Stronger Baselines for Evaluating Deep Reinforcement Learning in Chip Placement

Abstract

Deep Reinforcement Learning (DRL) has demonstrated stunning success in game-playing and several applied optimization problems. Improving chip designs is an attractive next application, as illustrated by a recently proposed DRL technique for chip layout optimization based on EdgeGNNs. To evaluate this technique, we develop stronger baselines by enhancing established techniques. Compared to the DRL implementation reported earlier, our stronger baselines produce consistently superior design layouts both on the chip designs used in the DRL paper and on public benchmarks. We also produce competitive layouts with computational resources smaller by orders of magnitude. Ablation studies help explain these wins and point to weaknesses in how DRL is applied. Furthermore, our stronger baselines and our empirical results suggest that the work of human chip designers cannot serve as a strong baseline in scientific settings.

1 Introduction

Deep RL and Electronic Design Automation. The revolutionary improvements brought about by deep learning (DL) methods in machine perception and language tasks, and by deep reinforcement learning (DRL) methods in beating the best human players in games such as Go [28], Shogi, and Chess [27] have led to immense interest in applying deep learning to other challenging domains. One such domain is electronic design automation (EDA), the field that is concerned with developing silicon compilers, software tools that are used to design the integrated circuits which underpin much of modern life. Although there is a long history of applying AI and machine-learning techniques in EDA, only recently has revolutionary success—in the sense of an EDA system being "superhuman" at solving a placement problem—been claimed in a scientific setting [24]. This work, published in the prestigious multi-disciplinary journal Nature (the "Nature paper"), has attracted broad attention. This attention is for good reason.

A natural opportunity. In the early 1990s, due to Very Large Scale Integration (VLSI), integrated circuits (ICs) became so complicated that designing a high-performance IC without software automation became impossible. The field of EDA historically developed by stringing together "point" optimization steps (e.g., logic optimization, technology mapping, placement, sizing and buffering, routing, etc.) into design flows where each point optimization step is typically an NP-hard combinatorial optimization problem solved with sophisticated domain-specific heuristics. In time, these design flows took over most manual tasks of circuit designers leading to modern "RTL-to-GDSII" flows—an industry staple for the last 20 years or so—in which register-transfer level (RTL) design descriptions are automatically converted to complete chip layouts in terms of polygonal shapes (the GDSII data format) while optimizing for or obeying constraints on critical metrics such as area, delay, and power. Although modern chip design flows are largely automated, it is not uncommon for human intervention to (a) customize the design flow (before software tools are used) in unusual cases, or (b) make clever small changes to improve a particular chip design after software tools do their job.

Modern AI techniques in the field of EDA (1) attempt to minimize human intervention, (2) learn how to predict the eventual design quality from the early stages of the design flow, and (3) aim to rival established point-optimization techniques. The first category is a relatively recent development, but design flow customization is already performed by commercial systems such as DSO.ai™

from Synopsys [29]. The second category can help guide the chip design process and make early optimizations more successful (e.g., [34, 20]). The third category of improvements—and the Nature paper addresses this category—leverages ML to improve individual point optimization steps in the chip-design flow. Such results are of broader interest beyond the chip-design community because core EDA tasks illustrate hard combinatorial optimization problems. A key scientific question, then, is if machines can automatically learn to solve these problems faster or better than the state-of-the-art.

A difficult baseline to beat. In the last 50 years, our ability to design ever larger and complex chips has been key to unlocking the full potential of Moore's law leading to dramatic advances in industries such as consumer electronics and entertainment, telecommunications, education, healthcare, automotive, aerospace, and defense. Consequently, Moore scaling was supported by considerable investment into EDA research since the 1970s, both in academia and in industry. As a result, EDA has been absorbing relevant breakthroughs in optimization and AI methods.

In particular, EDA problems have provided fertile ground for testing the efficacy of new general-purpose methods on practical problems. Examples include the adoption of the A* algorithm for wire routing [33], graph partitioning algorithms like Kernighan–Lin [17] and Fiduccia–Mattheyses [12] for automated placement [8, 12, 16], Boolean satisfiability solvers for verification and testing [32], and—of particular significance to this paper—simulated annealing for circuit placement. In fact, chip placement was one of two applications in the 1983 Science paper that proposed simulated annealing [19]. Given the commercial importance of chip design and the sophistication of existing automation for it, EDA tasks offer a good testbed to evaluate potential new fundamental algorithmic advances such as deep reinforcement learning; as Sinatra observed: *"if you can make it there, you can make it anywhere."*

Inspired by the Nature paper, our work seeks stronger optimization techniques for chip layout by drawing upon not only the newest RL methods, but also the rich toolbox of established methods [15, 22]. The results may surprise you.

2 Deep Reinforcement Learning for Chip Design

The EDA problem addressed in the Nature paper is block-level circuit placement which is the problem of assigning locations to circuit components within a block (a portion of a design) so as to minimize a certain objective function, subject to constraints.[1] A related optimization was illustrated by the 1983 Science paper that developed simulated annealing for circuit placement to optimize the length of wires that connect circuit components that must not overlap [19]. While circuits today are much larger than those 40 years ago, and use more wiring, another key complication is the presence of large macro blocks, such as IP blocks, memories, or register files, among numerous small "standard cells." The diversity of scale combines combinatorial block-packing with large-scale optimization, and the resulting challenge has been thoroughly studied in the literature [15, 22, 25] under the names of mixed-size circuit placement and "boulders-and-dust" placement. A common approach is to first optimize a closed-form proxy objective (such as estimates of total wirelength), then perform subsequent layout steps such as wire routing (which may fail), and then evaluate more sophisticated figures of merit, such as the total length of routed wires and circuit timing. Academic and commercial software tools for proxy optimization are widely known and can handle very large circuits with millions of components, and a variety of methods exist to coerce these optimizations to address post-routing figures of merit. It is possible that an experienced human designer with a deep understanding of a particular design may improve upon layouts produced by software tools, but in any case, most of the work is done by software. The strength of modern algorithms for mixed-size placement is that they gradually co-optimize the locations of many components (large and small) instead of making combinatorial decisions early and having to revisit those decisions, as was done by earlier methods.

RL Formulation. Since the task of placing millions of components would entail an intractably large action space for RL, the authors of the Nature paper consider a *reduced* placement problem with the following approximations:

1. **Clustered Graph.** The standard cells are clustered using a standard partitioning tool to get a 1000x smaller graph consisting of macros and a few thousand standard cell clusters.

[1] Somewhat confusingly, in several places, the Nature paper refers to this problem as "chip floorplanning." Floorplanning is a different problem (see e.g., [6, Chapter 13], [26]).

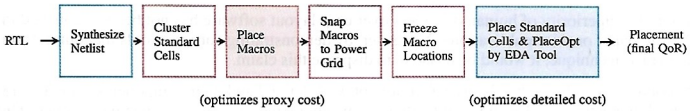

Figure 1: The methodology proposed in the Nature paper [24]. It differs from academic and commercial state-of-the-art by placing the few large components (macros) in a separate step using a simplified "proxy" cost model before placing the myriad small components (standard cells) using a detailed cost model. This old-school "separation of concerns" is intended to make the problem tractable for RL but is not necessary for modern numerical-optimization approaches which concurrently place macros and standard cells using gradient descent.

2. **Coarse Grid.** The placement region is partitioned into a coarse grid (128x128 or less), and each macro is required to be placed at the center of a grid cell, and no two macros are allowed to share a grid cell.

3. **FD Placement.** In an episode, the RL policy first places the macros one by one, and at the end, the standard cell clusters are placed using a simple force-directed placer[2] to obtain rough estimates of wirelength and congestion. These estimates are linearly combined to form the reward for the episode.

To solve the overall placement problem, the authors propose to first obtain macro locations by solving the reduced problem (the "macro placement" step), and then using a commercial tool to place the standard cells ("standard cell" placement or PlaceOpt) (see Figure 1).[3] The methodology is thus reminiscent of earlier placement methods which place large components first, and then optimize the smaller components in contrast to modern methods that determine optimal locations for both large and small objects concurrently.

To improve the runtime and possibly quality of RL, the authors propose to pre-train the policy on a set of training instances (48 hours in their experiments). When presented with a new instance, the policy is fine-tuned on that specific instance (for an additional 6 hours) to produce the final solution.

It is important to note that the objective function for RL is hand-crafted. Therefore,

- RL does not learn an objective that leads to better starting points for the downstream optimization such as PlaceOpt. To do so would require feedback from the downstream steps, but given each run of those steps can take hours, and current deep RL methods need hundreds of thousands of runs (episodes) to learn from, this is impractical.

- Neither does RL learn to produce layouts that are visually pleasing to physical designers or capture their intuition of a good placement (e.g., exhibiting greater symmetry or regularity). To learn human preferences would require a much larger database of human-crafted placements rather than the 20 (unplaced) netlists used in the Nature paper. Such a dataset would be difficult to obtain in practice given the sensitivity of chip designs. To the extent its placements appeal to human intuition, it is due to the grid constraint (and so would be true of solutions obtained by non-RL methods for the reduced problem as well).

3 Evaluating RL for Chip Placement

The main claim of the Nature paper is the ability of the RL agent to outperform human designers and a state-of-the-art academic placement tool (RePlAce [11]) on a set of 5 proprietary TPU blocks (Table 1 of the Nature paper). The experiment supporting this finding has several limitations.

[2] In recent work, the in-house FD placer has been replaced by a pre-existing academic placer [14].

[3] The placement of the (actual) standard cells in this step should not be confused with the previously described force-directed placement of the standard cell *clusters* during an RL episode to calculate the reward. The PlaceOpt step may take hours, as various placement optimizations as well as netlist transformations are performed based on accurate timing and routing models, whereas cluster placement must run in seconds exploiting more crude approximations since it happens in every RL episode.

First, the superiority of human designers over chip layout software has not been established in the Nature paper or prior publications. Moreover, by demonstrating software tools that outperform the novel RL technique, it would be possible to disprove this claim.

Second, the results in the Nature paper are not well-defined and their components cannot be reproduced independently. Design quality attained by a human designer is a subjective baseline. Unlike in Go or Chess, there is no world champion to be beat. Moreover, proprietary designs used as benchmarks cannot be used by other researchers. Hence the question:

> *How does the novel RL method perform on circuit benchmarks commonly used in academic research to evaluate new placement algorithms?*

To perform this evaluation, we use the venerable IBM placement benchmarks [1, 5] (based on diverse ASIC chips designed by IBM for a variety of companies) which have been used to evaluate academic software since 2004. Using previously published results and our own computational experiments, we compare RL to the three major types of placement algorithms (analytical, partitioning-based, and simulated annealing).

Third, the new RL methods are supported by a potentially limited methodology. As noted in Section 2, modern placement tools simultaneously place macros and standard cells [31, 30, 11, 13], but the methods in the Nature paper are based on placing macros first, and then standard cells.

Fourth, the Nature paper compares and combines tools that optimize somewhat different objectives. The RL agent optimizes for wirelength and congestion when placing macros, whereas the academic optimizer RePlAce [11] optimizes only for wirelength. The commercial tool on the other hand uses a complex objective that includes optimizing for worst-case timing path, power, routability etc.[4] This leads to the following question:

> *How does the two-step methodology proposed in the Nature paper compare to a modern mixed-size methodology in terms of an established objective that (1) the corresponding tools can model and optimize explicitly, and (2) is commonly reported in the literature?*

Comparisons against commercial mixed-size placement tools cannot be published due to license restrictions. However, since RePlAce [11] itself is a strong mixed-size placer, we use it as a stand-in for the commercial tool.

3.1 Experimental Setup

The IBM benchmark suite has been used in the literature to evaluate placement tools for more than a decade and provides a longitudinal view on progress in the field. This benchmark suite comprises 18 ASIC designs which integrate 250-800 macros and 12K-200K standard cells. One of the designs (**ibm05**) lacks macros, so we excluded it from the experiment (see Adya and Markov [2] for the detailed benchmark statistics).

For each benchmark circuit, we consider two methodologies and several placement tools.

- **Two-Step**: The methodology from the Nature paper that first clusters the circuit, then places macros, then places smaller circuit components (standard cells),
- **Mixed-Size**: Processing the circuits directly without clustering, while simultaneously placing macros and smaller components (this capability has been routine among academic and industry tools in the span of the last decade).

To make our experiments reproducible, we place standard cells using a leading academic tool rather than proprietary commercial tools with costly licenses. The impact on layout metrics appears minor and does not affect the results of comparisons between the methodologies, given that we use the same academic tool in the mixed-size methodology.

[4] This mismatch in objectives, along with the greater spacing entailed by the coarse grid for the RL, may explain why RePlAce underperformed in timing and routing congestion observed in their experiment: The RePlAce solution may simply be more congested which causes longer path delays. This could be addressed by running RePlAce in a congestion-driven mode, or by "bloating" macros to increase spacing.

Table 1: Comparison of the RL algorithm proposed in the Nature paper with RePlAce on a set of public benchmarks. RePlAce consistently produces better wirelength than RL with five orders of magnitude less compute. The CPU-Seconds for RL algorithm is computed using a common rule of thumb—1 GPU is equivalent to 10 CPUs. Section A explains the details of comparison metrics.

	Wirelength (HPWL)			Congestion (Median)			Total Compute (CPU-Seconds)			
							RL			
Benchmark	RL	RePlAce	Rel. Imp.	RL	RePlAce	Rel. Imp.	Macros	Std-Cells	RePlAce	Ratio
ibm01	3,171,490	2,282,370	38.96%	27.42	15.84	73.11%	2.76E+07	16	36	7.68E+05
ibm02	5,511,850	4,759,440	15.81%	11.71	7.75	51.18%	2.76E+07	28	54	5.12E+05
ibm03	7,999,620	6,435,680	24.3%	15.31	10.15	50.83%	2.76E+07	32	65	4.25E+05
ibm04	8,685,600	7,261,170	19.62%	27.85	16.00	74.07%	2.76E+07	43	70	3.95E+05
ibm05	(no macros)									
ibm06	6,347,590	5,807,790	9.29%	6.41	5.12	25.23%	2.76E+07	46	64	4.32E+05
ibm07	11,770,500	9,856,720	19.42%	18.07	16.70	8.26%	2.76E+07	73	95	2.91E+05
ibm08	13,476,900	11,467,000	17.53%	23.40	14.71	59.10%	2.76E+07	83	112	2.47E+05
ibm09	14,873,500	12,000,600	23.94%	13.79	11.72	17.70%	2.76E+07	90	113	2.45E+05
ibm10	44,078,200	27,428,500	60.70%	17.15	11.26	52.29%	2.76E+07	149	318	8.69E+04
ibm11	21,873,100	16,997,600	28.68%	20.31	13.55	49.87%	2.76E+07	117	167	1.66E+05
ibm12	43,857,000	30,633,300	43.17%	24.25	15.29	58.57%	2.76E+07	130	258	1.07E+05
ibm13	27,892,900	22,013,800	26.71%	25.75	17.39	48.14%	2.76E+07	150	210	1.32E+05
ibm14	45,531,700	34,180,200	33.21%	47.92	34.43	39.19%	2.76E+07	239	406	6.81E+04
ibm15	52,005,600	45,136,100	15.22%	39.91	32.88	21.41%	2.76E+07	369	455	6.08E+04
ibm16	64,208,400	51,604,800	24.42%	31.66	21.64	46.29%	2.76E+07	396	523	5.29E+04
ibm17	81,436,500	63,593,000	28.06%	63.16	45.24	39.59%	2.76E+07	441	658	4.20E+04
ibm18	45,067,400	39,943,300	12.83%	36.48	32.08	13.72%	2.76E+07	449	539	5.13E+04
Average	—	—	**25.99%**	—	—	**42.86%**	—	—	—	—
Geomean	—	—	—	—	—	—	—	—	—	**1.65E+05**

For the two-step methodology from Nature, we use the (code of the) RL implementation described in the Nature paper to place macros. We then fix the macro locations, and use RePlAce to place standard cells. We consider both variants of the RL method presented in the Nature paper:

- **RL-pt ("RL with pre-training")**: For this configuration, we pre-train a policy on 20 TPU blocks (for 48 hours using 200 CPUs and 20 GPUs) and then fine-tune for 6 hours (using 160 CPUs and 16 GPUs) on the specific benchmark.
- **RL**: For this configuration, we do not use any pre-training, but simply train the policy on the specific benchmark for 24 hours (using 160 CPUs and 16 GPUs).

We compare RL against two pre-existing baselines in two different contexts:

- **RePlAce**: Since RePlAce can solve the mixed-size placement problem, we simply use it to simultaneously place both macros and standard cells. Thus RePlAce provides a baseline for a modern placement flow that does not separate the macro and standard cell placement steps.
- **SA**: We consider a version of the simulated annealing baseline presented in the Nature paper that is strengthened with some new moves and provided with the same amount of compute resources as RL (we discuss this in more detail in the next section). Since SA solves the same reduced optimization problem as **RL** and **RL-pt** (see Section 2), it provides an alternative baseline for the atavistic flow proposed in the Nature paper. Just as with the RL algorithms, we use RePlAce to place standard cells after obtaining macro locations from SA.

The objective for all the methods is to minimize wirelength subject to density constraints (that is, no overlap between components), since as noted in the Nature paper, wirelength tends to be well correlated with dynamic power, routing congestion, and timing metrics; but is much faster to compute in the inner loop of an optimizer[5]. In all cases, we use NTUPlace3 [10] to legalize the final placement (that is, to ensure no overlap) and to perform detailed placement. Throughout the design flow, macro orientations were kept fixed. We measure the final pin-to-pin wirelength (specifically, half perimeter

[5] State-of-art commercial tools for mixed-size placement are able to optimize for more detailed timing and congestion metrics while concurrently placing macros and standard cells. Although comparisons with those tools are of great interest, the results cannot be published due to licensing restrictions.

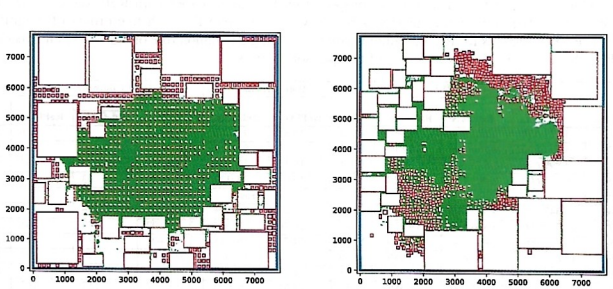

Figure 2: The final placement from **RL** (left) and **RePlAce** (right) for the **ibm10** benchmark. The coarse grid constraint imposed in the RL formulation (in order to have a manageable action space) can lead to unnecessary spreading of small macros which can increase wirelength (and congestion).

wirelength) using WLCalc [9], and median congestion using an academic congestion estimation tool [4] even though none of the tools explicitly optimize for congestion.

We used the default settings of RePlAce for all our experiments except for the cofmax parameter which was lowered to 1.03 (from 1.05) to aid convergence on a few benchmarks. This parameter corresponds to $\max(\mu_k)$ value in ePlace-MS algorithm [21, Section III] (on which RePlAce is based) that controls the decay/growth rate between the wirelength coefficient and the density coefficient.

3.2 Results

A summary of results is shown in Table 1. We find that **RePlAce** produces 26% better wirelength than **RL** while using 5 orders of magnitude less computation. Although congestion was not directly optimized, the considerable difference in wirelength is also reflected in median congestion which is 43% lower.

Pre-training does not help the RL agent much. Although **RL-pt** reduces the run-time (and compute) by 4×, **RL-pt** is still more than 4 orders of magnitude computationally more expensive than **RePlAce**, yet produces 3% worse wirelength than **RL** (Table B1).

With similar computational resources, **SA** does better than **RL** producing a relative improvement of 4.5% in wirelength (Table B2).[6] This indicates that even within the confines of a split macro and standard cell placement flow, there is room for improvement, and that RL as a combinatorial optimizer may have room to improve. We study this aspect in greater detail in the next section.

Finally, we note that although **RL** is not competitive on these benchmarks against **RePlAce**, a tool based on the relatively recent electrostatics-based approach from 2015, it compares favorably—in terms of wirelength—against Capo and FengShui, prior generation tools that are based on the partitioning-driven approach from the early 2000s. **RL** produces an improvement of 2% in wirelength compared to **Capo v9.0** [3] and is only 4% worse than **FengShui v2.6** [18]. However, **RL** uses 3-4 orders of magnitude more compute than these tools (compare Table 7 of [3] with Table 1).

3.3 Discussion

An inherent weakness in the formulation proposed in the Nature paper is illustrated by **ibm10**, the benchmark on which **RL** performs the worst compared to **RePlAce**. Recall that in order to ensure a tractable action space in RL, a grid cell can only be occupied by a single macro, and that the grid is quite coarse (at most 128 x 128). Thus if there are many small macros, they are forced to be spread

[6] This observation correlates with a 3.39% improvement in the proxy cost with a win-rate of >88% for **SA** relative to **RL**.

apart since they are confined to their own respective grid cells (see Figure 2). This can dramatically increase the wirelength (and even congestion due to greater aggregate routing demand).

While spreading due to the grid constraint may help with congestion control around macros in some cases, it is not sufficient, and is usually inferior to more flexible "bloating" techniques:

- **Not sufficient**: If macros are large compared to grid cells, then the grid constraint is not sufficient to enforce distance between two neighboring macros (they may still abut).
- **Not necessary**: If congestion from macros is a problem, a common practice is to "bloat" the macros, that is, to virtually increase their bounding box to create a greater separation from other objects. This can be done in RePlAce without incurring the run-time overhead of a discrete placement on a grid. Bloating becomes more powerful when performed based on congestion maps estimated for a given trial placement.

4 Evaluating RL as a Combinatorial Optimizer

In applying RL to chip placement, the authors of the Nature paper have formulated a specialized combinatorial optimization problem, namely the reduced problem from Section 2 ("coarse-grid clustered-graph placement"), and leveraged the solution of this problem to solve the actual chip placement problem. As shown in the previous section, the authors' implementation is sometimes inferior to modern mixed-size placement tools that avoid clustering and gridding. This gap may be due to weaknesses in the problem formulation or weaknesses in RL-based optimization. The following question helps to clarify this ambiguity:

> *How does DRL compare with other well-studied generic techniques such as simulated annealing in solving an application-specific combinatorial optimization problem?*

In addition to the chip placement context, this question can clarify the broader potential of current RL to solve practical optimization problems: The coarse-grid clustered-graph placement problem is representative of practical problem formulations and instances, and the RL implementation from the Nature paper is representative of the sophistication an experienced machine learning team working for a few years can bring to bear on a new problem. We hope that our experiments help understand the potential of RL in solving practical $\mathcal{NP}$-hard combinatorial optimization problems, and particularly, the capacity of an agent to learn from "self-play" on a set of training instances, an active area of research (see e.g., [7, 23]).

4.1 Experimental Setup

We build on the comparison study with simulated annealing that is presented in the Nature paper. We use the 20 blocks from a recent TPU design which comprised the training set in the Nature paper as our benchmarks since they are most representative of the kinds of blocks the system is designed for.

Algorithms. We compare the following algorithms (using the same codebase as Nature):

- **RL**: This is the RL algorithm described in the Nature paper without pre-training which is run for 6 hours on the specific benchmark.
- **RL-pt-ub ("RL pre-training upper bound")**: This is RL with pre-training on the 20 TPU blocks (for 48 hours using 200 CPUs, 20 GPUs) and then fine-tuned for 6 hours (using 160 CPUs and 16 GPUs) on the specific benchmark. Since the pre-training is on the same benchmark set on which fine-tuning is done, *this setup does not reflect the performance on a new (previously unseen) block, but allows us to obtain an upper bound on what can be learnt offline from self-play across multiple instances.*
- **SA**: The implementation of simulated annealing from the Nature paper with some simple improvements (described in the next section) which is also run for 6 hours on the specific benchmarks. SA uses the same objective that is provided as reward for RL.

Note that using **RL-pt-ub** for comparison has the benefit of sidestepping questions about train and test overlap, and whether training and test examples come from the same distribution, which are both fraught questions in the context of the Nature paper given the presence of only a small number of

Table 2: A comparison of the simulated annealing used in Nature (**SA-Nature**) and ours (**SA**) (see Section 4).

	SA-Nature	**SA**
Actions	`swap, shift, mirror`	`swap, shift, mirror, move, shuffle`
Action Sampling Probability	$[[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]]$	$[[\frac{2}{7}, \frac{2}{7}, \frac{2}{7}, \frac{1}{7}, 0], [\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}]]$
# of Actions / Iteration (n: # of macros)	$2n$	$[2n, 3n, 4n, 5n]$
Maximum Temperature	[1e-5, 3e-5, 5e-5, 7e-5, 1e-4, 2e-4, 5e-4, 1e-3]	[5e-5, 7e-5, 1e-4, 5e-4, 1e-3]
Maximum SA Episode Length (# of iterations)	[5e4, 1e5]	[2e4, 5e4]
# of Random Seeds	5	4
Total SA workers	$1\times1\times1\times8\times2\times5 = 80$	$1 \times 2 \times 4 \times 5 \times 2 \times 4 = 320$

blocks available for the study, and the common presence of near duplicates in physical design.[7] It represents the best bounds that pre-training can buy.

We compare **RL**, **RL-pt-ub**, and **SA** on the value of the objective (that is, the *proxy cost*) achieved by their respective best solution.

4.2 Strengthening the SA Baseline

The authors of the Nature paper describe, in addition to **RL**, a simulated annealing based approach to solving the reduced coarse grid placement problem (**SA-Nature**), and use that as a baseline to evaluate the efficacy of RL on this problem. We strengthen **SA-Nature** as follows (see the detailed **SA** hyperparameters in Table 2):

- **Two new actions.** (i) A **move** action that allows a macro to be placed at any legal location. This is more effective than the shift action in the original action set used by the Nature paper which only moves a macro to a neighboring legal location since it allows macros to cross blockages. (ii) A **shuffle** operation that permutes 4 macros at a time. This generalization of the swap operation is useful where there may not be free room for a single large macro to move to, but it may be "permuted" with 3 smaller macros that are close by.
- **Equal compute as RL.** We use 320 CPUs to run 320 parallel SA runs with various hyperparameters. This matches the compute provided to RL which is 160 CPUs and 16 GPUs[8]. We did not optimize the hyperparameter set of SA to improve performance for this experiment but simply extended the hyperparameter set used in the Nature paper to accommodate the new actions and additional CPUs.
- **Better initialization.** We use a random placement algorithm to generate initial macro placements for SA where it sequentially places macros by randomly sampling legal grid locations on a chip canvas. However, if it is not able to find a legal macro placement, two simple packing methods are tried in sequence to place the macros instead: (i) spiral placement where the macros are sequentially placed around the boundary of the chip canvas in spiral fashion, and (ii) greedy packer where the macros are placed from the bottom-left corner to the top-right corner to minimize the gap between macros. Both packing methods start with the largest macros (which is similar to the order used by the RL agent).

We refer to the resulting simulated annealing implementation as **SA**. A full comparison of the hyperparameter configurations of **SA** and **SA-Nature** is shown in Table 2. We explain the details of all the comparison metrics in Section A. With these changes, we find that **SA** is a stronger baseline than **SA-Nature** with a 90% win-rate at 6 hours and a relative cost improvement of 1.8% at 6 hours (see Table B3 for details).

Our objective with this exercise was to see how a modest engineering effort could improve SA-Nature, but the underlying SA formulation in the Nature paper has a fundamental limitation that we do not address here. Although during annealing, the wirelength is allowed to increase, no action may violate legality (i.e., lead to overlap). This can cause SA to get stuck in certain parts of the solution space. For

[7] The Nature paper does not discuss the question of near-duplicates between the training and test set, and how that may impact the results. Near-duplicates are common in physical design since there is reuse of blocks across different generations of a chip, multiple versions of the same block as a design is refined, and multiple physical variants of the same logical block.

[8] This uses a common rule of thumb—also used in the Nature paper—that 1 GPU is equivalent to 10 CPUs.

Table 3: A comparison of simulated annealing (**SA**) and an *upper bound* on RL with pre-training (**RL-pt-ub**) on 20 blocks used in the Nature paper. **SA** wins on more than two thirds of the benchmarks, and shows a relative improvement in proxy cost of >4% over **RL-pt-ub**. **RL-pt-ub** was unable to find a feasible solution for **block20** (even on a second independent run). Section A explains the details of comparison metrics.

	Proxy Cost			
Benchmark	**RL-pt-ub**	**SA**	**Rel. Imp.**	**Score**
block01	0.04297	0.04289	0.19%	1.0
block02	0.04419	0.04448	-0.65%	0.0
block03	0.04406	0.04393	0.29%	1.0
block04	0.04420	0.04432	-0.27%	0.0
block05	0.05497	0.05573	-1.37%	0.0
block06	0.04618	0.04437	3.92%	1.0
block07	0.03442	0.03072	10.74%	1.0
block08	0.03497	0.03458	1.13%	1.0
block09	0.06639	0.06669	-0.45%	0.0
block10	0.06781	0.06826	-0.66%	0.0
block11	0.05940	0.05930	0.17%	1.0
block12	0.06163	0.06154	0.14%	1.0
block13	0.05088	0.05161	-1.41%	0.0
block14	0.05339	0.05286	1.00%	1.0
block15	0.06468	0.06537	-1.06%	0.0
block16	0.03254	0.02531	22.22%	1.0
block17	0.03218	0.03143	2.33%	1.0
block18	0.10482	0.08133	22.41%	1.0
block19	0.09623	0.07735	19.62%	1.0
block20	Failed	0.07663	N/A	1.0
Average	—	—	**4.13%**	**67.50%**

example, when there is a significant imbalance in macro sizes (e.g., see Figure 2), the larger macros are unlikely to move from their initial location. A better SA formulation may tolerate violations of legality early on in the optimization, to avoid getting stuck in this manner, or do multiple short runs of SA from various initial points. RL does not get stuck in this particular manner, since in each episode, it starts from a fresh canvas, and places macros one by one.

4.3 Results

Our main result is that even if RL is pre-trained for 48 hours on the *same* set of blocks on which it is fine-tuned for a further 6 hours, it is *not* enough to beat SA running for only 6 hours. As Table 3 shows, **SA** has a win-rate of 67.5% against **RL-pt-ub**, and a relative improvement of >4% in the objective. It should be noted that because of the pre-training, the aggregate compute (for all 20 benchmarks) used by **RL-pt-ub** is 1.5× the corresponding compute used by **SA**[9].

By way of ablation, we confirm that pre-training—at least on the same set of examples that are used for fine-tuning—helps. **RL-pt-ub** has a win-rate of 82.5% and a relative improvement of 1.62% in the objective against **RL** (see Table B4).

5 Conclusion

Our work seeks chip placement techniques that deliver best results on public benchmarks and on recent industry designs in terms of design quality metrics and also runtime. It is rare for empirical comparisons between leading techniques to show a clear winner—tradeoffs are more typical. To this end, our comparisons put side by side a novel RL technique against leading established techniques, with several enhancements. Empirical results indicate that these strong baselines reliably attain state of the art performance both on common public benchmarks and industrial designs.

Chip design flows consist of many stages and evaluate a series of increasingly realistic design metrics. Neither the Nature paper nor our study report end metrics at the stage where chip designs can be

[9] We note that if **SA** is run with half the compute—by using only 2 random seeds instead of 4 (see Table 2)—to match only the CPU resources used by **RL-pt-ub** during fine-tuning, it still has a win-rate of 55% and a relative improvement of 3.82% in proxy cost.

manufactured (detailed routing, final timing closure, design rule checks, etc.). We measure design metrics before routing but provide early estimates of routing congestion. These metrics are common in both academia and industry. They generally correlate with later-stage design metrics, except that optimizations that target later-stage metrics directly can spoil early-stage metrics. The RL technique in the Nature paper does not explicitly compute later-stage metrics and does not track such metrics computed by external EDA tools.[10] Therefore, comparisons by early-stage metrics are fair and should be representative of success in practice. Moreover, the large gap observed in our experiments (26% by wirelength and 42% in congestion) leaves little room for trend reversal in terms of later-stage metrics.

Despite the limitations of the reported metrics, the Nature paper mentions that RL contributed to layouts that were sent to a third party for post-placement optimization of a recent design. However, the RL method described in the paper was not used for all the blocks in that design, and since other methods were used for the remaining blocks, that would indicate a benchmarking loss for the method proposed in the Nature paper. Characterizing the set of other methods used, and measuring the relative losses would be an important scientific result.[11] Furthermore, while the main contribution of the Nature paper was in RL, these blocks were post-processed by SA, and none of the detailed metrics used in their main experiment were reported for the ablation or for a comparison with SA. It should be noted that results reported in Table 1 of the Nature paper do not correspond to the blocks sent to the third party, but to five blocks from a previous generation design. A critical contribution to the literature would be to characterize the performance of SA on these 5 blocks, and of all the methods, on the remaining blocks of that previous generation design.

In comparisons on public benchmarks by interconnect length and routing congestion estimates, layout quality attained by the RL methodology from the Nature paper is far below that of the leading academic software RePlAce (it was also used in the Nature paper, but in a different capacity). This loss can be decomposed into components—one that depends on the quality of RL, and the remainder that is due to the clustering-and-gridding methodology. We study the first component by comparing RL to simulated annealing (SA) and find consistent but modest losses. The methodology losses are more substantial and definitely leave room for improvement, although closing the performance and especially the computation cost gap with RePlAce would require a leap.

The Nature paper claims that RL outperforms human chip designers, but our stronger baselines outperform RL. Hence, the work of chip designers used in the Nature paper can not serve as a strong baseline in scientific settings. The consideration of runtime makes this conclusion particularly clear. Software takes minutes to produce high-quality placements of chip designs with hundreds of thousands components. In many cases, a human would take longer to read and understand the circuit, let alone optimize it.

We hope that our insights and our results help improve follow-up research on uses of RL in chip design problems.

References

[1] S Adya, I Markov, and S Chaturvedi. ICCAD04 mixed-size placement benchmarks. `http://vlsicad.eecs.umich.edu/BK/ICCAD04bench/`, 2021. Accessed: 2021-10-21.

[2] Saurabh N Adya and Igor L Markov. Combinatorial techniques for mixed-size placement. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(1):58–90, 2005.

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

[4] Saurabh N Adya, Mehmet Can Yildiz, Igor L Markov, Paul G Villarrubia, Phiroze N Parakh, and Patrick H Madden. Benchmarking for large-scale placement and beyond. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(4):472–487, 2004.

[5] SN Adya, S Chaturvedi, JA Roy, DA Papa, and IL Markov. Unification of partitioning, floorplanning and placement. In *Proc. ICCAD*, pages 550–557, 2004.

[10] Notably, several commercial tools today solve the problem formulation addressed in this paper and the Nature paper and report relevant metrics, but comparisons cannot be reported due to license limitations.

[11] Though without doubt, the raw numbers would be difficult to share due to the obvious IP concerns.

[6] Charles J Alpert, Dinesh P Mehta, and Sachin S Sapatnekar. *Handbook of algorithms for physical design automation.* CRC press, 2008.

[7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[8] Melvin A Breuer. A class of min-cut placement algorithms. In *Proceedings of the 14th Design Automation Conference*, pages 284–290, 1977.

[9] Andrew E Caldwell, Andrew B Kahng, and Igor L Markov. Toward cad-ip reuse: The marco gsrc bookshelf of fundamental cad algorithms. *IEEE Design and Test*, pages 72–81, 2002.

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

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

[12] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[13] Meng-Kai Hsu and Yao-Wen Chang. Unified analytical global placement for large-scale mixed-size circuit designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(9):1366–1378, 2012. doi: 10.1109/TCAD.2012.2193582.

[14] Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, and David Z Pan. Delving into macro placement with reinforcement learning. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–3. IEEE, 2021.

[15] Andrew B. Kahng. Advancing placement. In *ISPD '21: International Symposium on Physical Design, Virtual Event, USA, March 22-24, 2021*, pages 15–22. ACM, 2021.

[16] George Karypis and Vipin Kumar. A hypergraph partitioning package. *Army HPC Research Center, Department of Computer Science & Engineering, University of Minnesota*, 1998.

[17] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[18] Ateen Khatkhate, Chen Li, Ameya R Agnihotri, Mehmet C Yildiz, Satoshi Ono, Cheng-Kok Koh, and Patrick H Madden. Recursive bisection based mixed block placement. In *Proceedings of the 2004 international symposium on Physical design*, pages 84–89, 2004.

[19] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[20] Rongjian Liang, Hua Xiang, Diwesh Pandey, Lakshmi Reddy, Shyam Ramji, Gi-Joon Nam, and Jiang Hu. Drc hotspot prediction at sub-10nm process nodes using customized convolutional network. In *Proceedings of the 2020 International Symposium on Physical Design*, pages 135–142, 2020.

[21] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, et al. eplace-ms: Electrostatics-based placement for mixed-size circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):685–698, 2015.

[22] Igor L Markov, Jin Hu, and Myung-Chul Kim. Progress and challenges in vlsi placement research. *Proceedings of the IEEE*, 103(11):1985–2003, 2015.

[23] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, page 105400, 2021.

[24] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594 (7862):207–212, 2021. ISSN 0028-0836. doi: 10.1038/s41586-021-03544-w.

[25] Gi-Joon Nam and Jingsheng Jason Cong. *Modern circuit placement: best practices and results*. Springer Science & Business Media, 2007.

[26] Ralph HJM Otten. What is a floorplan? In *Proceedings of the 2000 international symposium on Physical design*, pages 201–206, 2000.

[27] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[28] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[29] Inc. Synopsys. What is Design Space Optimization? `https://www.synopsys.com/glossary/what-is-design-space-optimization.html`, 2021. Accessed: 2021-06-22.

[30] Cadence ™. Innovus Implementation System. `https://bit.ly/3Bq9Fpq`, 2021. Accessed: 2021-10-29.

[31] Synopsys ™. Synopsys and Samsung Foundry Collaboration Delivers Optimized Reference Methodology for High-Performance Compute Designs. `https://bit.ly/2ZLwdnw`, 2021. Accessed: 2021-10-29.

[32] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[33] Jurjen Westra and Patrick Groeneveld. Is probabilistic congestion estimation worthwhile? In *Proceedings of the 2005 international workshop on System level interconnect prediction*, pages 99–106, 2005.

[34] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. Routenet: Routability prediction for mixed-size designs using convolutional neural network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

Appendices

A Comparison Metrics

Quality. In our experiments we measure the cost of the best solution produced by each algorithm. We compare the algorithms based on the following metrics:

- **Win rate.** This is computed as follows: The algorithm that obtains the minimum cost solution on a benchmark gets a score of 1 point for that benchmark. If multiple algorithms attain the same (up to 0.1% relative error) minimum cost, we declare a tie and divide the point equally between them. The win-rate of an algorithm is its average point score over the benchmark set. This convenient but coarse aggregate of overall performance is mostly used to tell if one of the algorithms is clearly better or if the algorithms are generally comparable.
- **Relative improvement (Rel. Imp.)** When comparing two algorithms A and B in a head-to-head fashion over a set of benchmarks, we compute the relative cost improvement of A over B on a particular benchmark as $(b - a)/\min(a, b)$ where a and b are the costs of the solutions produced by A and B respectively (on that benchmark). The relative improvement of A over B is the average of this value over the benchmark set.

As summary statistics, win-rate and relative cost improvement have complementary strengths: Win-rate is less influenced by the out-performance of one algorithm over another on one or a few benchmarks than relative cost improvement, but at the same time, may overstate the small differences in the costs of the solutions produced by two algorithms. Therefore, we show both these statistics when summarizing results.

Runtime. Since **RL** constructively produces a solution, it may not produce a valid (feasible) solution for a while. We measure **RL** runtime from the time it produces the first feasible solution or 35 minutes after start, whichever is earlier. We do not need this consideration for **SA** since **SA** always starts with a valid solution and optimizes it. **RL** and **SA** runtimes also exclude the time taken to cluster the standard cells.

B Detailed Results

This section provides more details on the results referenced from the main text.

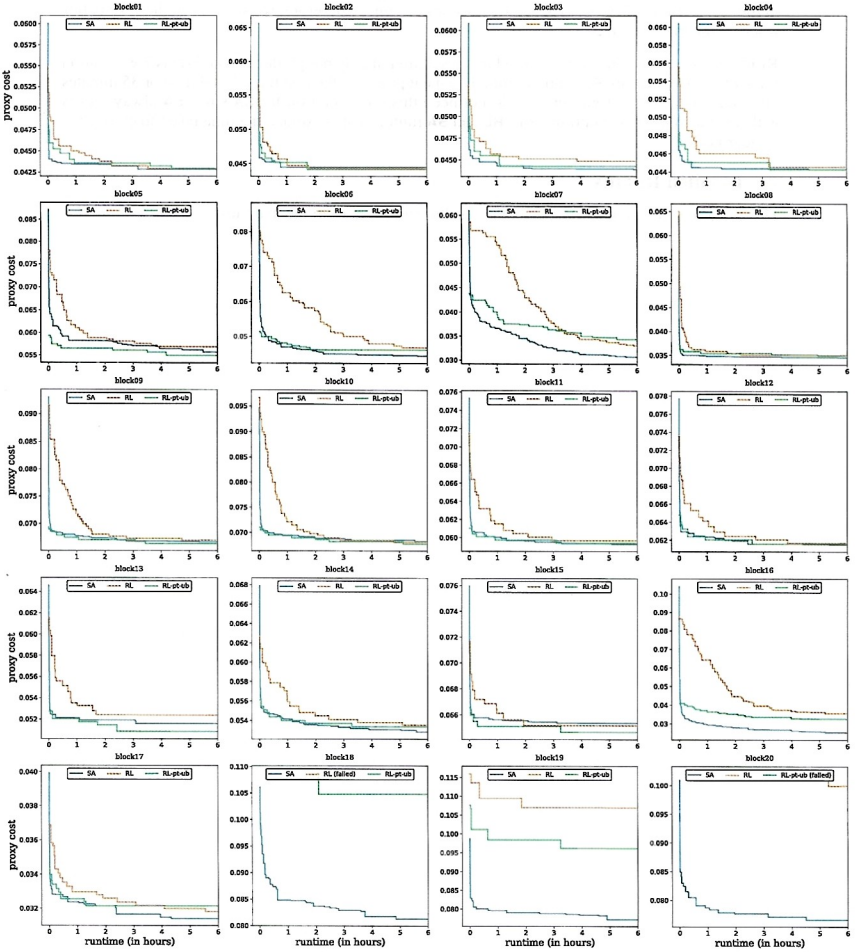

Figure B1: Convergence curves of **SA**, **RL**, and **RL-pt-ub** ("upper bound on RL with pretraining") on a set of 20 blocks from a recent TPU design. Note that the y-axis does not start at 0. **RL** and **RL-pt-ub** failed to produce feasible solutions for block18 and block20 (even with second independent runs), respectively. See Section 4 for details.

Table B1: Detailed comparison of **RL** and **RL-pt** (see Section 3 for details).

	Wirelength (HPWL)			Congestion (Median)		
Benchmark	**RL**	**RL-pt**	**Rel. Imp.**	**RL**	**RL-pt**	**Rel. Imp.**
ibm01	3,171,490	3,786,510	-19.39%	27.42	31.12	-13.50%
ibm02	5,511,850	5,521,780	-0.18%	11.71	11.53	1.61%
ibm03	7,999,620	8,343,440	-4.30%	15.31	17.33	-13.18%
ibm04	8,685,600	8,806,920	-1.40%	27.85	23.71	17.49%
ibm05	(No Macros)					
ibm06	6,347,590	6,454,980	-1.69%	6.41	6.48	-1.10%
ibm07	11,770,500	11,771,200	-0.01%	18.07	21.08	-16.63%
ibm08	13,476,900	14,489,100	-7.51%	23.4,26.18	-11.87%	
ibm09	14,873,500	14,167,200	4.99%	13.79	15.22	-10.34%
ibm10	44,078,200	46,304,200	-5.05%	17.15	19.84	-15.72%
ibm11	21,873,100	22,878,900	-4.60%	20.31	25.96	-27.86%
ibm12	43,857,000	43,577,300	0.64%	24.25	25.66	-5.84%
ibm13	27,892,900	28,282,100	-1.40%	25.75	29.63	-15.03%
ibm14	45,531,700	48,356,400	-6.20%	47.92	50.43	-5.23%
ibm15	52,005,600	51,848,600	0.30%	39.91	42.34	-6.08%
ibm16	64,208,400	70,492,300	-9.79%	31.66	34.72	-9.66%
ibm17	81,436,500	82,082,800	-0.79%	63.16	61.48	2.72%
ibm18	45,067,400	44,187,200	1.99%	36.48	35.45	2.92%
Average	—	—	**-3.20%**	—	—	**-7.49%**

Table B2: Detailed comparison of **RL** and **SA** (see Section 3 for details).

	Wirelength (HPWL)			Congestion (Median)			Proxy Cost		
Benchmark	**RL**	**SA**	**Rel. Imp.**	**RL**	**SA**	**Rel. Imp.**	**RL**	**SA**	**Rel. Imp.**
ibm01	3,171,490	2,585,270	22.68%	27.42	18.90	45.10%	0.17174	0.14848	15.66%
ibm02	5,511,850	5,487,130	0.45%	11.71	9.81	19.44%	0.12057	0.11604	3.90%
ibm03	7,999,620	8,067,560	-0.85%	15.31	18.67	-21.91%	0.15781	0.14969	5.42%
ibm04	8,685,600	8,331,590	4.25%	27.85	23.77	17.16%	0.13475	0.13057	3.20%
ibm05	(No Macros)								
ibm06	6,347,590	6,908,770	-8.84%	6.41	14.12	-120.30%	0.12581	0.13407	-6.57%
ibm07	11,770,500	11,102,600	6.02%	18.07	21.55	-19.25%	0.12993	0.12965	0.22%
ibm08	13,476,900	13,106,500	2.83%	23.4	25.96	-10.95%	0.14611	0.14291	2.24%
ibm09	14,873,500	13,544,500	9.81%	13.79	17.30	-25.42%	0.11578	0.11443	1.18%
ibm10	44,078,200	42,313,700	4.17%	17.15	26.66	-55.47%	0.12632	0.12184	3.68%
ibm11	21,873,100	21,011,600	4.10%	20.31	28.88	-42.21%	0.13562	0.13440	0.91%
ibm12	43,857,000	41,050,000	6.84%	24.25	32.82	-35.34%	0.14700	0.13792	6.58%
ibm13	27,892,900	25,989,300	7.32%	25.75	29.23	-13.51%	0.13974	0.13678	2.16%
ibm14	45,531,700	40,580,100	12.20%	47.92	41.80	14.65%	0.21130	0.19613	7.73%
ibm15	52,005,600	51,006,400	1.96%	39.91	40.09	-0.44%	0.18306	0.18045	1.44%
ibm16	64,208,400	61,454,200	4.48%	31.66	33.92	-7.14%	0.15784	0.14742	7.07%
ibm17	81,436,500	72,040,100	13.04%	63.16	55.78	13.23%	0.23965	0.23019	4.11%
ibm18	45,067,400	44,200,200	1.96%	36.48	35.46	2.90%	0.23925	0.24222	-1.24%
Average	—	—	**5.44%**	—	—	**-14.09%**	—	—	**3.39%**

Table B3: Detailed comparison of **SA-Nature** and **SA** (see Section 4 for details).

	Proxy Cost			
Benchmark	**SA-Nature**	**SA**	**Rel. Imp.**	**Score**
block01	0.04294	0.04289	0.12%	1.0
block02	0.04475	0.04448	0.60%	1.0
block03	0.04420	0.04393	0.61%	1.0
block04	0.04645	0.04432	4.59%	1.0
block05	0.05785	0.05573	3.66%	1.0
block06	0.04336	0.04437	-2.28%	0.0
block07	0.03322	0.03072	7.53%	1.0
block08	0.03471	0.03458	0.37%	1.0
block09	0.06831	0.06669	2.37%	1.0
block10	0.06877	0.06826	0.74%	1.0
block11	0.05943	0.05930	0.22%	1.0
block12	0.06149	0.06154	-0.08%	0.5
block13	0.05193	0.05161	0.62%	1.0
block14	0.05462	0.05286	3.22%	1.0
block15	0.06533	0.06537	-0.06%	0.5
block16	0.02696	0.02531	6.12%	1.0
block17	0.03168	0.03143	0.79%	1.0
block18	0.08279	0.08133	1.76%	1.0
block19	0.07975	0.07735	3.01%	1.0
block20	0.07978	0.07663	3.95%	1.0
Average	—	—	**1.89%**	**90.00%**

Table B4: Detailed comparison of **RL** and **RL-pt-ub** (see Section 4 for details).

	Proxy Cost			
Benchmark	**RL**	**RL-pt-ub**	**Rel. Imp.**	**Score**
block01	0.04298	0.04297	0.03%	0.5
block02	0.04437	0.04419	0.40%	1.0
block03	0.04490	0.04406	1.87%	1.0
block04	0.04461	0.04430	0.69%	1.0
block05	0.05701	0.05497	3.58%	1.0
block06	0.04691	0.04618	1.55%	1.0
block07	0.03308	0.03442	-3.88%	0.0
block08	0.03511	0.03497	0.39%	1.0
block09	0.06699	0.06639	0.89%	1.0
block10	0.06817	0.06781	0.53%	1.0
block11	0.05972	0.05940	0.54%	1.0
block12	0.06172	0.06163	0.14%	1.0
block13	0.05243	0.05088	2.95%	1.0
block14	0.05358	0.05339	0.36%	1.0
block15	0.06522	0.06468	0.83%	1.0
block16	0.03583	0.03254	9.19%	1.0
block17	0.03184	0.03218	-1.07%	0.0
block18	Failed	0.10482	N/A	1.0
block19	0.10710	0.09623	10.15%	1.0
block20	0.10005	Failed	N/A	0.0
Average	—	—	**1.62%**	**82.50%**