality of data, quality of presentation

The presentation of the evaluation methodology can be improved with more details and clarity. The experiments are not explained in sufficient detail to be repeatable for comparison. While I understand that TPU designs are proprietary, the methodology for the open source Araine netlist should be more clearly described so subsequent research can clearly compare against these numbers.

> As you say, we cannot share further details of the TPU designs, as they are not only proprietary, but also subject to export control. To improve reproducibility, we therefore followed your suggestion and provided additional details on how we ran all of the experiments, particularly those involving the Ariane core. For the Ariane benchmark, we used the following open-source design (https://github.com/pulp-platform/ariane), and mapped all logical memories to physical memories of size 256x16, resulting in 133 macros, as described in the newly added Section 8.2.

How is training and test set separated and chosen? What are the small, medium, large datasets specifically?

> Our training set is 20 TPU-v5 blocks, and our test set is composed of 5 blocks from TPU-v4 and Ariane. Due to confidentiality and export control, we cannot reveal the exact identity of the TPU blocks, though we do provide details of the Ariane RISC-V CPU in our response above and in the manuscript (Section 8.2). However, we can say that TPU blocks are quite diverse, and we were careful to include a representative range of functionalities (e.g. on-chip and inter-chip network blocks, computation cores, memory controllers, data transport buffers and logic, and various interface controllers), saturations (ratio of total area of macros to that of the canvas, <30%, 30-60%, >60%), and macro counts (up to a few hundred). The small set contains 2 blocks, the medium set contains 5 blocks, and the large set contains 20 blocks. We updated the manuscript to include all of these details (Section 4.2).

What are the hyper parameters and how are they chosen?

> As described in Section 3, we grounded the selection of policy architecture in the supervised task of predicting proxy wirelength and routing congestion. We've updated the manuscript to clarify this process and our motivation for it (Section 6.7). We've also added a section to include the details of our hyperparameter choices (Section 8.1). In terms of optimization hyperparameters, we used common defaults (e.g. Adam optimizer) and performed minimal hyperparameter tuning. We used the same hyperparameters for all experiments reported in the paper, and we do not perform hyperparameter tuning when fine-tuning on test blocks.

The evaluation can be strengthened with comparisons against the academic benchmark suites for placement, such as those from ISPD, ICCAD and DAC, which is de rigeur standard in the academic community. For instance, the ReplAce paper compared its results against several other algorithms using the above academic datasets. Nevertheless, I appreciate the practicality of the data sets used for evaluation here for their scale.

> As you suggest, current academic benchmarks often contain only a handful of macros, and do not measure performance on the target task: generating macro placements that are production-quality, meaning that they have low wirelength, density, congestion, power, timing, and area. RePlAce reports results on ISPD-2005, ISPD-2006, and MMS, but these benchmarks consider only wirelength. Optimizing only for wirelength is insufficient for generating usable placements, as there is an inherent tradeoff between important metrics such as wirelength and routing congestion (and a design with low wirelength but high routing congestion is effectively unusable). The RePlAce paper also reports both wirelength and routing congestion on DAC-2012 and ICCAD-2012 [8], but the macros are fixed in these benchmarks, so they are not useful for evaluating the performance of our (macro placement) method.

Given that our goal is to develop methods that help chip designers do their job better and faster, we designed our experiments to mimic the true production setting as closely as possible. By evaluating performance on a modern production chip (TPU) with highly accurate measurements of all the metrics used to decide which designs are sent for tapeout (wirelength, routing congestion, timing, power, and area from a commercial EDA tool), we hope to show a meaningful and practical comparison of top-performing placement methods.

[8] N. Viswanathan, C. J. Alpert, C. N. Sze, Z. Li, and Y. Wei, “The DAC 2012 routability-driven placement contest and benchmark suite,” in Proc. DAC, San Francisco, CA, USA, 2012, pp. 774–782.

It is tough, however, to glean insights on why the proposed algorithm does better on certain metrics and not others, when the evaluations against RePlAce and manual are on TPU blocks that are not open, with little information. It will help if some hypothesis can be presented for the results: For instance, why is TNS so much smaller for the proposed work vs. the manual placement for block 4? When wirelength and wiring density are comparable?

> Although there is correlation between wirelength and timing (e.g. TNS) and most of the literature optimizes for wirelength as a proxy for timing, these two measures are not interchangeable. For example, in the case of block 4, a small number of timing paths significantly affect TNS, but the impact they have on total wirelength is relatively low. In this case, the TNS is quite low for all of the methods.

Below we share some observations about our method’s behavior that may provide insight into the metrics in Table 1, which we have also added to the manuscript (Section 8.7):

1) The RL policy tends to place macros on the same datapath close to each other, which results in better timing performance. The graph neural network embeds the features of each node within the netlist hypergraph by iteratively averaging and applying non-linear transformations to the node’s neighboring nodes and edges. This means that the representation for each node is affected by all nodes and edges within a k-hop radius, where k is the number of iterations of this algorithm. Therefore, one hypothesis is that

the representation of nodes in a given datapath are similar to one another, causing our policy network to generate similar predictions about where they should be placed on the canvas. This naturally results in nodes in the same datapath being placed near to one another, improving timing performance.

2) The policy learns to reserve sufficient area for the subsequent placement of standard cells, as this effectively optimizes its reward function. Even at zero-shot (meaning that we run inference on our policy network in less than one second), our method already exhibits this behavior as shown in Figure 6.

As a concrete example, in Block 5, RePlAce achieves the lowest wirelength by placing the macros very tightly together. However, some macros in the same datapath were placed far from one another, and the area was severely fragmented, resulting in longer timing paths. On the other hand, our method, like the manual design, placed macros in the same datapath closely together, while leaving enough space for related standard cells. By spacing macros more widely, our method degraded TNS, but it achieved the lowest WNS.

The use of blocks from the same TPU design as training dataset may also have helped in the learning process, as the algorithm was trained with blocks placed by the same person, and hence could be tuned to the heuristics used by that person. Given the homogeneous nature of the TPU, where there are many identical tiles, it is unclear if the same effectiveness can be realized when the placer is trained with different designs.

> In order to address this potential concern, we trained on 20 distinct blocks of TPU-v5 and tested on 5 distinct blocks of TPU-v4 as well as the open-source Ariane core. Therefore, the training blocks were not placed nor the heuristics tuned by the same person, and the blocks differ significantly in their input graphs and their final placements. As described earlier, the TPU blocks on which we train and evaluate are not homogenous and in fact represent a wide range of functionalities, saturations, and macro counts. We also show that the policy pre-trained on TPU-v5 is able to effectively place a completely different block (open-source Ariane core), even at zero-shot (running inference in subseconds as shown in Fig. 6). Furthermore, our method has generated optimized placements on every chip that we've tried it on so far, including TPU blocks, open source RISC-V blocks, blocks in Pixel phones, and other accelerator blocks, so it seems unlikely that it has overfit to heuristics used by individual TPU designers.

D. Appropriate use of statistics and treatment of uncertainties

Yes, this is reasonable

E. Conclusions: robustness, validity, reliability

This is a solid paper with a novel algorithm that has been applied to practical product. The impact is substantial.

However, a key contribution of the paper is its application to a large netlist that is an actual production chip, the TPU. Yet, due to confidentiality reasons, this cannot be revealed (even a single snapshot of the generated placement had to be fuzzified) and hence, the validity of this is difficult to verify.

> To address your concerns, we earlier described newly added experimental evidence (including new ablation studies) and further insights into the experimental results. Although we cannot share unblurred images of the TPU floorplans, we do provide clear (unblurred) visualizations of our placement on the Ariane core. As you suggested, we updated the manuscript to provide more experimental details to support reproducibility (added Section 8.1), particularly on the open-source Ariane RISC-IV CPU (Added Section 8.2).

F. Suggested improvements: experiments, data for possible revision

See above for suggested improvements to provide more clarity into the evaluations and enable repeatability and fair comparisons of subsequent research. This will heighten the impact of this work, so follow on academic research can quantitatively compare to this work.

> We completely agree and have updated the manuscript to better support reproducibility and maximize the future impact of this work.

In summary, we added the following new sections to the manuscript:
Section 4: Runtime and compute used in pre-training, fine-tuning, and zero-shot
Section 6.3.1: Synthesis of the Input Netlist
Section 6.3.4: Clustering of Standard Cells
Section 6.3.5: Generation of Adjacency Matrix (in which we discuss how we generated the adjacency matrix for the hypergraph corresponding to the clustered netlist.)
Section 8: Further Evaluation and Experimental Setup (added a flowchart (Figure 7) along with details of how the baseline comparisons were conducted.)
Section 8.1: Experimental Setup (described hyperparameters of the RL and FD algorithm as well as the hMetis hyperparameters)
Section 8.2: Open-Source Benchmark: Ariane RISC-V CPU
Section 8.3: Use in a Production Setting
Section 8.8: Comparing with Simulated Annealing (updated this section to include more details of the simulated annealing algorithm, move set, runtime, and compute usage)

G. References: appropriate credit to previous work?

There have been recent research efforts applying machine learning to VLSI placement. The paper cited a DATE 2019 paper [49]. Another closely related work that was missed and should be discussed is “DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement”, Lin et al, DAC 2019. It is an open source placement tool that maps RePlAce as deep neural networks on GPUs.

> We are of course aware of DREAMPlace (DAC Best Paper Award!), but in the process of rearranging and reducing our manuscript for this submission, we inadvertently dropped this reference, which our arXiv preprint did discuss (https://arxiv.org/pdf/2004.10746.pdf). While DREAMPlace is excellent work, the paper describes an approach to standard cell placement, meaning that it solves a different problem. Furthermore, even on this separate task, DREAMPlace only claims to speed up placement “without quality degradation” as compared to RePlAce, so we felt that a comparison to RePlAce would suffice. We have re-introduced this citation and discussion to our paper (Section 7). Thank you for catching our unintentional omission!

H. Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions

The paper is very well written, with very clear description of the research landscape and the research gap that this work addresses.

Referee #3 (Remarks to the Author):

A. Summary of the key results.
B. Originality and significance.
C. Data & methodology: validity of approach, quality of data, quality of presentation.
D. Appropriate use of statistics and treatment of uncertainties.
E. Conclusions: robustness, validity, reliability.
F. Suggested improvements: experiments, data for possible revision.
G. References: appropriate credit to previous work?
H. Clarity and context: lucidity of abstract/summary, appropriateness of abstract, introduction and conclusions.

(Points A, B.) The reviewed paper aims to present a deep reinforcement learning approach to macro placement, a key aspect of physical floorplanning at the block level of modern system-on-chip physical implementation. Macro placement is an important, high-stakes problem, and any progress toward automation is welcome. The many authors have clearly devoted enormous effort and computational resources toward achieving the reported outcomes. The authors claim superiority of results produced by their deep RL network in hours, relative to the results of months of effort by human experts.

The authors' descriptions of methods and experimental validations require substantial improvements for clarity, and for reproducibility and verifiability. Particularly since the approach is so obviously effortful, and its stated impacts and value so great, the clarity and reproducibility of the description are all the more important to provide. Several comments (related to Points C, D, E, F and H) follow.
> We very much appreciate your detailed and thoughtful review, to which we respond in detail below. Additionally, we've used your feedback to substantially improve the clarity of the manuscript and to better support reproducibility.

1. Clarity and precision of exposition.

1.a. The terms "in production" (Lines 315, 786) or "manufacturable" (Lines 6, 102) have well-understood meaning in the IC design / EDA domain. It would be very helpful for the authors to clarify their achievement using accepted definitions, e.g., "Our floorplan solutions are in the product tapeout of a recent-generation TPU chip" or similar. If the authors' floorplan

solutions from the fine-tuned policy undergo adjustments and transformations before being used in a production chip (i.e., final floorplan ".def"), those must be explained clearly and in a quantified way.
> Thanks for your suggestion. "Our floorplan solutions are in the product tapeout of a recent-generation TPU chip" is accurate, so we've added a similar statement to the manuscript in Section 1.

Note that the placement results on TPU-v4 in Table 1 are reported post-PlaceOpt, but our production placements for TPU-5 were validated externally all the way to tapeout.

In the production flow, we use exactly the same RL method described in the manuscript, as well as the same EDA workflow to place standard cells. Although the RL placements were already comparable to manual designs, we performed an additional fine-tuning step with simulated annealing to further boost performance (only in the production workflow), which helped with macro orientation (as rotations are not part of the RL action space). Adding this fine-tuning step improved wirelength by an average of 1.07% (stddev=.04%), slightly reduced timing (average 1.18ns reduction in TNS, stddev=2.4ns), and negligibly affected congestion (less than 0.02% reduction in vertical or horizontal congestion in all cases). The SA fine-tuning had the same action space and hyperparameters as described in Section 8.8. The resulting end-to-end runtime was 8 hours on average. Since that production launch in TPU-v5, we have removed SA from our production workflow and replaced it with a greedy postprocessing step that can optimize macro orientation in less than 5 minutes, which significantly reduced our end-to-end runtime without degrading quality.

We fully automate the placement process through PlaceOpt, at which point the design is sent to a third party for post-placement optimization, which includes detailed routing, clock tree synthesis, and post-clock optimization. This is a standard practice for many hardware teams, and physical designers spend months iterating with commercial EDA tools to produce designs that meet the strict requirements for post-PlaceOpt. The same post-placement optimization is then performed on both manual and ML-generated designs.

We added these details to the paper to clarify our production workflow (Section 8.3).

1.b. The term "netlist graph" is used a number of times in the paper, along with graph-related terms such as "adjacency matrix". However, the VLSI "netlist" is actually a hypergraph (cf. use of hMETIS (h = hypergraph), Lines 406, 433). The transformation of hypergraph to graph is significant and is the subject of many works in the literature. Clarification of specifics (weights, thresholding, directionality, star / clique / other "net model", etc.) should be provided to enable reproducibility.
> As you point out, the netlist is actually a hypergraph, where the standard cells are the nodes and the nets connecting those cells are the hyperedges [9], and we have updated the text of the manuscript appropriately.

To convert the netlist hypergraph into an adjacency matrix that can be consumed by our graph neural network encoder, we generate edges based on register distances between each pair of macros or standard cell clusters. For full details, we've added a new section (Section 6.3.5: Generation of Adjacency Matrix).

Before generating the adjacency matrix, we use hMETIS to cluster the millions of standard cells in the netlist hypergraph into a few thousand clusters. Placing these clusters with force-directed methods is much faster than placing individual standard cells, and allows us to estimate the quality of the mixed-size placement quickly enough to use this as feedback in the loop of RL.

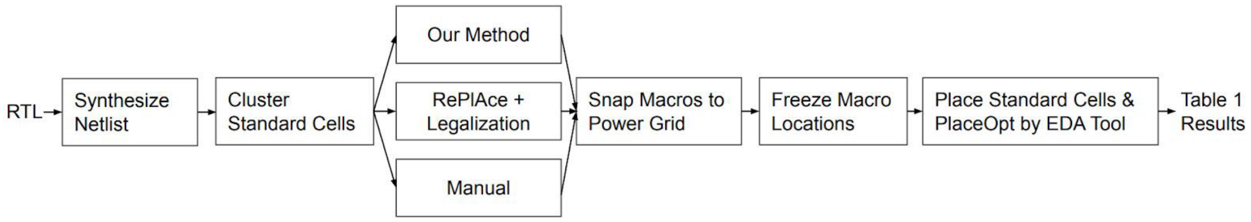
More specifically, we used an open-source hMETIS implementation and below we list our choice of hyperparameter settings:

UBfactor=5: the extent to which unbalanced partitions are permitted.
Nruns=10: the number of bisections performed by hMETIS, the best of which is returned as the final solution.
CType=5 (coarsening type): Edge Coarsening (EC) algorithm (heavy-edge maximal matching). In this mode, pairs of vertices are grouped together if they are connected by multiple hyperedges.
RType=3 (refinement type): Early-Exit Fiduccia-Mattheyses refinement scheme (FM-EE) algorithm. In this mode, the FM iteration is aborted if the quality of the solution does not improve after a relatively small number of vertex moves.
Vcycle=3: Performs Vcycle refinement on each intermediate solution, meaning that each one of the Nruns bisections is also refined using Vcycle refinements.

For all other hyperparameters, we simply use the default hMETIS settings. We've updated the manuscript to include these details for reproducibility (Section 6.3.5, Section 8.1).

[9] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69-79, March 1999, doi: 10.1109/92.748202.

1.c. As noted below, without any "flowchart" or "pseudocode", and missing many details, the experiments are challenging for readers to follow.
> Adding a flowchart is a great suggestion! To clarify our experimental setup, we have added a flowchart (Figure 7 and shown below), as well as a new section containing full details of our experimental setup (Section 8.1).



In summary, we obtain the experimental results in Table 1 as follows: After running each method (including our own, RePlAce, SA, and human experts), we snap macros to the nearest power grid, freeze the macro placements, run a commercial EDA tool to optimize the standard cell placements for the given macro placements and then report the QoR metrics after PlaceOpt. We provide the same input to each method (the clustered netlist hypergraph), and evaluate with the same EDA settings. Note that for RePlAce, before freezing the macros, we also apply its post-processing legalization step, in order to generate legalized placements.

2. Experimental methods and reporting.

2.a. A flowchart or procedural "pseudocode" of the experimental setup and process must be provided. This is customary in domains such as VLSI CAD / EDA. This reviewer is unclear, for example, on whether the authors' macro placements are modified in any way subsequent to the one-shot, or after the fine-tuned, macro placement processes. Such an important aspect must be clearer to readers.

> As mentioned above, we have added a flowchart to clarify our experimental setup. For the fine-tuning and zero-shot results reported in the manuscript, we did not perform any modifications to the macro placement, aside from snapping macros to the nearest power grid to generate the results in Table 1 (described in Section 6.3.7). In order to evaluate with a commercial EDA tool, it is necessary to perform this step before PnR. It was applied equally to all placements in Table 1, and in practice has negligible impact on macro locations. After snapping, we freeze the macro placements and do not allow any further adjustment to their locations.

2.b. The compute resources at each step (training, fine-tuning, etc.) must be specified, as is also customary in works that present the results of optimization. (Put another way: In optimization, results improve with additional resources. Therefore, computational and schedule resources used to obtain results need to be stated.)

> We of course agree that additional computational resources can improve optimization results, and in fact, a recent blog post shows the benefit of scaling up our method on TPU (optimized placements in <1 hour). We've updated the paper to include more details on the computational and schedule resources, which we also describe below.

In terms of resource usage, for pre-training, we used the same number of workers as blocks in our training dataset (e.g. for the largest training set with 20 blocks, we pre-trained with 20 workers) and the pre-training runtime was 48 hours. To generate the fine-tuning results in Table 1, our method ran on 16 workers for up to 6 hours, but the runtime was often significantly lower due to early stopping. For both pre-training and finetuning, a worker consists of an Nvidia Volta GPU and 10 CPUs each with 2GB of RAM. For zero-shot mode (applying a pre-trained policy to a new netlist with no fine-tuning), we can generate a placement in less than a second on a single GPU. We've updated the manuscript to clarify the resource usage (Section 4).

2.c. Usage of the commercial place-and-route tool needs several important clarifications. (Even if "Cadence" or "Synopsys" is not mentioned, it is necessary to qualitatively indicate the use or non-use of standard steps (e.g., post-placement optimization) or standard options (e.g., dont_touch, low-power effort).) Put another way: Readers must be able to understand how credit is due to the deep-RL macro placement versus the commercial tool that was subsequently applied. (i) Importantly, the “Area” column of Table 1 shows that the netlist is being changed, which suggests that optimization steps are being performed in the commercial tool. The authors should clarify why attribution of timing or power outcomes is meaningful in this light. The customary way to do this is to explain how many sizing or buffer/inverter insertions were made by the commercial tool. Pre- and post-tool timing is important as well; see Comment 2.d.iv below.

> Given that the same downstream optimization was applied to each method, for the purposes of comparison, we believe it is appropriate to attribute differences in the final result to differences in the upstream methods. However, we agree that it is interesting to consider how best to attribute the final outcome among the optimization stages of the end-to-end workflow.

In terms of comparing pre- and post-tool timing, the EDA tool does not allow us to run evaluation without running optimization, so it is impossible for us to provide this directly. In fact, early in the project, we met with the EDA company’s engineers multiple times and repeatedly requested this feature, and they were unable to provide it.

The results reported in Table 1 are after PlaceOpt. As mentioned earlier, we freeze the macros for all methods and allow the EDA tool to place standard cells and run PlaceOpt given fixed macro locations. Because we wanted our evaluation to simulate real world conditions, the EDA settings are drawn directly from our production flow and thus we cannot share the details, but we use exactly the same settings (e.g., power effort level) to evaluate each method.

Although we do not optimize directly for timing or power, one explanation for the performance of our method is that, by providing layouts with lower wirelength, routing congestion, and density, we make it easier for the downstream tool to optimize for overall quality of result.

Another explanation for our method’s timing performance is that it tends to place macros on the same datapath close to each other. Our graph neural network embeds the features of each node within the netlist hypergraph by iteratively averaging and applying non-linear transformations to the node’s neighboring nodes and edges. Therefore, one hypothesis is that the representation of nodes in a given datapath are similar to one another, causing our policy network to generate similar predictions about where they should be placed on the canvas. This naturally results in nodes in the same datapath being placed near to one another, improving timing performance. We’ve added this new discussion to the manuscript in Section 8.7.

One way to probe the appropriateness of attributing final QoR improvements to the RL algorithm is to vary properties of the algorithm and observe the impact on these final metrics. To that end, we performed a new ablation study (Table 5), which showed that increasing the congestion weight in our reward function affects congestion of the final placements (after EDA tool PlaceOpt). See below for further discussion of this table.

Congestion Weight	WNS (ps)	TNS (ns)	Wirelength (m)	Congestion Horizontal (%)	Congestion Vertical (%)
0.01	163	22.85	48190736	0.30	0.03
0.1	154	11.80	50841227	0.06	0.03
1.0	118	34.73	53153141	0.07	0.02

As shown in the Table above (which we also added to the manuscript as Table 5), as congestion weight increases, horizontal and vertical congestion decrease, but only up to a point. However, a higher congestion weight comes at the expense of higher wirelength, since there is an inherent tradeoff between these two metrics. A congestion weight of 0.1 represents a sweet spot in this case, as routing congestion is already quite good, but wirelength has not yet overly degraded, which together contribute to lower TNS. We've updated the manuscript with these new results (Section 8.4).

Ultimately, given that all placements (ours, manual, and RePlAce) benefitted from the same downstream optimization method (EDA tool), we believe that improvements in QoR metrics are meaningful, and in fact they were directly used to generate better production TPU layouts.

(ii) The provenance of the (mixed-size) input gate-level netlist needs to be clarified. One obvious clarification to make: was any physical synthesis option used?

> We use Synopsys DC TOPO (Design Compiler Topological Synthesis) to synthesize the netlist from RTL. Synthesis is physical-aware in the sense that it has access to the floorplan size and the locations of the Input-Output (I/O) pins, which were informed by inter- and intra-block level information. The same netlist is given as input to each method with which we compare. We updated the manuscript to include these details in a newly added Section 6.3.1.

(iii) the choice of max density threshold 0.6 (Line 139) relative to usage in production for next-generation TPU chips is worth clarifying as well. This reviewer is unclear on the final utilizations after the commercial tool finalizes the testcase layouts.

> To avoid over-utilization (which would render the layout physically impossible), the TPU team chose 0.6 density as a practical density threshold. We initially ran RePlAce with its default density threshold of 1.0, but found that it performed better with a threshold of 0.6, so we reported RePlAce results with this more favorable setting.

The physical design team gave the following intuition behind their choice of density threshold. As we move to more advanced technology node sizes, the metal pitch does not shrink at the same rate as transistor density, due to challenges with transistor connectivity. Therefore, a relatively low density threshold was chosen to prevent over-utilization.

2.d. The "RePlAce" mixed-size placer is presumably retrieved from the RePlAce GitHub repository. (i) To permit basic understanding, the authors should specify version (commit) and options used, whether the same clustered netlist was provided to RePlAce (as it was to SA, Line 827?), whether post-processing of

macro placements (if any) was equally applied, and other basic details. Again, the customary flowchart (showing "A vs. B") would help.
2.e.

> Yes, we used the version of RePlAce provided in https://github.com/The-OpenROAD-Project/RePlAce/commit/05ef3eeaea0510ce740b9aed26479641679d40d6 based on the commit on Jan 9th, 2020. Aside from density threshold (where it benefitted RePlAce to deviate from its default setting), we used the default settings for RePlAce. We added this information to Section 4.4 of the paper.

As described earlier (and as shown in our newly added flowchart), we ran RePlAce (and SA) on the same clustered netlist as our method. For each method, we snap macros to the nearest power grid in order to be able to perform evaluation with the commercial EDA tool. For RePlAce, we also apply its legalization post-processing step, but otherwise use the same post-processing steps for all methods (snapping only).

(ii) Timing slack and congestion are used to "gray out" three out of five RePlAce results in Table 1. The authors need to describe how these metrics were assessed (i.e., at what point in the commercial tool flow, based on what early/trial or final routing).
> As described earlier, the QoR metrics in Table 1 are obtained after the PlaceOpt step, and we've updated the manuscript to clarify this. According to the TPU physical design team, any block with congestion greater than 1% or WNS significantly above 15% of the total clock cycle (~150ps) is nearly impossible to close in the final metrics. To make our table more readable, we therefore used this criteria to "gray out" placements which the physical design team considered unusable.

(iii) Whether the timing-driven capability of RePlAce was used needs to be stated clearly.
> We did not use the timing-driven capability of RePlAce, and we added a statement to this effect in Section 4.4.

(iv) A sentence or two to confirm the final metrics of Timing, Power, Wirelength after the commercial tool run, from the 15 rows of Table 1, will help the reader understand that the "graying out" filters are indeed meaningful. See Comment 2.e, next.
2.f. It is well-known that tools have inherent noise or even chaotic behavior, where a small change to inputs results in large changes to outputs. Therefore, in the VLSI CAD literature, and in chip design practice, results are typically compiled after "denoising". The authors should clarify whether their experimental methodology used denoising, and if so, at what junctures.
> Because evaluation with commercial EDA tools is extremely expensive (takes up to 72 hours with an EDA license that costs ~$MM/year), we generate and evaluate just one placement for each block. We did not perform any hyper-parameter optimization to generate the results in Table 1, nor did we perform any denoising to generate the final metrics.

As we discuss in response to your earlier question (2.c), we believe that these results are meaningful because each method is evaluated in the same manner, and these metrics are what physical designers use in practice to decide which designs to tapeout. As stated above, to improve the clarity of Table 1, we employed criteria provided by the TPU physical design team to “gray out” placements that they consider unusable.

To address questions about the robustness of our method to noise, we also performed sensitivity analysis on the impact of random seed, which we describe below and in the newly added Table 6 in the manuscript.

2.g.Section 8.2 provides a comparison with Simulated Annealing (SA). While the authors note a number of hyperparameters explored, this is “qualitative”, with no specifics. (i) The computational resources (e.g., number of states explored, Line 834, per testcase) should be mentioned. (ii) Important details of any SA implementation would include the neighborhood structure (move set), how feasibility is maintained (or, allowed to be violated, with what penalty), move acceptance rates seen (a confirmation of SA well-tuning), etc.

> The SA results presented in Table 3 were generated with the following hyperparameter settings, resulting in a total of 80 SA workers. Each SA worker uses 2 CPUs and 2GB RAM.

Max temperature: `t_max`	1e-5, 3e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4, 1e-3
SA steps: `steps`	5e4, 1e5
seed	5 different random seeds

Our SA algorithm works as follows: in each SA iteration (step), we perform 2*N macro actions (where N is the number of macros). A macro action takes one of three forms: swap, shift, and rotate. Swap selects two macros at random and swaps their locations, if feasible. Shift selects a macro at random and shifts that macro to a neighboring location (left, right, up, or down). Rotate selects a new orientation for the macro at random ('north', 'flipped-north', 'south', 'flipped-south', 'east', 'flipped-east', 'west', 'flipped-west'), regenerating the random orientation if it happens to be identical to the original one. After N macro actions are taken, we use a Force-Directed (FD) method to place clusters of standard cells, while keeping macro locations fixed, just as we do in our RL method. For each macro action or FD action, the new state is accepted if it leads to a lower cost. Otherwise, the new state is accepted with a probability of `exp((prev_cost - new_cost) / t)`, where `t=t_max * exp(-log(t_max / t_min) * step / steps)` and `t_min=1e-8.`

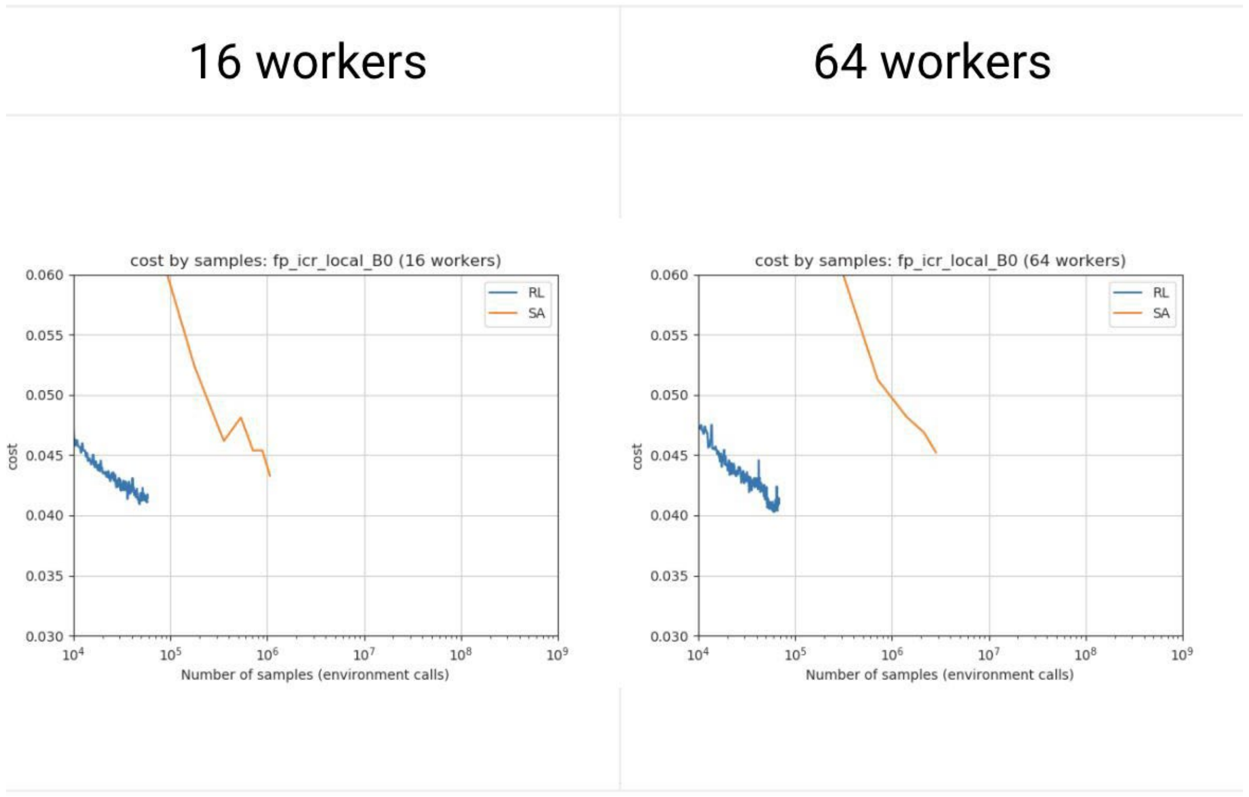
To make comparisons fair, we ran multiple SA experiments sweeping different hyper-parameters, including max temperature (1e-5, 3e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4, 1e-3), max episode length (5e4, 1e5), and seed (5 different random seeds), and report the best result in Table 7. The SA baseline uses more compute (80 SA workers * 2 CPUs per SA worker * 18 hours of runtime = 2880 CPU-hours) than our method (16 RL workers * (1 GPU + 10 CPUs per RL worker) * 6 hours = 1920 CPU-hours). Here, we are treating the cost of one GPU as roughly 10 times that of a CPU. If we had stopped SA after 12 hours (and if we didn't use early stopping in RL), then the two methods would have used equivalent compute, but the SA results after 12 hours were not even close to competitive. Even with additional time (18 hours of SA vs. 6 hours of RL), SA struggles to produce high-quality placements compared to our approach, generating placements with 14.4% higher wirelength and 24.1% higher congestion on average. To maintain feasibility in SA, we employ the same mask to avoid invalid placements as in our RL algorithm, as described in Section 6.4.3.

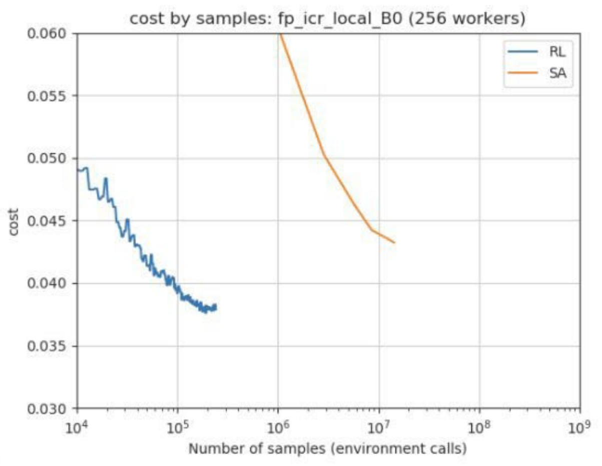
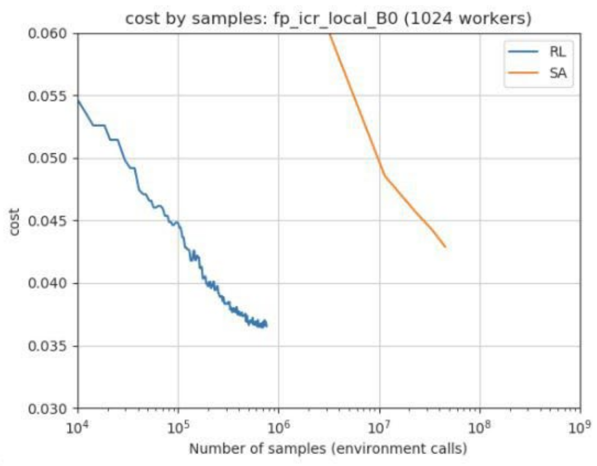
The number of states SA explored (per worker) was num_steps * (2 * num_macros + 1). For example for Block 1, the number of explored states is 1000 * (2 * 101 + 1) = 203K.

We've updated the manuscript with these details (Section 8.8), but the SA implementation / formulation briefly mentioned in our appendix was the result of significant engineering and optimization effort, and we are in the process of preparing a separate publication on both its surprising effectiveness (i.e., given massive compute, SA can scale to large netlists with advanced technology node sizes), but also where it falls short of RL (e.g., ability to leverage compute, sample efficiency, ability to optimize non-smooth reward functions such as congestion, and overall quality of result).

We are saving the unpublished results below for our new paper, but we wanted to give you a preview (proxy cost as a function of the number of states visited), as we thought you might find these results interesting!

Sample Efficiency of RL vs. SA on a TPU block:



256 workers	1024 workers
	

3. Ablation studies.

3.a. The authors report a final result of automated manufacturable, production-worthy macro placements in latest technology nodes (Line 221), with up to a few hundreds of macros and millions of standard cells (Lines 421, 432, 817, etc.). Especially given the effort required to achieve this result, this reviewer believes that “ablation studies” (effort or complexity vs. benefit) and/or application of “Occam’s Razor” will add great value to the present reporting. It would be helpful if the authors provided insight into the aspects of their architecture and methodology that contribute the most, at what incremental costs, to the final system and methodology that is described in this paper.

> As you suggested, to provide greater insight into the aspects of our architecture and methodology that matter most, we performed three new ablation studies:

> **(For architecture)** We ablate our novel edge-based graph neural network (Edge-GNN) and instead use a standard non-edge based GNN baseline (GCN), showing that this architectural innovation impacts the ability of our method to learn from experience and generalize. The figure below depicts the average zero-shot performance of our method on our test set (5 blocks in Table 1) as the policy is pre-trained on our training data (20 blocks). Our Edge-GNN performs better at this generalization task as compared to the baseline GCN method. We’ve added this new ablation study to the manuscript as Figure 8.

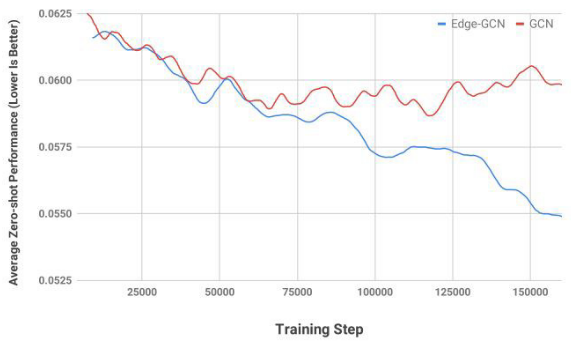


Figure 8: The zero-shot performance of our Edge-GNN vs. GCN [35]. The agent with an Edge-GNN architecture is more robust to over-fitting and yields higher quality results, as measured by average zero-shot performance, on the test blocks in Table 1.

> **(For reward function)** As described earlier, we ran a study on the impact of congestion cost on final quality of result. We show that higher congestion weight improves post-PlaceOpt congestion results, suggesting that our congestion weight allows us to control the tradeoff between wirelength and routing congestion. We've added details of this new study to the manuscript (Section 8.4 and Table 5).

Table 5: Effect of different cost trade-offs on the post-PlaceOpt performance of Block 1 in Table 1. As expected, increasing congestion weight improves both horizontal and vertical congestion up to a point, but results in wirelength degradation, due to the inherent trade-off between these two metrics.

Congestion Weight	WNS (ps)	TNS (ns)	Wirelength (m)	Congestion Horizontal (%)	Congestion Vertical (%)
0.01	163	22.85	48190736	0.30	0.03
0.1	154	11.80	50841227	0.06	0.03
1.0	118	34.73	53153141	0.07	0.02

> **(Impact of random seed)** To address questions about the robustness of our method to noise, we performed sensitivity analysis on the impact of random seed when fine-tuning a policy on a new block. At a high-level, we found that the choice of random seed had negligible impact, with a standard deviation of .0022 in the overall cost across 8 runs. We've added the full results to the manuscript in Table 6.

Table 6: Sensitivity of results to the choice of random seed as measured on a Ariane RISC-V block. We observed little sensitivity to random seed in any of the three cost functions, though variations in wirelength and density are lower than congestion. The overall cost for these eight runs had a standard deviation of 0.0022.

Seed	**Wirelength**	**Congestion**	**Density**
111	0.1187	0.9856	0.5780
222	0.1237	1.0251	0.5691
333	0.1207	0.9456	0.5714
444	0.1189	0.9559	0.5681
555	0.1174	0.9168	0.5561
666	0.1187	0.9676	0.5815
777	0.1200	0.9693	0.5772
888	0.1199	1.0087	0.5819
mean	0.1198	0.9718	0.5729
std	0.0019	0.0346	0.0086

4. Minor.

4.a. (Point G.) Errors in the references (citations) should be fixed.
> Yes, thank you for pointing that out! We've fixed these errors in the references.

4.b. Terms such as "normalized" minimum cut objective (Line 407) are confusing to this reviewer. The use of "netlist graph" was also noted, above.
> Thanks, we've updated the paper to remove confusing and non-standard usage of these terms.

How Figure 6 is known to be quite close to the "optimal" arrangement is unclear.
> This was an observation by the physical design team, who thought it was "clever" of our method to (at zero shot!) reserve a convex hull in the center of the canvas in which to place the standard cells. However, given that we do not know the "optimal" placement, we agree that it is questionable to speculate about how close a given placement is to optimal, and have updated the text of the manuscript and the caption accordingly.

4.c. Line 220: The Ariane RISC-V CPU does not have so many macros as in the authors' presentation. This is confusing to this reviewer. See, e.g., Slide 21 of https://riscv.org/wp-content/uploads/2018/05/14.15-14.40-FlorianZaruba_riscv_workshop-1.pdf . Was the Ariane design transformed for convenience in some way? If the 100+ macros are of identical size as suggested by Figure 6, could the authors comment on variation of results across multiple topological orders that could have been used for tie-breaking (Section 6.3.3, Lines 465-)?
> We used the following Ariane design (https://github.com/pulp-platform/ariane), and mapped all logical memories to physical memories of size 256x16, resulting in a higher number of macros (133 macros). We added a new section to the manuscript which includes these details (Section 8.2). In terms of the ordering, we randomly select a valid topological sort, but as noted in the paper, this is an area in which further performance gains could most likely be achieved (Section
6.3.3 Selection of Macro Order).

4.d. Clarifying information (Comment 1) should be easy to add without changing the overall length of the paper. "Method" does not add reproducibility beyond the earlier exposition, and the presentation is repetitive. Examples: [Equations (1-2) discussion, Equations (6-7) discussion.], [Lines 405-, Lines 430-].
> Our understanding is that materials in the "Method" section do not count toward the word / page limit, so unfortunately deduplicating here will not free up space in the main manuscript. To comply with space limitations, we therefore added clarifying information and additional details to better support reproducibility to the "Method" section. Per your suggestion, in addition to adding and reordering experimental details, we removed duplication that did not sufficiently contribute to clarity.

In summary, we added the following information to the paper to further enable reproducibility:
Section 4: Runtime and compute used in pre-training, fine-tuning, and zero-shot
Section 6.3.1: Synthesis of the Input Netlist
Section 6.3.4: Clustering of Standard Cells

Section 6.3.5: Generation of Adjacency Matrix (in which we discuss how we generated the

adjacency matrix for the hypergraph corresponding to the clustered netlist.)
Section 8: Further Evaluation and Experimental Setup (added a flowchart (Figure 7) along with details of how the baseline comparisons were conducted.)
Section 8.1: Experimental Setup (described hyperparameters of the RL and FD algorithm as well as the hMetis hyperparameters)
Section 8.2: Open-Source Benchmark: Ariane RISC-V CPU
Section 8.3: Use in a Production Setting
Section 8.8: Comparing with Simulated Annealing (updated this section to include more details of the simulated annealing algorithm, move set, runtime, and compute usage)

4.e. Line 67: This reviewer believes that 1000^1000 should be replaced by 1000!
> Good catch! We updated the paper to reflect this.

Reviewer Reports on the First Revision:

Reviewer #1

Thank you for thoroughly addressing my comments and suggestions.

At this point I have one more question / suggestion:
In the rebuttal you state that " We have tried running our RL policies for longer, and as shown in Figure 4, the RL policy fully
converges after ~40 hours and does not benefit from additional optimization time."
From the data shown in Figure 4 it does not look like the policy trained from scratch has actually converged at 40 hours.
Instead it looks like it would in fact *overtake* the pre-trained policy if both were trained for another 40 hours. Is this something you have tried? If so, what happened? If not, why not?

To be clear, I don't think this would invalidate the main claims in the paper, but it would be great to add this highly relevant datapoint before publication.

I would also appreciate if more of the training curves for the previous attempts (REINFORCE, DQN, DDQN) could be added to the paper since it is rare to see side-by-side comparisons of all of these methods and PPO on a difficult benchmark problem.

Reviewer #2

The authors did a detailed, commendable job addressing the various comments and suggestions raised in my earlier review, adding new results, providing more insights, contrasting against related works, and clarifying their evaluation methodology.

Reviewer #3

This review focuses on points C (data and methodology), D (appropriate use of statistics and treatment of uncertainties), E (conclusions: robustness, validity, reliability) and F (suggested improvements: experiments, data for possible revision). Points A, B, G, H have been addressed in the previous review and by other reviewers.

The authors have provided a lengthy rebuttal to the reviewer comments, with a number of new details and clarifications. This reviewer would like to express appreciation for the authors' efforts.

----- Regarding the rebuttal to Reviewer 3, Comment 1.a.
The authors write: "Our floorplan solutions are in the product tapeout of a recent-generation TPU chip" is accurate …. Note that the placement results on TPU-v4 in Table 1 are reported post-PlaceOpt, but our production placements for TPU-5 were validated externally all the way to tapeout."
This reviewer understands the authors' statements to mean: "Macro placements determined by the RL macro placer are generated and frozen as shown in Figure 7, fine-tuned in place with SA as described in Section 8.3, and used in the tapeout of the TPU-5 chip."
If this reviewer's understanding above is correct, then hearty congratulations to the authors are in order. (If not correct, then clarification would be appreciated.)

----- Regarding the rebuttal to Reviewer 3, Comment 1.b.
The authors write: "More specifically, we used an open-source hMETIS implementation and below we …"
This reviewer clicked on the given link but did not find any "open-source" implementation. In fact, the executable that was available at the link indicated that it was not usable for commercial purposes. Could the authors please clarify?

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (1)
The authors write, "Given that the same downstream optimization was applied to each method, for the purposes of comparison, we believe it is appropriate to attribute differences in the final result to differences in the upstream methods. However, we agree that it is interesting to consider how best to attribute the final outcome among the optimization stages of the end-to-end workflow."
This reviewer respectfully suggests that the authors revisit the question of attribution, and "correlation or causation". Please see below.

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (2)
The authors write, "In terms of comparing pre- and post-tool timing, the EDA tool does not allow us to run evaluation without running optimization, so it is impossible for us to provide this directly. In fact, early in the project, we met with the EDA company's engineers multiple times and repeatedly requested this feature, and they were unable to provide it."
This reviewer's original comment was poorly written, and the reviewer apologizes for this. The request was actually very simple: report the final (after the rest of the flow has been run) metrics, post-detailed routing, in addition to what is reported in Table 1 from the post-PlaceOpt (with "early global routing") state of the design. I.e., pre-"tool" would be the end of the flow presented in the paper, and post-"tool" would be the final post-detailed route outcome.
(The authors have made it clear (Line 840, Figure 7, Table 5, Line 874, etc.) that results are post-PlaceOpt, which explains the differences in the Area column of Table 1. This said, the senior author of this paper presented details such as "Buf + Inv" Area (sq. um) and "Route DRC violations" in the public talk recounted in T. P. Morgan, "Google Teaches AI to Play the Game of Chip Design", TheNextPlatform, February 20, 2020. This reviewer believes that final metrics such as those previously presented by the senior author would help flesh out Table 1 data and provide better understanding to readers. Such information is commonly presented in the VLSI CAD literature, since there is no comparison between competing EDA tools that would constitute "benchmarking".)

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (3)
The authors write, "Although we do not optimize directly for timing or power, one explanation for the performance of our method is that, by providing layouts with lower wirelength, routing congestion, and density, we make it easier for the downstream tool to optimize for overall quality of result."
This reviewer has studied the authors' rebuttal here and to Reviewer 1's comment (page 13/28 in the Rebuttal PDF file). Could the authors comment on the impact of using hMetis to partition into clusters of ~1000s of standard-cell instances? The literature notes that hMetis-based clustering not only reduces problem size (instance complexity), but also helps to "prevent mistakes" - e.g., splitting up timing paths.

----- Regarding the rebuttal to Reviewer 3, 2.c. (4)

The authors write, “One way to probe the appropriateness of attributing final QoR improvements to the RL algorithm is to vary properties of the algorithm and observe the impact on these final metrics. To that end, we performed a new ablation study (Table 5), which showed that increasing the congestion weight in our reward function affects congestion of the final placements (after EDA tool PlaceOpt).”
This reviewer thanks the authors for providing the further information in their revision. In the new Table 5, with Congestion Weight = 0.01 the WNS is 163ps, and with Congestion Weight = 0.1 the WNS is 154ps. The Table 5 caption specifies “Effect of different cost trade-offs on the post-PlaceOpt performance of Block 1 in Table 1.”
Section 8.1 (Lines 840-841) specifies, “The final metrics in Table 1, are reported after PlaceOpt, meaning that global routing has been performed by the EDA tool.”
Section 2 (Lines 146-147) states that “In our experiments, congestion weight λ is set to 0.01, …”
It appears to this reviewer that the authors are reporting quite different results for Block 1 with Congestion Weight = 0.01, in Table 1 (WNS = 84ps) and in the first line of Table 5 (WNS = 163ps). At the same time, the other metrics for Block 1 are similar between Table 1 / Table 5: TNS 23.3 / 22.85, Cong(H) 0.34 / 0.30, and Cong(V) 0.03 / 0.03.
Could the authors confirm that the solutions with WNS = 163ps and WNS = 154ps would have been deemed “unusable” and grayed out if reported in the Table 1 context? Further, Section 8.4 of the revision states, “A congestion weight of 0.1 represents a sweet spot in this case, as routing congestion is already low, but wirelength has not yet overly degraded, which together contribute to lower TNS as well.” The authors do not mention WNS in Section 8.4, but this reviewer believes that some commentary would be appropriate.

----- Regarding the rebuttal to Reviewer 3, Comment 2.d.ii.
The authors have changed “100ps” in the original submission to “150ps” in the revision, in the Table 1 caption and at Line 329. In their rebuttal, the authors mention “... or WNS significantly above 15% of the total clock cycle (~150ps) is nearly impossible …” (In the rebuttal, the authors do not comment on having made the change from “100ps” to “150ps”.)
The Manual solutions in Table 1 are characterized (Lines 334-338) as follows. “Table 1 also shows the results for the manual baseline, which is the actual production design of the previous TPU chip. This baseline was generated by the TPU’s physical design team, and involved many iterations of placement optimization, guided by feedback from a commercial EDA tool over a period of several months.” This reviewer observes that in Table 1, Block 1 and Block 5 have Manual WNS values that exceed the “100ps” threshold but not the “150ps” threshold. These blocks are not grayed out in Table 1 of the original submission. Was that an error in the original submission, and/or has the true criterion been “150ps” all along? (It is this reviewer’s understanding that according to the “100ps” criterion, the Manual solutions for Blocks 1 and 5 would have been “unusable”. Obviously they were used, and indeed they are not “unusable” when the “150ps” threshold is applied.) Please clarify.

----- Regarding the rebuttal to Reviewer 3, 2.d.iii.
The authors write, “We did not use the timing-driven capability of RePlAce, and we added a statement to this effect in Section 4.4.”
This reviewer respectfully claims that this begs a question of why RePlAce results could then be “grayed out” based on timing. See also 2.c (4) and 2.d.ii, above, and 2.e, below.

----- Regarding the rebuttal to Reviewer 3, 2.e.
The authors write, “Because evaluation with commercial EDA tools is extremely expensive (takes up to 72 hours with an EDA license that costs ~$MM/year), we generate and evaluate just one placement for each block. We did not perform any hyper-parameter optimization to generate the results in Table 1, nor did we perform any denoising to generate the final metrics. As we discuss in response to your earlier question (2.c), we believe that these results are meaningful because each method is evaluated in the same manner, and these metrics are what physical designers use in practice to decide which designs to tapeout. As stated above, to improve the clarity of Table 1, we employed criteria provided by the TPU physical design team to “gray out” placements that they consider unusable. To address questions about

the robustness of our method to noise, we also performed sensitivity analysis on the impact of random seed, which we describe below and in the newly added Table 6 in the manuscript."
This reviewer thanks the authors for providing the further information in their revision. In Table 6, there are eight Congestion values that range from 0.9168 to 1.0251. To this reviewer's best understanding, this is nearly an 11% range; the significance of the statistics of eight numbers is unclear. (In this reviewer's opinion, Table 6 and Table 5 strongly reinforce the reviewer's comments on the original submission.)
This reviewer understands that the run with Congestion 1.0251 would have been "grayed out" in Table 1, if seed "222" happened to have been the one run that was used in the authors' study. Could the authors please confirm this reviewer's understanding? (Would a comment in the paper be warranted?)
This reviewer also understands that wirelength varies by ~5% (.1237 / .1174 = 1.054) over the eight runs in Table 6. Could the authors please comment on significance of comparisons or inferences based on the data in Table 1? (Please refer also to 2.c (4).)

----- Regarding the rebuttal to Reviewer 3, 2.f.
The authors provide many details of their SA implementation, for which this reviewer is very appreciative. [In this reviewer's opinion, the SA description still needs move type generation probabilities, clarification of how 5 seeds are used by the 80 workers, and other details to enable reproducibility.]
This reviewer would like to focus on the authors' statement, "A macro action takes one of three forms: swap, shift, and rotate. Swap selects two macros at random and swaps their locations, if feasible. Shift selects a macro at random and shifts that macro to a neighboring location (left, right, up, or down). Rotate selects a new orientation for the macro at random ('north', 'flipped-north', 'south', 'flipped-south', 'east', 'flipped-east', 'west', 'flipped-west') , regenerating the random orientation if it happens to be identical to the original one." [The Swap and Shift definitions are in Section 8.8 of the revision; the Rotate definition seems missing at Line 989?]
The authors separately write (Section 8.3, Lines 881-892): "In the production flow, we use the same reinforcement learning method described in the manuscript and the same EDA workflow to place standard cells. Although the RL placements were already comparable to manual designs, we performed an additional fine-tuning step with simulated annealing to further boost performance, which helped to improve macro orientation, as we do not currently perform rotations in RL. Adding this fine-tuning step improved wirelength by an average of 1.07% (stddev=.04%), slightly reduced timing (average 1.18ns reduction in TNS (stddev=2.4ns)), and negligibly affected congestion (less than 0.02% variation in vertical or horizontal congestion in all cases). The SA fine-tuning had the same action space and hyperparameters as described in Section XX." ["Section XX" = Section 8.8?]
This reviewer has four questions.
Q1. Referring to Section 8.3: Given that the RL did not perform any rotations, could the authors please comment on how many rotations (relative to the orientations in the RL placement) are seen in the post-fine-tuning-with-SA macro placement?
Q2. Referring to Section 8.8 and the authors' description above: Could the authors please comment on whether both {*north, *south} and {*east, *west} orientations are seen in any of the results reported in Table 7 (Replacing Deep RL with SA) or in any of Tables 1, 5 and 6?
Q3. This reviewer believes that it is odd for (memory) macros to be rotatable in any recent technology (certainly, from 45nm onward). Could the authors please comment on their use of rotations in the move set of the SA macro placement and in their actual production flow for TPU product chips (i.e., extending past the "Freeze Macro Locations" step of Figure 7)?
Q4. This reviewer speculatively notes that the described wirelength improvements of the fine-tuning step (~1%, Lines 887-888) might be consistent with "mirroring", which is often seen when memories are situated "back-to-back". (This is relevant when pins emerge only on one long side, so edges with pins face each other across "channels", and edges with no pins can abut "back-to-back". Note that the macro-to-macro spacing becomes uneven with this common placement strategy.) Could the authors comment on how their use of the fixed, pre-"mirroring" (i.e., pre-fine-tuning) macro locations in product tapeouts is consistent (or, not) with being able to "mirror" macros in place? [Refer Lines 991-992: "while keeping

macro locations fixed, just as we do in our RL method.”]

----- Regarding the rebuttal to Reviewer 3, 4.a (Point G.)
The authors write, “Yes, thank you for pointing that out! We’ve fixed these errors in the references.”
The authors still have not fixed all references. See [47] - this is Will Naylor, Ross Donelly, Lu Sha = the very famous log-sum-exp patent. Please re-check more carefully. In this reviewer’s opinion, the patent number (US6301693B1) might be given in the citation.

----- Regarding the rebuttal to Reviewer 3, 4.c Line 220:
The authors write: “We used the following Ariane design (https://github.com/pulp-platform/ariane), and mapped all logical memories to physical memories of size 256x16, resulting in a higher number of macros (133 macros). We added a new section to the manuscript which includes these details (Section 8.2). In terms of the ordering, we randomly select a valid topological sort, but as noted in the paper, this is an area in which further performance gains could most likely be achieved (Section 6.3.3 Selection of Macro Order).”
This reviewer appreciates the additional details, and would like to observe that an easy “ablation study” pertaining to Section 6.3.3. is immediately possible with the Ariane testcase. The ties among 130+ identical macro areas can be broken by making epsilon changes in area, essentially to (randomly) order the macros’ processing according to the procedure described in Section 6.3.3. In other words, playing with LEF masters or other means can avoid the “break ties using a topological sort”. This reviewer would be very interested to see the variation in solution quality that is seen across, say, 10 random orderings (realized by descending size) of the 130+ Ariane macros. The “ablation study” here would shed light on contributions of “break ties using a topological sort” vs. “sort macros by descending size”.

Author Rebuttals to First Revision:

Referees' comments:

Referee #1 (Remarks to the Author):

Thank you for thoroughly addressing my comments and suggestions.

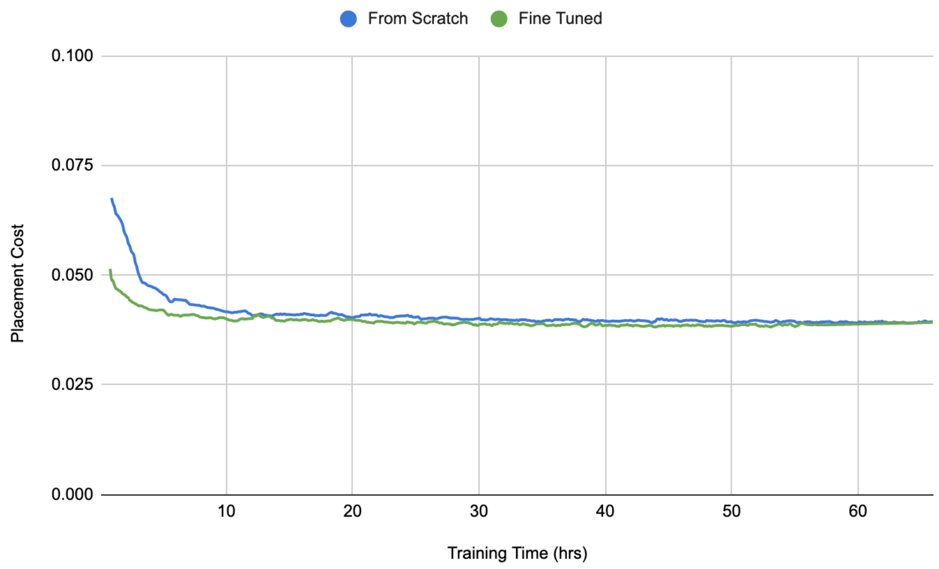
> We are glad to hear that we have addressed your comments and suggestions, and thanks again for your insightful review! We really appreciate you taking the time to share your RL expertise and help us improve the manuscript.

At this point I have one more question / suggestion:
In the rebuttal you state that " We have tried running our RL policies for longer, and as shown in Figure 4, the RL policy fully
converges after ~40 hours and does not benefit from additional optimization time."
From the data shown in Figure 4 it does not look like the policy trained from scratch has actually converged at 40 hours.
Instead it looks like it would in fact *overtake* the pre-trained policy if both were trained for another 40 hours. Is this something you have tried? If so, what happened? If not, why not?
To be clear, I don't think this would invalidate the main claims in the paper, but it would be great to add this highly relevant datapoint before publication.

> We agree that it’s interesting to run RL for longer to see the full convergence properties of the pre-trained policy as well as one that was trained from scratch. Although we weren’t able to find the exact checkpoints for the policies in Figure 4 (as those experiments were run over a year ago), we did plot placement cost over time for another TPU block. As you can see, both policies have fully

converged and produce placements of comparable quality after roughly 60 hours.

It is very possible that on a different block a policy trained from scratch could eventually overtake the pre-trained policy. However, from a practical perspective, performance gains seen only after multiple days would not be useful to a production chip design team.



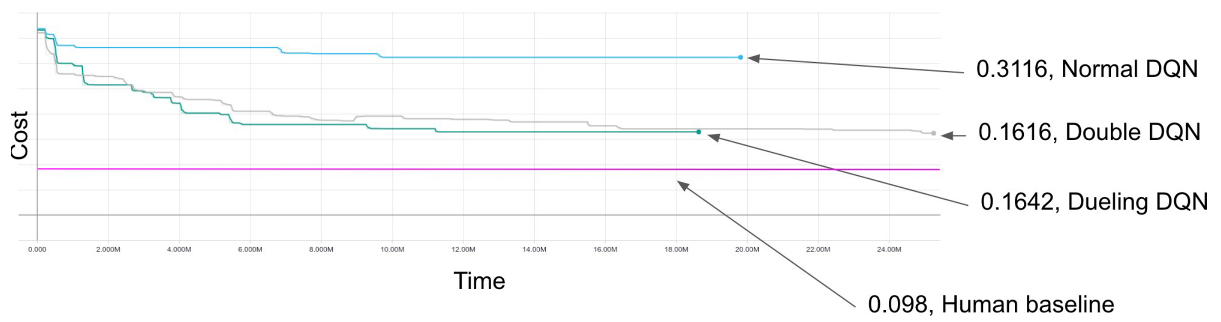
I would also appreciate if more of the training curves for the previous attempts (REINFORCE, DQN, DDQN) could be added to the paper since it is rare to see side-by-side comparisons of all of these methods and PPO on a difficult benchmark problem.

> We agree that it would be valuable to see more side-by-side comparisons of widely used RL algorithms, such as PPO, REINFORCE, DQN, and DDQN, especially on challenging benchmarks. In the very early days of the project, we ran these comparisons and plotted relative performance. You can see our comparison against DQN variants in the plots below, which show a huge gap between DQN and PPO (0.1642 vs. 0.1112, meaning that PPO achieved 32% higher reward). We apologize for the low-quality of these graphs, as they were plotted a long time ago to guide our own decision-making and with no intention of including them in a publication. Given the strong performance of PPO, we developed a distributed PPO implementation for this task, and have since significantly updated our RL architecture, reward functions, and overall problem formulation.

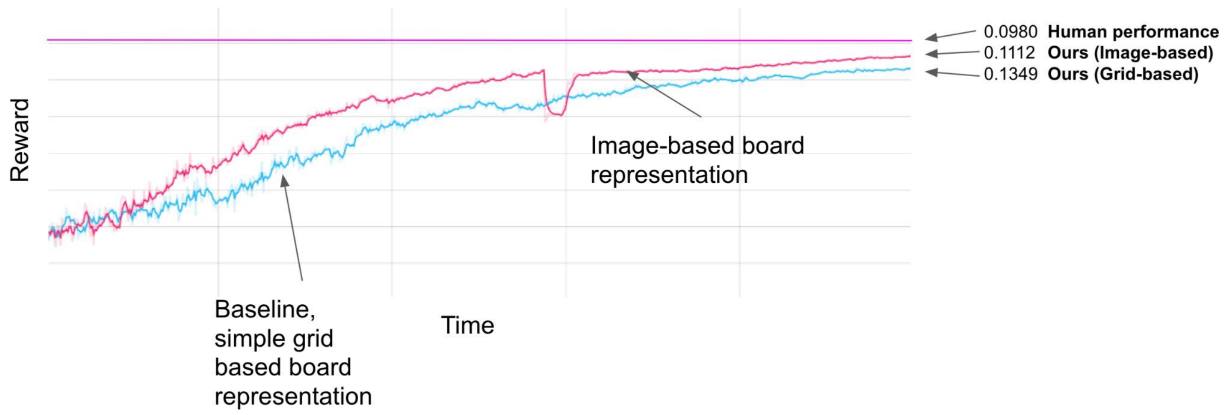
In response to your question (and in recognition of the fact that after such significant changes to our overall method, it is very possible that these other algorithms would now outperform PPO), we did attempt to revive our implementations of other RL optimization algorithms. However, we found that, for example, DQN no longer converges, and we would need to dedicate significant additional development and tuning effort in order to produce fair side-by-side comparisons.

In chip floorplanning, where the search space is massive and the vast majority of solutions are invalid, PPO is an effective choice because it allows for safe and efficient exploration, but we do think it is possible that other RL algorithms could enable even better performance.

DQN Variants vs. Human Baseline on an Edge Block



PPO vs. Human on an Edge Block



Referee #2 (Remarks to the Author):

The authors did a detailed, commendable job addressing the various comments and suggestions raised in my earlier review, adding new results, providing more insights, contrasting against related works, and clarifying their evaluation methodology.

> Thank you! We greatly appreciated your comments and suggestions, and are glad to hear that we have now addressed them. In particular, your suggestion to add more physical design insights was very helpful, and we really appreciate you taking the time to share your physical design expertise and help us strengthen the manuscript.

Referee #3 (Remarks to the Author):

This review focuses on points C (data and methodology), D (appropriate use of statistics and treatment of uncertainties), E (conclusions: robustness, validity, reliability) and F (suggested improvements: experiments, data for possible revision). Points A, B, G, H have been addressed in the previous review and by other reviewers.

The authors have provided a lengthy rebuttal to the reviewer comments, with a number of new details and clarifications. This reviewer would like to express appreciation for the authors' efforts.

> Thank you! We also greatly appreciate your thoughtful review, and we wanted to once again express our appreciation for you taking the time to share your expertise in physical design. Your efforts have certainly helped us to improve the quality and clarity of our manuscript.

----- Regarding the rebuttal to Reviewer 3, Comment 1.a.
The authors write: "Our floorplan solutions are in the product tapeout of a recent-generation TPU chip" is accurate …. Note that the placement results on TPU-v4 in Table 1 are reported post-PlaceOpt, but our production placements for TPU-5 were validated externally all the way to tapeout."
This reviewer understands the authors' statements to mean: "Macro placements determined by the RL macro placer are generated and frozen as shown in Figure 7, fine-tuned in place with SA as described in Section 8.3, and used in the tapeout of the TPU-5 chip."
If this reviewer's understanding above is correct, then hearty congratulations to the authors are in order. (If not correct, then clarification would be appreciated.)

> Yes, your understanding of our method's role in the TPU-v5 tapeout is correct!

----- Regarding the rebuttal to Reviewer 3, Comment 1.b.
The authors write: "More specifically, we used an open-source hMETIS implementation and below we …" This reviewer clicked on the given link but did not find any "open-source" implementation. In fact, the executable that was available at the link indicated that it was not usable for commercial purposes. Could the authors please clarify?

> Google has a licensing agreement with hMETIS for this project and several others, so we do pay licensing fees for use of hMETIS in production. As far as we know, the features that we use (as specified in "Experimental Setup" - note that we have removed number subheadings to comply with Nature's style guide) are also available in open-source hMETIS. In order to be fully transparent, we have updated our "Experimental Setup" section to clarify that we use the licensed version of hMETIS.

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (1)
The authors write, "Given that the same downstream optimization was applied to each method, for the purposes of comparison, we believe it is appropriate to attribute differences in the final result to differences in the upstream methods. However, we agree that it is interesting to consider how best to attribute the final outcome among the optimization stages of the end-to-end workflow."
This reviewer respectfully suggests that the authors revisit the question of attribution, and "correlation or causation". Please see below.

> Thank you for clarifying below what you meant by pre- and post-tool timing metrics. We revisit this question in our responses below.

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (2)
The authors write, "In terms of comparing pre- and post-tool timing, the EDA tool does not allow us to run evaluation without running optimization, so it is impossible for us to provide this directly. In fact, early in the project, we met with the EDA company's engineers multiple times

and repeatedly requested this feature, and they were unable to provide it."
This reviewer's original comment was poorly written, and the reviewer apologizes for this. The request was actually very simple: report the final (after the rest of the flow has been run) metrics, post-detailed routing, in addition to what is reported in Table 1 from the post-PlaceOpt (with "early global routing") state of the design. I.e., pre-"tool" would be the end of the flow presented in the paper, and post-"tool" would be the final post-detailed route outcome.
(The authors have made it clear (Line 840, Figure 7, Table 5, Line 874, etc.) that results are post-PlaceOpt, which explains the differences in the Area column of Table 1. This said, the senior author of this paper presented details such as "Buf + Inv" Area (sq. um) and "Route DRC violations" in the public talk recounted in T. P. Morgan, "Google Teaches AI to Play the Game of Chip Design", TheNextPlatform, February 20, 2020. This reviewer believes that final metrics such as those previously presented by the senior author would help flesh out Table 1 data and provide better understanding to readers. Such information is commonly presented in the VLSI CAD literature, since there is no comparison between competing EDA tools that would constitute "benchmarking".)

> The following information is confidential and cannot be added to the manuscript, but we have been authorized to disclose that the TPU team hands off the layouts post-PlaceOpt to a third party vendor, which performs detailed routing and sign off. It was difficult for us to obtain permission to share this information and we trust that you will keep this in confidence, but we felt that this context was important in answering your question.

In Table 1, we show results on the previous generation of TPU (TPU-v4), so that we can compare against manual placements generated by the TPU team. On TPU-v4, our method was generating placements with strong QoR metrics (post-PlaceOpt), but their unusual shape was concerning to the TPU team, so they decided to send just one of our placements to this third party vendor for detailed routing and signoff, along with their own manual placement. Our placement met all of the QoR requirements (with slightly better metrics than the human placement) and was closable. Generally speaking, this process is very expensive, but in this one case, both our TPU-v4 placement and the human expert placement were sent to this external vendor for detailed routing and signoff, producing the results that our senior author presented on February 20, 2020.

In TPU-v5, on the other hand, placements generated by our method were sent for detailed routing and signoff (with no accompanying manual placement), and have now been validated all the way to tapeout.

However, to provide insight into the relationship between pre- and post-tool metrics, we provide these metrics for an example TPU-v5 block, as placements generated by our method were taped out in TPU-v5, and therefore were taken all the way post-RouteOpt.

Below, we show a blurred placement of this TPU-v5 block, with the yellow/orange rectangles corresponding to macros. Note that no macros were moved in the course of the external vendor's detailed routing optimization.

Below we report the metrics we observed post-PlaceOpt (pre-tool), as well as the post-PlaceOpt (pre-tool) and post-RouteOpt (post-tool) metrics reported by the external vendor. Note that the vendor did not share wirelength metrics, but we were pleased to have been able to obtain and share the timing and congestion metrics below to answer your question.

Core Block	Step	Wirelength (um)	WNS (ps)	TNS (ns)	Congestion (Vertical)	Congestion (Horizontal)
fp_hde_top	PlaceOpt (Ours)	7.90E+06	-0.002	0	0.00	0.00
fp_hde_top	PlaceOpt (Vendor)	-	-0.008	-0.015	0.00	0.01
fp_hde_top	RouteOpt (Vendor)	-	0.00	0.00	0.00	0.00

----- Regarding the rebuttal to Reviewer 3, Comment 2.c. (3)
The authors write, "Although we do not optimize directly for timing or power, one explanation for the performance of our method is that, by providing layouts with lower wirelength, routing congestion, and density, we make it easier for the downstream tool to optimize for overall quality of result."
This reviewer has studied the authors' rebuttal here and to Reviewer 1's comment (page 13/28 in the Rebuttal PDF file). Could the authors comment on the impact of using hMetis to partition into clusters of ~1000s of standard-cell instances? The literature notes that hMetis-based clustering not only reduces problem size (instance complexity), but also helps to "prevent mistakes" - e.g., splitting up timing paths.

> We completely agree and have updated the manuscript to include this additional insight, namely that hMETIS-based clustering likely helps not only with reducing problem size, but also helps "prevent mistakes".

Here is the updated text:
As has been suggested in the literature [1], such clustering helps not only with reducing problem size, but also helps "prevent mistakes" (e.g. prevents timing paths from being split apart).

[1] A. Kahng. Futures for Partitioning in Physical Design. In Proc. ACM/IEEE intl. Symp. on Physical Design, pages 190–193, 1998.

----- Regarding the rebuttal to Reviewer 3, 2.c. (4)
The authors write, “One way to probe the appropriateness of attributing final QoR improvements to the RL algorithm is to vary properties of the algorithm and observe the impact on these final metrics. To that end, we performed a new ablation study (Table 5), which showed that increasing the congestion weight in our reward function affects congestion of the final placements (after EDA tool PlaceOpt).”
This reviewer thanks the authors for providing the further information in their revision. In the new Table 5, with Congestion Weight = 0.01 the WNS is 163ps, and with Congestion Weight = 0.1 the WNS is 154ps. The Table 5 caption specifies “Effect of different cost trade-offs on the post-PlaceOpt performance of Block 1 in Table 1.”
Section 8.1 (Lines 840-841) specifies, “The final metrics in Table 1, are reported after PlaceOpt, meaning that global routing has been performed by the EDA tool.”
Section 2 (Lines 146-147) states that “In our experiments, congestion weight λ is set to 0.01, …”
It appears to this reviewer that the authors are reporting quite different results for Block 1 with Congestion Weight = 0.01, in Table 1 (WNS = 84ps) and in the first line of Table 5 (WNS = 163ps). At the same time, the other metrics for Block 1 are similar between Table 1 / Table 5: TNS 23.3 / 22.85, Cong(H) 0.34 / 0.30, and Cong(V) 0.03 / 0.03.
Could the authors confirm that the solutions with WNS = 163ps and WNS = 154ps would have been deemed “unusable” and grayed out if reported in the Table 1 context?

> WNS tends to vary more than other metrics, which makes sense given that a single path can significantly degrade WNS, whereas TNS and other metrics are more cumulative and therefore more stable.

To answer your question, these placements would not have been grayed out in the Table 1 context, as their WNS is not significantly above 150ps. That being said, as discussed below, these thresholds are engineering practices of the TPU team and may not be universal. Therefore, we have decided to no longer “gray out” placements, as you can see in the updated Table 1.

Further, Section 8.4 of the revision states, “A congestion weight of 0.1 represents a sweet spot in this case, as routing congestion is already low, but wirelength has not yet overly degraded, which together contribute to lower TNS as well.” The authors do not mention WNS in Section 8.4, but this reviewer believes that some commentary would be appropriate.

> Thanks for noticing this omission. We’ve update the text as follows:

“A congestion weight of 0.1 represents a sweet spot in this case, as routing congestion is already low, but wirelength has not yet overly degraded, which together contribute to lower TNS and WNS as well.”

----- Regarding the rebuttal to Reviewer 3, Comment 2.d.ii.
The authors have changed “100ps” in the original submission to “150ps” in the revision, in the Table 1 caption and at Line 329. In their rebuttal, the authors mention “... or WNS significantly above 15% of the total clock cycle (~150ps) is nearly impossible …” (In the rebuttal, the authors

do not comment on having made the change from “100ps” to “150ps”.)

> In response to your earlier questions about WNS thresholds, we met with the physical design team, and learned that they had earlier misinformed us, and that the WNS threshold is in fact 150ps, not 100ps.

We mentioned their reasoning for the 150ps threshold in response to question 2.d.ii in the previous review, but we apologize for not making it more clear that we had corrected this number from 100ps to 150ps. Regardless, as described above, we have reflected on several of the points that you made and have decided to no longer “gray out” placements.

The Manual solutions in Table 1 are characterized (Lines 334-338) as follows. “Table 1 also shows the results for the manual baseline, which is the actual production design of the previous TPU chip. This baseline was generated by the TPU’s physical design team, and involved many iterations of placement optimization, guided by feedback from a commercial EDA tool over a period of several months.” This reviewer observes that in Table 1, Block 1 and Block 5 have Manual WNS values that exceed the “100ps” threshold but not the “150ps” threshold. These blocks are not grayed out in Table 1 of the original submission. Was that an error in the original submission, and/or has the true criterion been “150ps” all along? (It is this reviewer’s understanding that according to the “100ps” criterion, the Manual solutions for Blocks 1 and 5 would have been “unusable”. Obviously they were used, and indeed they are not “unusable” when the “150ps” threshold is
applied.) Please clarify.

> As you suggested, the true criterion all along was that WNS could not be significantly above 150ps. This was reflected in the placements that the physical design team chose to gray out, and when we met with them they confirmed that 150ps was the threshold that they had used. We apologize for the confusion.

----- Regarding the rebuttal to Reviewer 3, 2.d.iii.
The authors write, “We did not use the timing-driven capability of RePlAce, and we added a statement to this effect in Section 4.4.”
This reviewer respectfully claims that this begs a question of why RePlAce results could then be “grayed out” based on timing. See also 2.c (4) and 2.d.ii, above, and 2.e, below.

> As stated above, we have decided to no longer “gray out” any placements.

----- Regarding the rebuttal to Reviewer 3, 2.e.
The authors write, “Because evaluation with commercial EDA tools is extremely expensive (takes up to 72 hours with an EDA license that costs ~$MM/year), we generate and evaluate just one placement for each block. We did not perform any hyper-parameter optimization to generate the results in Table 1, nor did we perform any denoising to generate the final metrics. As we discuss in response to your earlier question (2.c), we believe that these results are meaningful because each method is evaluated in the same manner, and these metrics are what physical designers use in practice to decide which designs to tapeout. As stated above, to improve the clarity of Table 1, we employed criteria provided by the TPU physical design team to “gray out” placements that they consider unusable. To address questions about the robustness of our method to noise, we also performed sensitivity analysis on the impact of random seed, which we describe below and in the newly added Table 6 in the

manuscript."
This reviewer thanks the authors for providing the further information in their revision. In Table 6, there are eight Congestion values that range from 0.9168 to 1.0251. To this reviewer's best understanding, this is nearly an 11% range; the significance of the statistics of eight numbers is unclear. (In this reviewer's opinion, Table 6 and Table 5 strongly reinforce the reviewer's comments on the original submission.)
This reviewer understands that the run with Congestion 1.0251 would have been "grayed out" in Table 1, if seed "222" happened to have been the one run that was used in the authors' study.

Could the authors please confirm this reviewer's understanding? (Would a comment in the paper be warranted?)

> While in Table 1 we report congestion as measured by the commercial EDA tool, in Table 6 we show proxy congestion values, because we wanted to show the impact of random seed on the metrics observed by the RL agent (and proxy cost is also far less expensive to evaluate). Proxy congestion correlates with "true" congestion (as measured by a commercial EDA tool), but is approximate, is on a different scale, and does not have precisely the same meaning. As described in the "Reward" Section under "Detailed Methodology" (formerly Section 6.4.2), the proxy routing congestion cost is the average congestion of the top 10% most congested grid cells. We have updated Extended Data Tables 5 and 6 (formerly Tables 6 and 7), as well as the text of "Robustness to Noise" and "Comparing with Simulated Annealing", to make it clear that we are reporting proxy congestion in Extended Data Tables 5 and 6, rather than the congestion measured by the commercial EDA tool. Thanks for helping us realize that this was unclear!

In terms of proxy congestion values reported in Extended Data Table 5, the standard deviations for wirelength (0.0019) and routing congestion (0.0346) are very low. Although measuring proxy cost is faster and less expensive than evaluating with a commercial EDA tool (which can take up to 72 hours per run), each of the runs in Extended Data Tables 5 and 6 still required 16 GPUs for up to 6 hours each. We therefore chose 8 runs because we felt that this represented a meaningful number, but wouldn't consume all of our compute resources (which we wanted to reserve for other experiments in the previous rebuttal). Also, for what it's worth, we've never observed post-PlaceOpt congestion (horizontal or vertical) above 0.5 for any TPU block placed by our method.

This reviewer also understands that wirelength varies by ~5% (.1237 / .1174 = 1.054) over the eight runs in Table 6. Could the authors please comment on significance of comparisons or inferences based on the data in Table 1? (Please refer also to 2.c (4).)

> As discussed above, the metrics in Extended Data Table 5 are proxy wirelength, congestion, and density, and although correlated with QoR metrics reported in Table 1 (commercial EDA tool), they do not have the same scale or meaning. While the proxy wirelength does range from 0.1237 to 0.1174 over 8 runs, the standard deviation is only 0.0019, so even for eight runs, these results do seem meaningful.

----- Regarding the rebuttal to Reviewer 3, 2.f.
The authors provide many details of their SA implementation, for which this reviewer is very appreciative. [In this reviewer's opinion, the SA description still needs move type generation

probabilities, clarification of how 5 seeds are used by the 80 workers, and other details to enable reproducibility.]

> We are happy to provide these additional details in order to enable reproducibility.

In terms of the move type generation probabilities, we apply a uniform probability over the three move types, meaning that at each time step there is a ⅓ chance of swapping, a ⅓ chance of shifting, and a ⅓ chance of flipping.

To control for computational resource usage, we ran multiple SA experiments sweeping different hyperparameters, including max temperature (1e-5, 3e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4, 1e-3), max episode length (5e4, 1e5), and random seed (5 different random seeds), and report the best result in Extended Data Table 6.

In terms of how the 5 seeds are used by the 80 workers, each worker corresponds to a particular choice of the 5 seeds x 2 different episode lengths x 8 different max temperatures.

The random seed affects the outcome of the following operations:

a. **Macro selection**: In each iteration (step), we conduct 2*N macro actions (swap, shift, or flip), where N=number of macros. For shift and flip, one random macro is selected (with replacement). For swap, two random macros are selected (with replacement) to swap positions (if legal).
b. **Location / flipping selection**: For shifts, one of the eight legal neighboring grid cells (i.e., +/-1 row/col) is randomly selected. For flips, one of the three valid flips is randomly selected (flip horizontal, flip vertical, or flip both vertical and horizontal).
c. **State acceptance**: When the new state is worse than the previous state, transition to the new state with a probability (temperature dependent).

We've updated "Comparing with Simulated Annealing (formerly Section 8.8) to clarify and include these additional details for reproducibility.

This reviewer would like to focus on the authors' statement, "A macro action takes one of three forms: swap, shift, and rotate. Swap selects two macros at random and swaps their locations, if feasible. Shift selects a macro at random and shifts that macro to a neighboring location (left, right, up, or down). Rotate selects a new orientation for the macro at random ('north', 'flipped-north', 'south', 'flipped-south', 'east', 'flipped-east', 'west', 'flipped-west') , regenerating the random orientation if it happens to be identical to the original one." [The Swap and Shift definitions are in Section 8.8 of the revision; the Rotate definition seems missing at Line 989?]

> Thanks for pointing out this omission! We've updated the manuscript to include the Rotate (Flip/Mirror) definition in "Comparing with Simulated Annealing".

The authors separately write (Section 8.3, Lines 881-892): "In the production flow, we use the same reinforcement learning method described in the manuscript and the same EDA workflow to place standard cells. Although the RL placements were already comparable to manual designs, we performed an additional fine-tuning step with simulated annealing to further boost performance, which helped to improve macro orientation, as we do not currently perform rotations in RL. Adding this fine-tuning step improved wirelength by an average of 1.07%

(stddev=.04%), slightly reduced timing (average 1.18ns reduction in TNS (stddev=2.4ns)), and negligibly affected congestion (less than 0.02% variation in vertical or horizontal congestion in all cases). The SA fine-tuning had the same action space and hyperparameters as described in Section XX." ["Section XX" = Section 8.8?]

> Yes, you are correct that Section XX is Section 8.8, though of course we have now been asked to remove all numbering for subheadings.

This reviewer has four questions.

> First off, thanks for pointing out that "rotation" was a confusing choice of word. We want to clarify that we only allow mirroring across the X axis, across the Y axis, or across both the X and Y axes. Therefore, flipping or mirroring, rather than "rotating", better describes our move set, and we have updated our manuscript to make this more clear.

Q1. Referring to Section 8.3: Given that the RL did not perform any rotations, could the authors please comment on how many rotations (relative to the orientations in the RL placement) are seen in the post-fine-tuning-with-SA macro placement?

> Thanks to your comment, we performed further analysis and discovered that because most macros have ports on their left and/or right sides (rather than their top or bottom), many flips (e.g. mirroring horizontally) have little impact on placement quality. On the other hand, a vertical flip can kick off a cascade of (vertical) macro flips, inflating the number of flips. Across a set of TPU blocks, an average of 69% of the macro orientations changed (stddev=0.048%). Note that we recently deprecated SA in our production workflow, replacing it with a simple greedy heuristic that runs much faster (<5 minutes).

Q2. Referring to Section 8.8 and the authors' description above: Could the authors please comment on whether both {*north, *south} and {*east, *west} orientations are seen in any of the results reported in Table 7 (Replacing Deep RL with SA) or in any of Tables 1, 5 and 6?

> Table 1 and Extended Data Tables 4, 5, and 6 (previously Tables 1, 5, 6, and 7) contain RL-only results for our method, so no flipping / rotation (or any kind of fine-tuning) has been performed. As discussed above, {*east, *west} orientations cannot and do not occur in any table in our manuscript or in our production layouts.

Q3. This reviewer believes that it is odd for (memory) macros to be rotatable in any recent technology (certainly, from 45nm onward). Could the authors please comment on their use of rotations in the move set of the SA macro placement and in their actual production flow for TPU product chips (i.e., extending past the "Freeze Macro Locations" step of Figure 7)?

> As discussed above, "rotation" was a poor choice of word, and we've updated the manuscript to better describe this move as flipping or mirroring. Thanks for your help in improving the clarity of our manuscript!

Q4. This reviewer speculatively notes that the described wirelength improvements of the fine-tuning step (~1%, Lines 887-888) might be consistent with "mirroring", which is often seen when memories are situated "back-to-back". (This is relevant when pins emerge only on one long side, so edges with pins face each other across "channels", and edges with no pins can abut “back-to-back”. Note that the macro-to-macro spacing becomes uneven with this common placement strategy.) Could the authors comment on how their use of the fixed, pre-”mirroring” (i.e., pre-fine-tuning) macro locations in product tapeouts is consistent (or, not) with being able to “mirror” macros in place? [Refer Lines 991-992: “while keeping macro locations fixed, just as we do in our RL method.”]

> This is an interesting and subtle point that you raise. For TPU-v5 production layouts, the TPU team requires a minimum spacing between macros, which we cannot disclose as this is a confidential aspect of our production workflow. To enforce this constraint, the TPU team performs macro “bloating” as part of their standard flow, artificially increasing the width and height of each macro by a small requisite amount. Therefore, freezing macro locations does not interfere with mirroring macros, as there is always enough of a gap, regardless of which edges now have pins.

----- Regarding the rebuttal to Reviewer 3, 4.a (Point G.)
The authors write, “Yes, thank you for pointing that out! We’ve fixed these errors in the references.”
The authors still have not fixed all references. See [47] - this is Will Naylor, Ross Donelly, Lu Sha = the very famous log-sum-exp patent. Please re-check more carefully. In this reviewer’s opinion, the patent number (US6301693B1) might be given in the citation.

> Our sincere apologies for omitting the patent number for log-sum-exp. We did another pass through all of the references, and we think that we’ve fixed all issues, but please let us know if there was something else that you had in mind. Once again, we really appreciate your patience here!

We also took this opportunity to add a number of citations that we had meant to add earlier, so we hope that this has also strengthened the manuscript, especially the related work section. Please let us know if you have any suggestions, as we do want to make sure that we have painted a clear and relatively complete picture of the field’s history, bearing in mind that we are limited in the number of references that we are allowed to include.

----- Regarding the rebuttal to Reviewer 3, 4.c Line 220:
The authors write: “We used the following Ariane design (https://github.com/pulp-platform/ariane), and mapped all logical memories to physical memories of size 256x16, resulting in a higher number of macros (133 macros). We added a new section to the manuscript which includes these details (Section 8.2). In terms of the ordering, we randomly select a valid topological sort, but as noted in the paper, this is an area in which further performance gains could most likely be achieved (Section 6.3.3 Selection of Macro Order).”
This reviewer appreciates the additional details, and would like to observe that an easy “ablation study” pertaining to Section 6.3.3. is immediately possible with the Ariane testcase. The ties among 130+ identical macro areas can be broken by making epsilon changes in area, essentially to (randomly) order the macros’ processing according to the procedure described in Section 6.3.3. In other words, playing with LEF masters or other means can avoid the “break ties using a topological sort”. This reviewer would be very interested to see the variation in solution quality that is seen across, say, 10 random orderings (realized by descending size) of the 130+ Ariane macros. The “ablation study” here would shed light on contributions of “break ties using a topological sort” vs. “sort macros by descending size”.

> Thanks for suggesting this new ablation study! Below, we show 10 random orderings for the Ariane core. As you can see, the standard deviation of wirelength and congestion cost is very low across different macro orderings, perhaps because our edge-based graph convolutional neural network is able to effectively capture netlist topology. This is consistent with what we have observed in earlier work on device placement, where using GNNs greatly reduced the impact of placement ordering.

Ariane:

Random Ordering	Proxy WL cost	Proxy Cong cost
#1	0.1144	0.9515
#2	0.1137	0.9378
#3	0.1138	0.9629
#4	0.1131	0.9166
#5	0.1134	0.915
#6	0.1152	0.9401
#7	0.1127	0.9033
#8	0.1122	0.9294
#9	0.1133	0.9163
#10	0.1145	0.9195
MEAN	0.1136	0.9292
STD	0.0009	0.0186

Reviewer Reports on the Second Version:

Reviewer #3

This reviewer would like to express sincere appreciation for the authors' efforts and patience, as well as the many clarifications. All of this reviewer's comments and questions have been well-addressed. Congratulations are due to the authors on their groundbreaking, landmark work.

One comment: Before publication, please fix Reference [64], Line 1087. The *LAST* names of the authors are Naylor, Donelly and Sha ! This reviewer appreciates the addition of the patent number, but please correct the author names as well. This reviewer apologizes for not spelling out the error more clearly in the previous reviews.

Again, thanks and congratulations to the authors.