GauS: Differentiable Scheduling Optimization via Gaussian Reparameterization

Yaohui Cai
Cornell University
yc2632@cornell.edu

Vesal Bakhtazad
Cornell University
vb354@cornell.edu

Cunxi Yu
University of Maryland, College Park
cunxiyu@umd.edu

Zhiru Zhang
Cornell University
zhiruz@cornell.edu

Abstract

Efficient operator scheduling is a fundamental challenge in software compilation and hardware synthesis. While recent differentiable approaches have sought to replace traditional ones like exact solvers or heuristics with gradient-based search, they typically rely on categorical distributions that fail to capture the ordinal nature of time and suffer from a parameter space that scales poorly. In this paper, we propose a novel differentiable framework, GauS, that models operator scheduling as a stochastic relaxation using Gaussian distributions, which fully utilize modern parallel computing devices like GPUs. By representing schedules as continuous Gaussian variables, we successfully capture the ordinal nature of time and reduce the optimization space by orders of magnitude. Our method is highly flexible to represent various objectives and constraints, which provides the first differentiable formulation for the complex pipelined scheduling problem. We evaluate our method on a range of benchmarks, demonstrating competitive empirical quality–runtime tradeoffs.

1 Introduction

The efficient scheduling of operations onto a finite set of resources, an NP-hard problem known as operator scheduling, is a fundamental challenge across the entire computing stack. In software compilation, it is the critical phase that determines the execution order of instructions to maximize instruction-level parallelism and minimize pipeline stalls Lam [1988]. In hardware synthesis, it is the cornerstone of high-level synthesis (HLS) Cong and Zhang [2006], Ku and De Mitcheli [2002], Radivojevic and Brewer [1996], where it dictates the cycle-by-cycle behavior of custom logic and the allocation of physical functional units. At its core, the problem involves mapping a set of operations to a discrete set of time steps while minimizing a given cost function and honoring a set of constraints.

As we transition into an era of massive-scale deep learning models and heterogeneous architectures, the complexity of scheduling tasks has scaled dramatically from hundreds to tens of thousands of nodes. Modern workloads, such as those targeting the Groq Tensor Streaming Processor (TSP) Abts et al. [2020], require orchestrating thousands of functional units simultaneously to exploit massive data parallelism. Similarly, emerging Processing-in-Memory (PIM) architectures Guo et al. [2025], Seshadri et al. [2017], present unique scheduling challenges where bit-serial execution across large memory arrays requires precise temporal and spatial specification of thousands of operations.

Historically, efforts to solve the operator scheduling problem have fallen into two disparate camps, neither of which fully satisfies the needs of modern large-scale systems Leiserson and Saxe [1991]. On one hand, Integer Linear Programming (ILP) Fan et al. [2005] and Satisfiability Modulo Theory

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

(SMT) Fan et al. [2008] solvers provide optimal solutions but suffer from exponential runtime complexity, making them impractical for realistic workloads Amarú et al. [2015], Soi et al. [2025]. On the other hand, heuristic-based methods, such as list scheduling Parker et al. [1986], Kondratyev et al. [2011] and force-directed scheduling (FDS) Verhaegh et al. [1992], Paulin and Knight [1987], offer the speed required for large-scale graphs but often yield suboptimal results, as they lack a global view of the optimization space and struggle to balance competing constraints like memory footprint and communication overhead.

A promising third path has recently emerged: differentiable combinatorial optimization. which has been widely used in many real-world combinatorial problems Bengio et al. [2021], Zhang et al. [2025], Cai et al. [2025], Lin et al. [2019]. By relaxing discrete scheduling decisions into a continuous space, these methods transform the NP-hard combinatorial search into a smooth, differentiable landscape. This shift provides a holistic view of the global optimization problem and allows the problem to be solved by gradient descent and utilizing the modern ML ecosystem including parallel computing devices like GPUs. While GS-Schedule Liu et al. [2024] as an early attempt in this domain successfully proved the feasibility of differentiable optimization for scheduling using categorical relaxations (i.e., Gumbel-Softmax), it does not fully utilize the parallelism of GPUs and the number of parameters requiring optimization increases linearly with respect to the depth (D) and size ($|V|$) of the graph.

In this paper, we introduce GauS, a differentiable Gaussian reparameterization Scheduling framework. We model the execution step of each operator as a continuous random variable following a Gaussian distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where the mean ($\mu$) represents the expected schedule step and the standard deviation (σ) represents the uncertainty. Under this formulation, the number of parameters requiring optimization reduces to $2|V|$. The probability density function (PDF) of random variables provides a smooth and continuous proxy for discrete placement. Because of the properties of Gaussian distribution, we can represent a wide spectrum of expectations of objectives and constraint violations as differentiable functions of the Gaussian parameters (μ and σ). By minimizing the weighted sum of them, the original discrete combinatorial scheduling problem is essentially transformed to a continuous problem that can be efficiently solved by gradient descent.

GauS is a training-free per-instance optimizer within differentiable combinatorial optimization, rather than a learned scheduling policy. Learned schedulers train reusable policies for related problems such as cluster and job-shop scheduling Mao et al. [2019], Zhang et al. [2020]; such policies can amortize decisions or provide initialization when suitable training data are available. The scheduling-specific contribution is the ordinal location–scale representation; augmented Lagrangian optimization, rounding, and greedy repair are standard components.

Our contributions are: **(i) Novel Representation**: the first use of Gaussian reparameterization in operator scheduling, which naturally encodes temporal proximity and reduces the parameter space from $O(D|V|)$ to $O(|V|)$; **(ii) Differentiable Flexibility**: differentiable expressions for diverse objectives and constraints, enabling the first differentiable formulation of pipelined scheduling Lam [1988], Rau and Glaeser [1981].

Experiments on synthetic and realistic benchmarks show competitive empirical quality–runtime tradeoffs. Compared to previous differentiable method and commercial solvers, GauS achieves equivalent solution quality while one to two orders of magnitude faster

2 Preliminaries

2.1 Scheduling Problem

Scheduling is the process of assigning a set of operations to specific time steps, while satisfying given constraints and optimizing for performance objectives. In this section, we formalize the objectives and constraints for both regular and pipelined scheduling.

2.1.1 Regular Scheduling

A workload is represented as a directed acyclic graph (DAG), $G = (V, E)$, where vertices $V = \{v_1, \ldots, v_n\}$ represent computational operations and edges E denote data dependencies. In a regular scheduling problem, each operator v_i is assigned a discrete start time step $s_i \in \mathbb{N}$

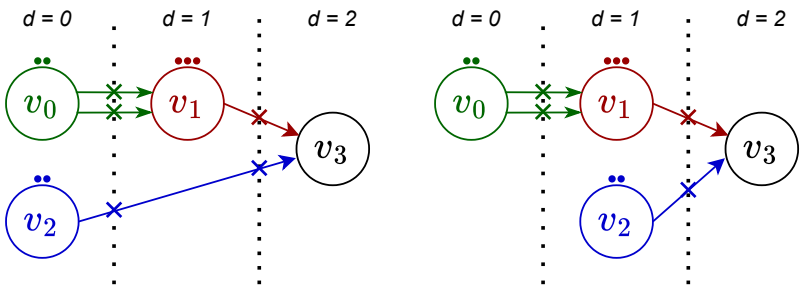

Figure 1: **Impact of optimization objectives on scheduling** — Impact of optimization objectives on scheduling. Nodes v_i (operators) show resource requirements r_i (number of dots) and bitwidths of storage units required b_i (number of arrows). Max depth (D) sets to 3. Left schedule minimizes resources ($\mathcal{L}_{res} = 4, \mathcal{L}_{mem} = 3$); right schedule minimizes memory footprint ($\mathcal{L}_{res} = 5, \mathcal{L}_{mem} = 2$).

Dependency constraint (Dep). For each edge $(v_i, v_j) \in E$, let $g \in \{0, 1\}$ be the required separation: $g = 0$ allows chaining in the same step, whereas $g = 1$ forbids it. For a horizon of D steps, the constraints are

$$\underline{Dep}: \; s_i + g \leq s_j, \qquad \underline{Lat}: \; s_i \in \{0, \ldots, D-1\}. \tag{1}$$

The original Form A comparison uses $g = 0$; the additional categorical extensions of Forms B and C use $g = 1$. More general operation latencies can replace the uniform gap.

Resource usage. Let r_i denote the resource demand of operator v_i. The resource usage at step d, $Res(d)$ is the sum of resources required for all operators active at step d, and the global resource usage is the maximum resource usage over all steps:

$$Res(d) = \sum_{v_i \in V} r_i \cdot \mathbb{I}[d_i = d], \quad \mathcal{L}_{Res} = \max_d Res(d) \tag{2}$$

Here $\mathbb{I}[C]$ is the indicator function, which outputs 1 when condition C is met and 0 otherwise.

Memory Footprint. To carry data across steps, operator v_i utilizes b_i storage units to store its output. Depending on the problem settings, storage units could be registers, SRAM, or DRAM. A storage units is occupied from the step s_i until the step of its latest successor starts. Let $succ(v_i) = \{v_j \mid (v_i, v_j) \in E\}$ denote the set of immediate successors of v_i. The memory usage at step d is the sum of all active storage units which is required by all operators v_i scheduled at or before step d whose outputs are used by at least one operator scheduled after step d. And the global memory footprint is the maximum memory usage over all step[1]:

$$Mem(d) = \sum_{v_i \in V} b_i \cdot \mathbb{I}\left[s_i \leq d < \max_{v_j \in succ(v_i)} s_j\right], \quad \mathcal{L}_{Mem} = \max_d Mem(d) \tag{3}$$

Figure 1 demonstrates how different objectives impact the schedule of the same workload. A resource usage optimized schedule is on the left and a storage unit optimized schedule is on the right.

2.1.2 Modulo Scheduling

For high-throughput applications, *modulo scheduling* Rau and Glaeser [1981], Lam [1988] achieves pipelined execution by starting a new loop iteration (or function invocation) every *initiation interval* (II) steps. Because schedules from consecutive iterations overlap, the lifetimes of an operator and its output are wrapped within the initiation interval, which affects the resource requirements. Consequently, an operator scheduled at any step d such that $d = t + k \cdot \text{II}, k \geq 0$, will demand resources ($MRes$) and occupy storage units ($MMem$) at time step $t, 0 \leq t < \text{II}$:

$$MRes(t) = \sum_k Res(t + k \cdot \text{II}), \quad MMem(t) = \sum_k Mem(t + k \cdot \text{II}) \tag{4}$$

[1]We add an explicit sink node as a successor for all leaf nodes (nodes without successors) to ensure their outputs are also carried toward the end.

Recurrence Constraint (Rec): With loop-carried dependencies between iterations, a back-edge from v_j to v_i with a distance k (iterations) represents the output from v_j is required for a v_i after k iterations. This constraint effectively sets a deadline for operator v_j relatively to the operator v_i.

$$\text{Rec}: \quad s_i + k \cdot \text{II} \geq s_j + g \tag{5}$$

2.2 Scheduling Algorithms

Exact approaches formulate scheduling as ILP Hwang et al. [2002], Gebotys and Elmasry [1993], Fan et al. [2005] or SAT Fan et al. [2008], Zhang et al. [2004]; they guarantee optimality but scale exponentially and rarely converge on realistic graphs.

Heuristics such as ASAP/ALAP, List Scheduling Parker et al. [1986], Kondratyev et al. [2011], and Force-Directed Scheduling (FDS) Verhaegh et al. [1992], Paulin and Knight [1987] trade quality for speed via greedy local decisions; for *modulo scheduling*, where cyclic resource coupling and recurrence constraints often conflict with the target II, ad hoc greedy heuristics dominate Cong and Zhang [2006], Radivojevic and Brewer [1996], Ku and De Mitcheli [2002].

Differentiable methods Bengio et al. [2021], Lin et al. [2019] relax discrete decisions into continuous distributions, enabling first-order optimization, GPU vectorization, and smooth surrogates for non-linear objectives such as memory footprint or learned cost models Chen et al. [2024], Wu et al. [2023]. GS-Schedule Liu et al. [2024] first applied this paradigm to scheduling, modeling each placement as a categorical distribution sampled via the Gumbel-Softmax trick Jang et al. [2016].

3 Method

3.1 Gaussian Reparameterization

The previous differentiable scheduling attempt Liu et al. [2024], represents the placement of an operator v_i as a categorical distribution over D discrete steps, which requires a parameter size of $D \cdot |V|$. Such a formulation suffers from two primary limitations:

Ordinal blindness: Categorical distributions treat steps as independent, nominal labels. From the optimizer's perspective, the relationship between step d and $d+1$ is mathematically identical to that between d and $d+100$. This ignores the fundamental ordinal nature of time; in scheduling, proximity in the timeline should imply proximity in the optimization space.

Insufficient scalability: the optimization space $\mathbb{R}^{D \cdot |V|}$ becomes prohibitively large , as either maximum depth D or the number of operators $|V|$ increases. This limitation becomes increasingly prominent as the needs for large-scale modern workloads scheduling Abts et al. [2020], Soi et al. [2025] increases.

To address these limitations, we propose **Gaussian reparameterization**. In stead of D categorical probabilities, we model the schedule of each operator v_i as an independent Gaussian random variable X_i: $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, where $\mu_i \in \mathbb{R}$ represents the expected scheduling step and $\sigma_i \in \mathbb{R}^+$ represents the standard deviation of the distribution. The probability density function (PDF), $p(x; \mu_i, \sigma_i)$, represents the likelihood of an operator being scheduled at any continuous point x along the timeline. Figure 2 illustrates Gaussian reparameterization. Note that our framework is not limited to Gaussian. We empirically show that other *smooth symmetric unimodal CDFs* can perform similar in E.

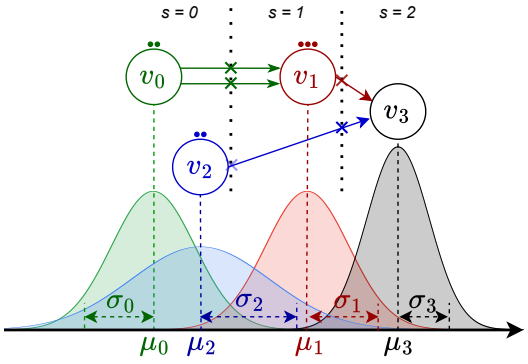

Figure 2: **Illustration of Gaussian reparameterization** — Each operator v_i is parameterized as an independent Gaussian $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ over the scheduling timeline.

By adopting this parameterization, we reduce the number of parameters to be optimized from $D \cdot |V|$ to $2|V|$. For a graph with N operators, we only need to optimize two vectors: mean vector ($\boldsymbol{\mu} \in \mathbb{R}^{|V|}$) that determines the central temporal location of each operator, and standard deviation vector ($\boldsymbol{\sigma} \in \mathbb{R}^{+|V|}$) that controls the degree of relaxation. During initial stage of optimization, $\boldsymbol{\sigma}$ is typically large to encourage encourages exploration of the schedule space; as optimization processes, $\boldsymbol{\sigma}$ usually converges to small values concentrating the probability mass and effectively hardening the schedule toward a deterministic discrete integer.

This Gaussian relaxation also introduces a crucial **temporal proximity** prior. The categorical formulation of GS-Schedule Liu et al. [2024] represents each step as an orthogonal one-hot vector, making all stages mutually equidistant in optimization space even though adjacent steps are temporal neighbors in the schedule space; a small shift in $\boldsymbol{\mu}$ under our Gaussian PDF instead smoothly transports probability mass between neighboring steps, so the gradient pulls each operator toward its temporal neighbors rather than across a flat one-hot simplex. We empirically isolate the temporal proximity effect in Appendix F: after removing the temporal proximity by a controlled d-axis shuffle, EPFL Amarú et al. [2015] benchmarks degrade with an average of 35%.

3.2 Representing Objectives and Constraints

With the Gaussian reparameterization established, the scheduling optimization is transformed into minimizing the expectations of performance objectives and constraint violations with respect to the parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma})$. These surrogates guide optimization; their agreement with decoded costs is conditional, as discussed in Section 3.5. We illustrate this through two important metrics used in regular scheduling; the full derivations, including pipelined scheduling, are provided in Appendix B.

As emphasized in Lin et al. [2019], Liu et al. [2024], Cai et al. [2025], It is important to maintain consistency between optimization and the final sampling process for differentiable combinatorial optimization. In our framework, to extract a discrete schedule from continuous variables, we round X_i to the nearest integer. Therefore, we use probability P_i^d to represent operator v_i being scheduled at step d by integrating the Gaussian PDF over the unit interval centered at d, which is used to align the sampling and optimization:

$$P_i^d = P\left\{\lfloor X_i + 0.5 \rfloor = d\right\} = \Phi\left(\frac{d + 0.5 - \mu_i}{\sigma_i}\right) - \Phi\left(\frac{d - 0.5 - \mu_i}{\sigma_i}\right) \tag{6}$$

Here $\lfloor \cdot \rfloor$ denotes truncation and $\Phi(\cdot)$ is the standard Gaussian cumulative distribution function (CDF). This formulation is efficient and differentiable. Notably, the latency constraint (Lat) is implicitly enforced by bounding the range of d within $\{0, \ldots, D-1\}$ [2]. Given P_i^d, the expectation of global objectives and violations of constraints can be expressed as differentiable functions of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

For *dependency constraint*, we calculate the expected number of violations across all edges. For gap g, an edge violates the constraint when $d_j < d_i + g$. The full expected number of violated edges is

$$\widehat{\mathcal{V}}_{Dep} = \sum_{(i,j)\in E} \sum_{d_i=0}^{D-1} \sum_{\substack{0 \le d_j < D \\ d_j < d_i + g}} P_i^{d_i} P_j^{d_j}. \tag{7}$$

In particular, equality is permitted for $g = 0$ and penalized for $g = 1$. The node distributions factorize, but graph-coupled losses jointly update all parameters. As variances shrink, the product family can approach a point mass at any discrete schedule; the approximation concerns finite-variance joint events.

Memory footprint requires calculating the probability that an operator's output is active at step d. A storage unit for v_i is occupied if the operator has started ($X_i \le d$) but at least one successor v_j has

[2]The lower limit for the first step ($d = 0$) and the upper limit for the last step ($d = D - 1$) are set to $-\infty$ and $+\infty$, respectively.

not yet been scheduled ($X_j > d$). The expected storage unit count at step d is:

$$\widehat{Reg}(d) = \mathbb{E}[Reg(d)] = \sum_{v_i \in V} P\left\{ X_i \leq d < \max_{v_j \in succ(v_i)} X_j \right\} \cdot b_i \tag{8a}$$

$$= \sum_{v_i \in V} \left(\sum_{d_i=0}^{d} P_i^{d_i} \right) \cdot \left(1 - \prod_{v_j \in succ(v_i)} \sum_{d_j=0}^{d} P_j^{d_j} \right) \cdot b_i \tag{8b}$$

The global memory footprint is defined by the max pressure across all steps. Since the `max` function is non-differentiable, we apply LogSumExp to the expected per-step pressure. This smooths the maximum of expected loads; it is not in general the expected maximum.

$$\widehat{\mathcal{L}}_{Reg} \approx \tau \cdot \log \left(\sum_s \exp \left(\frac{\widehat{Reg}(d)}{\tau} \right) \right) \tag{9}$$

3.3 Overall Algorithm

Optimization.

The expected objectives and violations turn scheduling into a continuous constrained optimization problem. Rather than tuning static penalty weights λ_i for $\min \widehat{\mathcal{L}} + \sum_i \lambda_i \widehat{\mathcal{V}}_i$, we use the *augmented Lagrangian method (ALM)* Powell [1969], Hestenes [1969], which adapts λ_i dynamically with violation magnitude:

$$\min_{\boldsymbol{\mu}, \boldsymbol{\sigma}} \mathcal{L}_{total} = \widehat{\mathcal{L}} + \sum_i \left(\lambda_i \widehat{\mathcal{V}}_i + \frac{\rho}{2} \|\widehat{\mathcal{V}}_i\|^2 \right) \tag{10}$$

where ρ is a global penalty parameter. Initially, λ_i is set to a negligible value (e.g., 10^{-6}). After each iteration, λ_i is updated according to the rule $\lambda \leftarrow \lambda_i + \rho \widehat{\mathcal{V}}_i$. This increases the penalty on violations, but does not itself guarantee feasibility.

Sampling. We round the means and check the full schedule. Regular legalization is a feasibility safety net, not a nearest-schedule or objective-optimal projection (Appendix A). It is used at initialization if needed and after 501 consecutive infeasible rounded candidates. A feasible repaired candidate is retained only if it improves the incumbent; otherwise optimization is re-anchored at the incumbent. Adam state persists across this trajectory restart. The output is the best feasible incumbent, not necessarily the final iterate.

Initialization. ASAP and ALAP denote earliest and latest topological timing bounds; the current unit-separation bounds are conservative heuristic windows when chaining is allowed. GauS can warm-start around heuristic solutions. We initialize $\boldsymbol{\mu} = (\boldsymbol{s}^{ASAP} + \boldsymbol{s}^{ALAP})/2$, or a list-scheduling output, and $\boldsymbol{\sigma} = \kappa \cdot (\boldsymbol{s}^{ALAP} - \boldsymbol{s}^{ASAP})$ proportional to each node's scheduling freedom, where κ controls its scale.

To utilize the parallelism of modern GPUs, we vectorize the entire workflow. The complete algorithm for GauS is summarized in Algorithm 1.

Algorithm 1 GauS algorithm

Input: $G = (V, E)$, max depth D, initial mean $\boldsymbol{\mu}_0$
Parameters: Lagrange ratios ρ, std factor κ.
Initialization:
  Compute $\boldsymbol{s}^{ASAP}$ and $\boldsymbol{s}^{ALAP}$
  Set $\boldsymbol{\sigma} = \kappa \cdot (\boldsymbol{s}^{ALAP} - \boldsymbol{s}^{ASAP})$
  Set $\boldsymbol{\mu} = \boldsymbol{\mu}_0$
repeat
  Compute primary objectives $\widehat{\mathcal{L}}(\boldsymbol{\mu}, \boldsymbol{\sigma})$
  Compute expected constraint violations $\widehat{\mathcal{V}}_i(\boldsymbol{\mu}, \boldsymbol{\sigma}), \forall i$
  $\mathcal{L}_{total} \leftarrow \widehat{\mathcal{L}} + \sum_i \left(\lambda_i \widehat{\mathcal{V}}_i + \frac{\rho}{2} \|\widehat{\mathcal{V}}_i\|^2 \right)$
  Update $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow$ optimizer$(\nabla \mathcal{L}_{total})$
  Update $\lambda_i \leftarrow \lambda_i + \rho \widehat{\mathcal{V}}_i, \forall i$
  Discrete schedule $\boldsymbol{s} =$ round$(\boldsymbol{\mu})$
  $\boldsymbol{\mu}, \boldsymbol{\sigma} \leftarrow$ legalize$(\boldsymbol{s})$ **if** $\boldsymbol{s}$ is illegal
until convergence or time limit exceeded
Output: Discrete schedule $\boldsymbol{s}$

3.4 Computational Complexity

For $n = |V|$, $m = |E|$, horizon D, and T iterations, the learned Gaussian parameters and their optimizer state occupy $O(n)$ space. In the current vectorized implementation, node–step and edge–step tensors give $\Theta(TD(n+m))$ time and $\Theta(D(n+m))$ peak working memory, assuming prefix-based dependency and recurrence evaluation. Regular topological repair takes $O(n+m)$. Thus $O(n)$ describes learned state, not total memory.

3.5 Theoretical Analysis

We analyze regular scheduling with a DAG, D steps, unit separation ($g = 1$), independent clipped Gaussian rounding, nonnegative resource weights, and the resource-plus-communication objective of Form A. The analysis does not directly certify the original Form A experiments with chaining ($g = 0$) or modulo scheduling. Let ℓ be the longest path length in edges, $\bar{s}$ the clipped rounded means, S an independently rounded sample, and $\epsilon_i = \Pr(S_i \neq \bar{s}_i)$. Let $\widehat{C}_\tau$ be LogSumExp of expected resource loads plus the expected positive edge spans, and define

$$B = \sum_i [r_i + \alpha(D-1)\deg(i)]\epsilon_i.$$

Theorem 3.1 (Feasibility and decoded quality)**.** *A feasible schedule exists if and only if $\ell \leq D-1$. On a feasible instance, a topological repair after ASAP/ALAP clamping produces a feasible schedule z in $O(|V|+|E|)$ time. If $\bar{s}$ is feasible, $|C(\bar{s}) - \widehat{C}_\tau| \leq B + \tau \log D$ and $C(\bar{s}) \leq \widehat{C}_\tau + B$. In general, the repaired schedule satisfies $C(z) \leq \widehat{C}_\tau + B + J$, where J is the repair charge defined in Appendix H.*

The repair is componentwise minimal above the clamped input, not necessarily the closest or lowest-cost repair. The bounds provide computable, instance-dependent certificates: concentration reduces B, while J accounts for the nodes moved by repair. They are not a global approximation guarantee; the objective is NP-hard even with two steps and no edges. Appendix H gives proofs, a randomized-decoding certificate, and conditional fixed-multiplier convergence results. These rates do not automatically apply to the implemented Adam/restart loop.

4 Evaluation

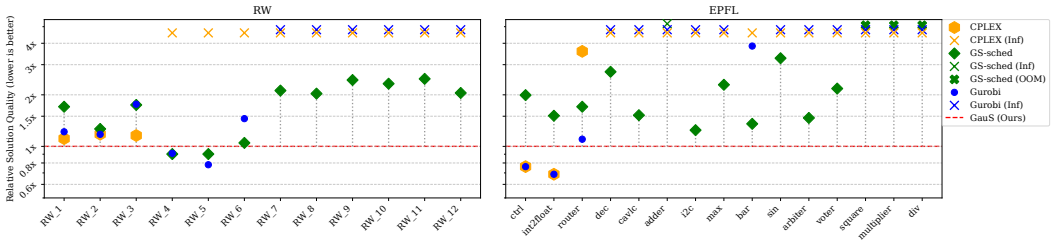

Figure 3: **Formulation A:** *Latency constrained, resource and communication optimization* — Problem instance size ($|V|$) increases from left to right. A 15-minute time limit is applied to all methods. Relative solution quality is represented as the ratio between the methods' solution quality and GauS, where lower values correspond to higher solution quality. Light crosses (labeled as inf) denote no feasible solution or quality exceeding $4\times$ GauS performance; filled crosses indicate GS-Schedule exceeded available CUDA memory (OOM). On shared feasible instances, GauS achieves a **71.8%** geometric mean improvement over GS-Schedule.

The GauS source code is available at `https://github.com/cornell-zhang/GauS`.

4.1 Quality Comparison

Formulation A: The quality comparison between GauS and traditional combinatorial solvers (CPLEX, Gurobi) as well as GS-Schedule is illustrated in Figure 3. The benchmarks are ordered by operator count ($|V|$) from left to right.

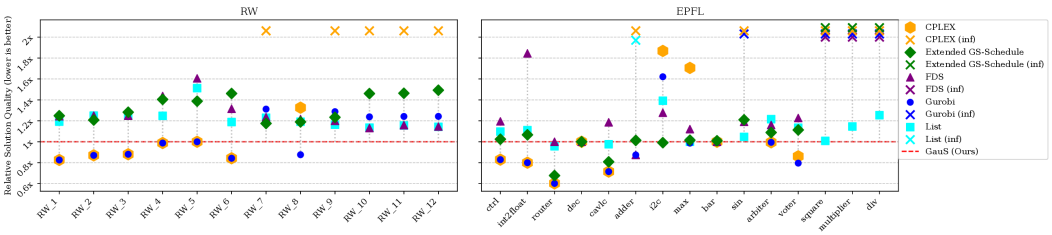

Figure 4: **Formulation B:** *Latency constrained, memory footprint optimization* — A 15-minute time limit is applied to all methods. Crosses labeled "inf denote no feasible solution or quality exceeding $4\times$ GauS performance. Green diamonds and crosses show the extended GS-Schedule baseline.

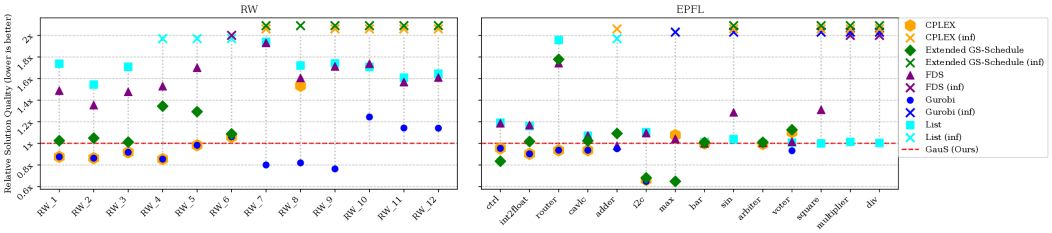

Figure 5: **Formulation C:** *Latency, resources, and recurrence-constrained, memory footprint modulo optimization* — A 15-minute time limit is applied to all methods. Crosses labeled "inf denote no feasible solution or quality exceeding $4\times$ GauS performance. Green diamonds and crosses show the extended GS-Schedule baseline. dec in EPFL has no feasible solution, thus omitted from the plot.

For smaller benchmarks, traditional ILP solvers often find optimal or near-optimal solutions. As the problem scale increases, GauS consistently produces high-quality schedules within the 15-minute time limit. In contrast, traditional solvers marked with light crosses fail to produce a feasible solution or result in a schedule quality exceeding $4\times$ the performance threshold relative to GauS.

While GS-Schedule is able to produce solutions on most benchmarks, the solution quality is about 30% to $3\times$ worse compared to GauS with a geometric mean of 71.8%. This demonstrates the superior efficacy of GauS. The memory efficiency reflects the other advantage of Gaussian reparameterization, parameter efficiency. As shown in the EPFL benchmarks, the GS-Schedule baseline frequently exceeds available CUDA memory (OOM) on larger graphs (represented by filled cross markers), while GauS is able to produce solutions for large graphs. The $O(|V|D)$ categorical trainable state is structural, but OOM depends on the implementation and device; it is not inherent to categorical parameterization. GauS uses $O(|V|)$ learned state, while its vectorized working tensors are larger (Section 3.4).

Formulation B and C: To demonstrate the versatility of GauS, we evaluate its performance on two advanced scheduling formulations: *latency-constrained, memory footprint optimization* and *latency, resource, and recurrence-constrained modulo optimization*.

We extend GS-Schedule's predecessor-conditioned straight-through Gumbel sampler with the liveness objective for Form B and with recurrence and modulo-resource penalties for Form C. These extensions use gap $g = 1$ and do not use GauS initialization, legalization, or post-processing. On RW1–RW12, the extended baseline produced exactly feasible schedules in 36/36 Form B runs and 18/36 Form C runs. On shared-feasible runs, its geometric-mean cost relative to GauS was 1.355 for Form B and 1.128 for Form C. Figures 4 and 5 show the extended baseline alongside the other methods; these findings apply to the tested extensions and do not establish a general limitation of categorical models.

While exact solvers like CPLEX and Gurobi achieve competitive results on smaller graphs, their performance degrades as problem scale increases, frequently exceeding the 15-minute time limit or failing to produce any feasible solution (indicated by light crosses). In contrast, GauS maintains a robust and stable performance profile, consistently matching or exceeding the quality of standard

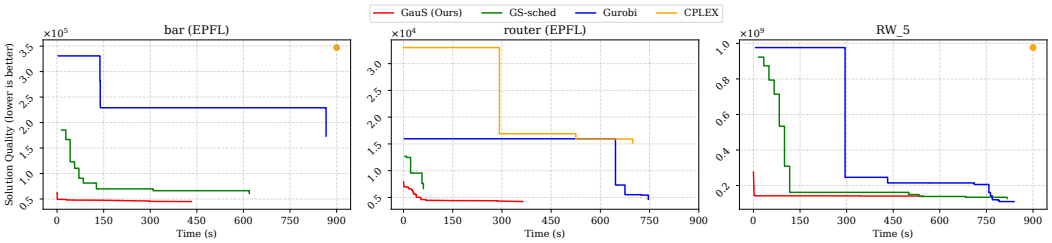

Figure 6: **Quality-Speed tradeoff representative benchmarks** – From left to right: RW_5, router, and bar on formulation A. Lines terminating before the 15-minute indicate no further improvement is made.

heuristics such as List Scheduling and Force-Directed Scheduling (FDS). On most benchmarks, these heuristics often result in 20% to 60% higher memory footprint than our method.

The performance gap narrows in large graphs in formulation C on EPFL. As the number of augmented back-edges is proportional to the graph size, feasible search space is significantly restricted, especially for large graphs. decreasing the overall degrees of freedom, making the margin for potential improvement becomes inherently smaller.

4.2 Tradeoff Analysis

To evaluate optimization efficiency, we analyze the anytime results (solution quality vs. runtime) for GauS against baselines. Figure 6 illustrates tradeoff for a subset of problems in Formulation A, with comprehensive results for all benchmarks provided in Appendix I. On the illustrated instances GauS reaches competitive solutions quickly; runtimes combine GPU and CPU implementations and do not isolate hardware from algorithmic effects. When both methods eventually converge to similar objectives, GauS achieves equivalent solution quality one to two orders of magnitude faster. These observations describe the measured quality–runtime tradeoffs, not a proof of global Pareto optimality.

A paired GauS CPU–GPU comparison in Appendix G.1 shows a workload-dependent runtime crossover: the single-thread CPU is faster on RW1–RW3, while the A100 is faster on RW4–RW6. GPU memory and utilization observations are discussed in Appendix G.

5 Conclusion

In this work, we presented GauS, a scalable differentiable framework for operator scheduling designed to overcome the bottlenecks of traditional combinatorial solvers and categorical differentiable models. By leveraging a Gaussian reparameterization, GauS achieves $O(|V|)$ parameter scalability, enabling the optimization of massive-scale hardware designs with tens of thousands of nodes that were previously considered intractable. As the first differentiable method to support pipelined scheduling, GauS offers a unique level of flexibility. Our experimental results across realistic and synthetic workloads demonstrate that GauS delivers the state-of-the-art convergence speeds and high-quality solutions, making it as a robust tool for modern scheduling problem.

GauS can converge to suboptimal solutions, and its factorized distribution approximates coupled joint events at finite variance. Factorization need not capture correlations in the joint distribution. The theoretical results concern regular Form A under explicit assumptions; neither global optimality nor convergence of the implemented Adam/restart procedure is guaranteed.

Acknowledgments

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, NSF Award #2403135, and a research gift from Amazon.

References

Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, et al. Think fast: A tensor streaming processor (tsp) for accelerating deep learning workloads. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 145–158. IEEE, 2020.

Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015. URL `https://si2.epfl.ch/~demichel/publications/archive/2015/IWLS15.pdf`.

Amir Beck. *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with Python and MATLAB*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition, 2023. doi: 10.1137/1.9781611977622.

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.

Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, Oxford, 2013. doi: 10.1093/acprof:oso/9780199535255.001.0001.

Yaohui Cai, Kaixin Yang, Chenhui Deng, Cunxi Yu, and Zhiru Zhang. Smoothe: Differentiable e-graph extraction. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 1020–1034, 2025.

Chen Chen, Guangyu Hu, Dongsheng Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

Jason Cong and Zhiru Zhang. An efficient and versatile scheduling algorithm based on sdc formulation. In *Proceedings of the 43rd annual Design Automation Conference*, pages 433–438, 2006.

Kevin Fan, Manjunath Kudlur, Hyunchul Park, and Scott Mahlke. Cost sensitive modulo scheduling in a loop accelerator synthesis system. In *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)*, pages 12–pp. IEEE, 2005.

Kevin Fan, Hyun hul Park, Manjunath Kudlur, and S ott Mahlke. Modulo scheduling for highly customized datapaths to increase hardware reusability. In *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, pages 124–133, 2008.

C.H. Gebotys and M.I. Elmasry. Global optimization approach for architectural synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(9):1266–1278, 1993. doi: 10.1109/43.240074.

Deyuan Guo, Mohammadhosein Gholamrezaei, Matthew Hofmann, Ashish Venkat, Zhiru Zhang, and Kevin Skadron. Pimsynth: A unified compiler framework for bit-serial processing-in-memory architectures. *IEEE Computer Architecture Letters*, 2025.

Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

C-T Hwang, J-H Lee, and Y-C Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10 (4):464–475, 2002.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the polyak–łojasiewicz condition. In Paolo Frasconi, Niels Landwehr, Giuseppe Manco, and Jilles Vreeken, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 9851 of *Lecture Notes in Computer Science*, pages 795–811. Springer, Cham, 2016. doi: 10.1007/978-3-319-46128-1_50.

Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, Boston, MA, 1972. doi: 10.1007/978-1-4684-2001-2_9.

Alex Kondratyev, Luciano Lavagno, Mike Meyer, and Yosinori Watanabe. Realistic performance-constrained pipelining in high-level synthesis. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

David C Ku and G De Mitcheli. Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):696–718, 2002.

Monica Lam. Software pipelining: An effective scheduling technique for vliw machines. In *Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 318–328, 1988.

Charles E Leiserson and James B Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

Mingju Liu, Yingjie Li, Jiaqi Yin, Zhiru Zhang, and Cunxi Yu. Differentiable combinatorial scheduling at scale. In *Proceedings of the 41st International Conference on Machine Learning*, pages 31464–31476, 2024.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the 2019 ACM SIGCOMM Conference*, 2019. URL `https://web.mit.edu/decima/content/sigcomm-2019.pdf`.

Colin McDiarmid. On the method of bounded differences. In J. Siemons, editor, *Surveys in Combinatorics, 1989*, volume 141 of *London Mathematical Society Lecture Note Series*, pages 148–188. Cambridge University Press, 1989. doi: 10.1017/CBO9781107359949.008.

Alice C Parker, Jorge Pizarro, and Mitch Mlinar. Maha: A program for datapath synthesis. In *23rd ACM/IEEE Design Automation Conference*, pages 461–466. IEEE, 1986.

P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87, page 195–202, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0818607815. doi: 10.1145/37888.37918. URL `https://doi.org/10.1145/37888.37918`.

Boris T. Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, 1963. doi: 10.1016/0041-5553(63)90382-3.

Michael JD Powell. A method for nonlinear constraints in minimization problems. *Optimization*, pages 283–298, 1969.

Ivan Radivojevic and Forrest Brewer. A new symbolic technique for control-dependent scheduling. *IEEE transactions on computer-aided design of integrated circuits and systems*, 15(1):45–57, 1996.

B Ramakrishna Rau and Christopher D Glaeser. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. *ACM SIGMICRO Newsletter*, 12(4):183–198, 1981.

Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1998. ISBN 9780471982326.

Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 273–287, 2017.

Rupanshu Soi, Rohan Yadav, Fredrik Kjolstad, Alex Aiken, Maryam Mehri Dehnavi, Michael Garland, and Michael Bauer. Optimal software pipelining and warp specialization for tensor core gpus. *arXiv preprint arXiv:2512.18134*, 2025.

Verhaegh, Lippens, Aarts, Korst, van der Werf, and van Meerbergen. Efficiency improvements for force-directed scheduling. In *1992 IEEE/ACM International Conference on Computer-Aided Design*, pages 286–291, 1992. doi: 10.1109/ICCAD.1992.279359.

Nan Wu, Yingjie Li, Cong Hao, Steve Dai, Cunxi Yu, and Yuan Xie. Gamora: Graph learning based symbolic reasoning for large-scale boolean networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL `https://papers.nips.cc/paper/2020/hash/11958dfee29b6709f48a9ba0387a2431-Abstract.html`.

Hantao Zhang, Dapeng Li, and Haiou Shen. A sat based scheduler for tournament schedules. In *SAT*, 2004.

Niansong Zhang, Anthony Agnesina, Noor Shbat, Yuval Leader, Zhiru Zhang, and Haoxing Ren. Cypress: Vlsi-inspired pcb placement with gpu acceleration. In *Proceedings of the 2025 International Symposium on Physical Design*, pages 31–41, 2025.

Yanqing Zhang, Haoxing Ren, Akshay Sridharan, and Brucek Khailany. Gatspi: Gpu accelerated gate-level simulation for power improvement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1231–1236, 2022.

A Legalization Heuristic

While the Gaussian stochastic relaxation encourages valid schedules through dependency penalties, rounding learned means μ_i to discrete integers s_i can occasionally result in causality violations. Therefore, we use some heuristics to legalize any solution that violates the constraint. Because of the recurrence constraint in modulo scheduling. we use two different legalization heuristic for regular and modulo scheduling problem.

A.1 Regular Scheduling Legalization Heuristic

The pseudocode below uses unit separation, a conservative choice for chaining when the corresponding unit-separation bounds are feasible. Appendix H analyzes a gap-aware variant; it is not a claim that the current implementation uses gap-aware bounds. This procedure processes the graph in topological order, shifting nodes forward in time only when necessary to satisfy dependency latencies while remaining within the pre-defined ASAP and ALAP bounds. By performing this one-pass refinement, we try to legalize the discrete output without significantly altering the global optimization results. The algorithm is summarized in Algorithm 2

Algorithm 2 Regular Schedule Legalization

Input: $G = (V, E)$, rounded schedule s.
Compute $\boldsymbol{s}^{ASAP}$ and $\boldsymbol{s}^{ALAP}$
$\boldsymbol{s}_{new} \leftarrow \text{clamp}(\boldsymbol{s}, \text{min} = \boldsymbol{s}^{ASAP}, \text{max} = \boldsymbol{s}^{ALAP})$
for each node v in topological_sort(G) **do**
  $preds \leftarrow \{v_j \mid (v_j, v) \in E\}$
  if $|preds| > 0$ **then**
    $t_{req} \leftarrow \max(\boldsymbol{s}_{new}[Preds]) + 1$
    $s_{new}[v] \leftarrow \max(\boldsymbol{s}_{new}[v], t_{req})$
  end if
end for
Output: Feasible discrete schedule s_{new}.

A.2 Modulo Scheduling Legalization Heuristic

Modulo scheduling introduces cyclic constraints due to back-edges. To resolve these, we employ a simple fixed-point relaxation algorithm. Unlike standard topological legalization, this iterative approach repeatedly passes through the graph to resolve back-edge requirements alongside forward dependencies. The process continues until the schedule stabilizes or a maximum iteration threshold (equal to the number of nodes) is reached, signaling a potential recurrence violation or depth overflow. This tries to find a feasible schedule for pipelined execution while not significantly changing the original input. The algorithm is summarized in Algorithm 3.

A.3 Empirical Behavior of Regular Legalization

Without runtime legalization, GauS found an exact feasible incumbent in 15/15 Form A runs. The runtime mechanism includes both candidate repair and re-anchoring at a saved incumbent.

B Derivations

We finish the derivation of expected objectives and violations of constraints in this section. After defining communication overhead, we will compute its expectation and expected resource usage under regular scheduling before deriving the metrics essential to modulo scheduling

Algorithm 3 Modulo Schedule Legalization

Input: $G = (V, E)$, back-edge set E_B, max depth D, initiation interval II, rounded schedule $\boldsymbol{s}$.
$\boldsymbol{s}_{new} \leftarrow \boldsymbol{s}$
$nodes \leftarrow$ topological_sort(G)
for $i = 0$ **to** $|V| - 1$ **do**
  $changed \leftarrow$ False
  for each v **in** $nodes$ **do**
    $t_{min} \leftarrow \max(\{\boldsymbol{s}_{new}[u] \mid u \in preds(v)\} \cup \{0\}) + 1$
    $t_{back} \leftarrow \max(\{\boldsymbol{s}_{new}[v_j] - k \cdot \text{II} + 1 \mid (v, v_j, k) \in E_B\} \cup \{0\})$
    $t_{req} \leftarrow \max(t_{min}, t_{back})$
    if $t_{req} > \boldsymbol{s}_{new}[v]$ **then**
      $\boldsymbol{s}_{new}[v] \leftarrow t_{req}, \quad changed \leftarrow$ True
    end if
  end for
  if not changed **then**
    break
  end if
end for
Output: s_{new}.

B.1 Communication overhead

In communication-bottlenecked workloads Zhang et al. [2022], we minimize the total edge length across all dependencies:

$$\mathcal{L}_{Com} = \sum_{(v_i, v_j) \in E} s_j - s_i \tag{11}$$

The following masked surrogate retains nonnegative spans only; it is not the full signed expectation for infeasible samples:

$$\widehat{\mathcal{L}}_{Com} = \sum_{(v_i, v_j) \in E} \sum_{d_i=0}^{D-1} \sum_{d_j=d_i}^{D-1} P_i^{d_i} P_j^{d_j} \cdot (d_j - d_i) \tag{12}$$

B.2 Expected Resource Usage

To represent the expected resource usage at step d, we reuse the probability that an operator v_i with resource demand w_i is active at that step defined in Equation 3.2. The expected resource usage is the sum of individual usage:

$$\widehat{Res}(d) = \mathbb{E}[Res(d)] = \sum_{v_i \in V} w_i \cdot P_i^d \tag{13}$$

To optimize for peak resource utilization, we apply the `LogSumExp` operator across all steps $d \in \{0, \ldots, D-1\}$. This is similar to the computation of global memory footprint in 3.2:

$$\widehat{\mathcal{L}}_{Res} \approx \tau \cdot \log \left(\sum_d \exp \left(\frac{\widehat{Res}(d)}{\tau} \right) \right) \tag{14}$$

We define the violations as the number of exceeded resources used summed across all steps. Then the expected violations can be computed as:

$$\widehat{\mathcal{V}}_{Res} = \sum_d \text{ReLU} \left(\widehat{Res}(d) - R \right) \tag{15}$$

where R is a given resources limit, and $\text{ReLU}(\cdot)$ ensures only exceeding resources are counted toward violations.

B.3 Modulo Metrics

In a pipeline, an operator scheduled at step d_i consumes resources at steps $d_i \mod II$. The probability that operator v_i occupies a resource at time $t \in \{0, \ldots, II-1\}$ within the modulo reservation table is the sum of its probabilities across all linear steps that map to t:

$$P\left\{\lfloor X_i + 0.5 \rfloor = t\right\} = \sum_{k=0}^{\lceil D/\text{II} \rceil} P_i^{t+k \cdot II} \tag{16}$$

where $P_i^{t+k \cdot \text{II}}$ represents the probability of scheduling operator v_i to step $d = t + k \cdot \text{II}$ is defined in Equation 6.

Modulo Resource Usage. Similar to the derivation in 3.2, at step $0 \leq t < \text{II}$, the expected modulo resource usage $\widehat{MRes}(t)$:

$$\widehat{MRes}(t) = \sum_{v_i \in V} w_i \cdot P\left\{\lfloor X_i + 0.5 \rfloor = t\right\} \tag{17}$$

Modulo Memory Footprint. Memory footprint in a pipeline is similarly wrapped. A memory unit carrying the output of v_i contributes at step t for each step $d = t + k \cdot \text{II}$ with $k \geq 0$ at which it is active. Using the linear probability $P(X_i \leq s < \max_{v_j \in succ(v_i)} X_j)$ defined in Equation 8a, we can compute the modulo expectation:

$$\widehat{MReg}(t) = \sum_{v_i \in V} b_i \cdot \sum_{k=0}^{\lceil D/\text{II} \rceil} P\left\{d_i \leq t + k \cdot \text{II} < \max_{v_j \in succ(i)} d_j\right\} \tag{18}$$

The expected maximum modulo resource usage and peak memory footprint can be computed using `LogSumExp` similar to Equation 9. The expected violations can be computed using ReLU similar to Equation 15.

Recurrence Violations. For modulo scheduling, a back-edge from producer v_j to consumer v_i at iteration distance k imposes $s_i + k \cdot \text{II} \geq s_j + g$. Let the set of these dependencies be defined as $(v_j, v_i, k) \in E_B$. Similar to the computation of expected dependency violations in Equation 7, the expected recurrence violation $\widehat{\mathcal{V}}_{rec}$ is formulated by summing the probabilities of all invalid steps assignments (d_i, d_j) that fail to satisfy the recurrence constraint:

$$\widehat{\mathcal{V}}_{Rec} = \mathbb{E}[\mathcal{V}_{Rec}] = \sum_{(v_j, v_i, k) \in E_B} \sum_{d_i=0}^{D-1} \sum_{\substack{0 \leq d_j < D \\ d_j > d_i + k \cdot \text{II} - g}} P_i^{d_i} P_j^{d_j} \tag{19}$$

C Detailed Evaluation Setup

C.1 Experiment Setup

Environment Settings. All the experiments are performed on a Linux server equipped two NVIDIA A100 GPUs with 80 GB memory and two AMD EPYC 9124 CPUs (2×16 cores) running at 3.7 GHz, 1.5 TB of RAM. For softwares, we use PyTorch 2.5.1 with CUDA 12.1, Gurobi 13.0, and CPLEX 22.1.1.

Hyper-parameter Settings. We use a uniform set of hyperparameters across all problem instances and formulations. We use the Adam optimizer with a learning rate of 10^{-2}. The penalty parameter ρ, used to enforce constraints, is set to 10^{-4}; the `LogSumExp` temperature parameter τ is set to 10^{-2}; the initialization freedom parameter κ is set to $\frac{1}{6}$.

C.2 Dataset Details

For the random workload (RW) evaluations, we utilize a subset of the datasets released by GS-Schedule Liu et al. [2024], though our selected subset is substantially larger than the one featured in the original study to better evaluate scalability.

Table 1: Summary of Graph Statistics (Sorted by Nodes)

EPFL			Random Workloads (RW)		
Name	Number of Nodes ($\|V\|$)	Depth (D)	Name	Number of Nodes ($\|V\|$)	Depth (D)
ctrl	182	11	RW_1	929	16
int2float	271	17	RW_2	941	16
dec	312	4	RW_3	949	15
router	344	55	RW_4	4713	18
cavlc	703	17	RW_5	4716	21
adder	1276	256	RW_6	4741	22
i2c	1504	21	RW_7	8058	9
max	3377	288	RW_8	8192	12
bar	3471	13	RW_9	8193	9
sin	5440	226	RW_10	9396	22
arbiter	12095	88	RW_11	9432	21
voter	14759	71	RW_12	9447	22
square	18550	251			
multiplier	27190	275			
div	57375	4373			

D Hyperparameter Sensitivity

GauS exposes three core hyperparameters that we hold fixed across every benchmark in the main paper: the initialization freedom κ, which scales each node's standard deviation σ_i to a fraction κ of its $[\text{ASAP}, \text{ALAP}]$ slack; the `LogSumExp` temperature τ that smooths the resource peak; and the Adam learning rate. We sweep each in turn while holding the other two at their defaults ($\kappa{=}1/6$, $\tau{=}10^{-2}$, lr$=10^{-2}$) on five EPFL benchmarks (`dec`, `cavlc`, `bar`, `arbiter`, `multiplier`) covering three orders of magnitude in graph size, with three seeds per cell, yielding $254/255$ feasible runs within the 900 s time limit.

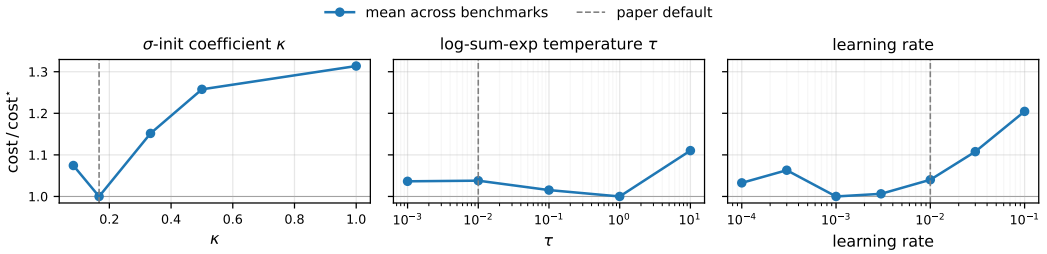

Figure 7: Sensitivity of GauS to its three core hyperparameters under Formulation A. Each panel sweeps one HP while the other two are held at the main-paper defaults ($\kappa{=}1/6$, $\tau{=}10^{-2}$, lr$=10^{-2}$). Per swept value we first compute, for each of five EPFL benchmarks (`dec`, `cavlc`, `bar`, `arbiter`, `multiplier`; 3 seeds each), the ratio of its mean cost to its own best across the swept range, and average those ratios across benchmarks. The curves are then re-normalized by their minimum so that $\text{cost}/\text{cost}^\star = 1$ at the consensus-best value. The dashed line marks the value used in the main paper: it coincides with the consensus best for κ, and lies within $1.04\times$ of it for τ and lr.

The default $\kappa = 1/6$ follows directly from the 3σ rule for the Gaussian. With $\sigma_i = \kappa\,(s_i^{\text{ALAP}} - s_i^{\text{ASAP}})$, taking $\kappa = 1/6$ places $\pm 3\sigma$ exactly at the boundary of the feasible $[\text{ASAP}, \text{ALAP}]$ window for each node, so the initial distribution covers $\approx$99.7% of the feasible steps without leaking probability mass outside. Smaller κ under-explores; larger κ wastes mass on infeasible steps, which Figure 7 (left) confirms causes a measurable cost penalty — particularly on shallow graphs (e.g. `dec`, depth ≈ 4) where the slack window is narrow and $\kappa{=}1$ pushes $\approx$30% of the initial mass outside it.

The other two hyperparameters are far less sensitive. The temperature τ keeps the consensus-normalized cost ratio under $1.04\times$ over $\tau \in [10^{-3}, 10^{0}]$ and only deteriorates at $\tau{=}10$, where the

soft-max becomes nearly flat and provides little gradient signal about the actual peak resource step. The learning rate exhibits a shallow basin from 10^{-3} to 10^{-2} within $1.04\times$ of the consensus best, and degrades only at the extremes ($1.20\times$ at lr=10^{-1}). The chosen lr=10^{-2} trades a small loss in average objective for faster convergence on the larger benchmarks, where lower learning rates do not finish within the 100 000-iteration cap. We note that this is a one-at-a-time study and HP interactions are not isolated; nonetheless the absence of feasibility cliffs across all 255 runs supports the claim that GauS is robust to the choice of HPs around the defaults.

E Distribution-Family Ablation

The Gaussian choice in GauS is motivated by unimodality, smooth temporal-proximity gradients, and a differentiable CDF — none of which are uniquely Gaussian. The forward pass only ever invokes the CDF, and no derivation in this paper requires Gaussianity. To test whether the framework's success is specific to the Gaussian or general to *smooth symmetric unimodal CDFs*, we replace $\Phi(\cdot)$ in Equation 6 with the logistic, Laplace, or Cauchy CDF, calibrating the scale parameter so that the same $\kappa = 1/6$ yields a comparable initial spread across families: variance-matching for logistic ($s = \sigma\sqrt{3}/\pi$) and Laplace ($b = \sigma/\sqrt{2}$); FWHM(Full width at half maximum)-matching for Cauchy ($\gamma = \sigma\sqrt{2\ln 2}$), since the Cauchy distribution has no finite variance. All other hyperparameters are held at the Form A defaults; we do not retune per family.

Table 2: Distribution-family ablation on EPFL (Form A, `icml` objective). Cells are mean cost relative to Gaussian ($\mathrm{cost}_{\mathrm{family}}/\mathrm{cost}_{\mathrm{Gaussian}}$); values below 1 indicate the family beats Gaussian. **Bold** marks the per-row winner. All four families share identical hyperparameters (no per-family retuning). Five benchmarks were run with 3 seeds and 14 with a single seed; `hyp` is excluded (load-time OOM for all four families).

Benchmark	seeds	Gaussian	Logistic	Laplace	Cauchy	Winner
`ctrl`	1	**1.000**	1.155	1.054	1.089	Gaussian
`int2float`	1	1.000	1.007	**0.994**	0.998	Laplace
`router`	1	1.000	1.006	0.939	**0.895**	Cauchy
`dec`	3	**1.000**	1.400	1.302	1.661	Gaussian
`cavlc`	3	1.000	1.004	1.019	**0.952**	Cauchy
`adder`	1	1.000	1.112	**0.534**	1.334	Laplace
`i2c`	1	1.000	**0.845**	0.853	0.975	Logistic
`max`	1	**1.000**	1.115	1.119	1.044	Gaussian
`bar`	3	1.000	0.946	0.921	**0.786**	Cauchy
`sin`	1	**1.000**	1.383	1.389	1.461	Gaussian
`arbiter`	3	1.000	1.142	1.157	**0.984**	Cauchy
`voter`	1	1.000	0.974	0.950	**0.944**	Cauchy
`square`	1	**1.000**	1.078	1.024	1.063	Gaussian
`multiplier`	3	**1.000**	1.103	1.077	1.109	Gaussian
`div`	1	1.000	0.999	1.000	**0.997**	Cauchy
Wins (out of 15)	—	6	1	2	6	—
Geomean cost / Gaussian	—	1.000	1.075	1.002	1.066	—
Mean rank (1 = best)	—	2.23	3.00	2.43	2.33	—

Across the 15 EPFL benchmarks where all four families are feasible, three of four families fall within $\sim 8\%$ on geometric-mean cost. For these tested families and instances, aggregate costs are similar; this does not establish comparable behavior for every symmetric unimodal CDF.

F Temporal-Proximity Ablation

Section 3.1 argues that a key advantage of the Gaussian parameterization over a categorical one is the *temporal-proximity* inductive bias: as μ_i shifts, the per-step probability mass P_i^d moves smoothly between *adjacent* time steps. We isolate this mechanism with a controlled ablation. Once P_i^d is computed according to 6, we apply a per-node fixed permutation of the d-axis (each node draws its

own permutation of $[0, D)$ once from the run seed and holds it constant). Each column of P still sums to 1, gradients still flow through (μ, σ), and the parameter count is unchanged — but the row index no longer corresponds to *actual time step*, so $\sum_d P_i^d$ no longer equals $\Pr(X_i \leq d)$ and per-step resource sums no longer aggregate operators that occupy the same step. A worse final cost under shuffling is direct evidence that the proximity mechanism contributed real optimization signal.

Shuffling changes the relationship between rows and time, so this intervention tests the combined effect of breaking the surrogate's temporal semantics rather than temporal proximity alone.

G GPU Efficiency Analysis

To evaluate the practical GPU efficiency of GauS, we performed a comparative GPU profiling analysis against the baseline, GS-Schedule, using the EPFL benchmarks. Figure 8 illustrates the peak GPU memory usage and compute utilization across various workloads.

Memory Scalability: GauS exhibits a remarkably consistent and relatively low memory footprint across all benchmarks, roughly linear with the size of the graph ($|V|$). In contrast, GS-Schedule shows a more drastic growth in memory consumption as the size ($|V|$) and the depth (D) of the graph increases, resulting out-of-memory (OOM) issues on the largest designs like square, multiplier, and div.

Compute Utilization: Our Gaussian parameterization consistently maintains near-100% averaged GPU utilization. This demonstrates that the $O(N)$ parameter space is not only memory-efficient but also highly amenable to parallel execution. Conversely, GS-Schedule struggles to saturate the GPU, with utilization dropping below 40% as the graph complexity increases, likely due theirs poor GPU implementation.

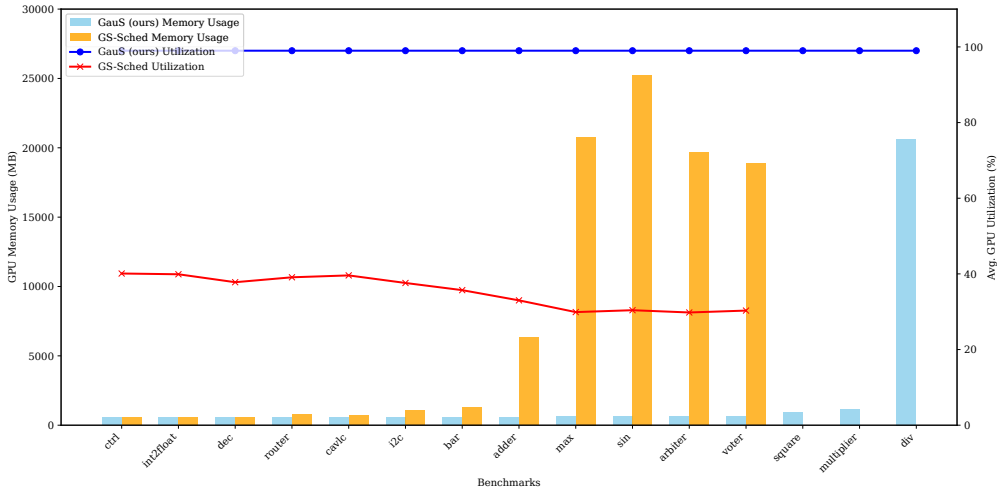

Figure 8: **GPU Profiling Results on EPFL** — The bar charts show the GPU memory usage, corresponding to the left axis. The curve show the Avg. GPU utilization, corresponding to the right axis. Avg. GPU utilization is the averaged GPU utilization (%) during the entire execution. Because GS-Schedule is out-of-memory (OOM) on square, multiplier, and div, no data is provided for GS-Schedule on these benchmarks

G.1 GauS CPU–GPU Runtime Comparison

To examine the role of hardware in GauS runtime, we compare one CPU thread with one A100 GPU on Form A for RW1–RW6 (Figure 9). Both runs use the same initialization, seed, and budget of 1,000 updates. On the approximately 1,000-node RW1–RW3 graphs, the CPU is 1.49–1.67× faster; on the approximately 5,000-node RW4–RW6 graphs, the GPU is 2.46–2.61× faster. GPU launch and synchronization overhead can dominate smaller workloads, whereas larger tensors offer more parallel work. Device-specific randomness and floating-point operations yield different optimization trajectories. This is a single-repetition comparison and does not isolate algorithmic effects from hardware effects in comparisons with other methods.

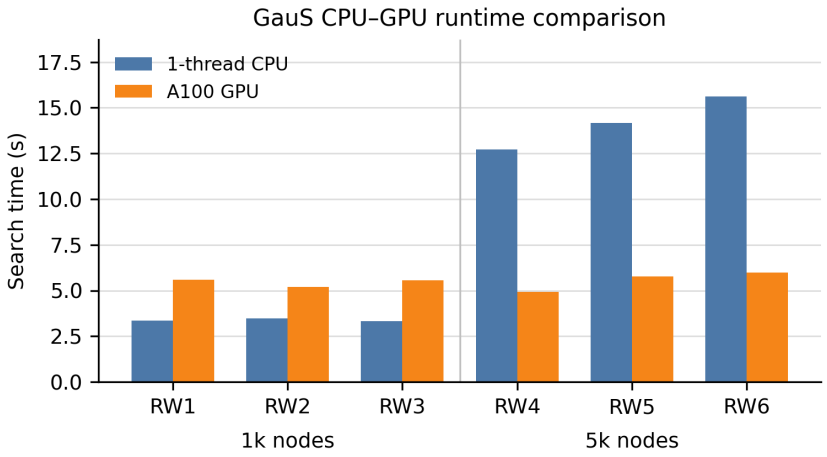

Figure 9: GauS Form A search time on one CPU thread and one A100 GPU, using the same initialization, seed, and 1,000-update budget. Each bar represents one run.

H Theoretical Analysis for Regular Scheduling

This section presents the thesis analysis for the regular resource-and-communication objective under unit separation ($g = 1$). The original Form A comparison uses chaining ($g = 0$), so these unit-gap guarantees should not be presented as a direct certificate of that experimental setting without adapting the assumptions. The results do not assume that feasible schedules are dense or approximately convex. Instead, they separate the exact convex structure of feasibility from the nonconvexity of the peak-resource objective and the Gaussian parameterization. The principal guarantees are additive and instance dependent: their terms are either user controlled, such as the `LogSumExp` temperature, or computable after optimization, such as Gaussian concentration and legalization movement.

H.1 Setup and Computational Limits

Let $[D] = \{0, \dots, D-1\}$ and let $\mathcal{F} \subseteq [D]^n$ be the set of unit-latency regular schedules on $G = (V, E)$, where $n = |V|$. Let $r_i > 0$ be the resource demand of v_i and let $\alpha \geq 0$. For compact theorem notation, define

$$R_d(\boldsymbol{s}) = \sum_{v_i \in V} r_i \mathbb{I}[s_i = d], \qquad R(\boldsymbol{s}) = \max_{d \in [D]} R_d(\boldsymbol{s}), \tag{20}$$

$$A(\boldsymbol{s}) = \sum_{(v_i, v_j) \in E} (s_j - s_i), \qquad C(\boldsymbol{s}) = R(\boldsymbol{s}) + \alpha A(\boldsymbol{s}), \quad \boldsymbol{s} \in \mathcal{F}. \tag{21}$$

This is the resource-and-communication objective for the unit-gap feasible set.

For the Gaussian relaxation, independently draw $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ and let

$$S_i = \mathrm{clip}_{[D]}\left(\left\lfloor X_i + \frac{1}{2} \right\rfloor\right), \qquad P_i^d = \Pr(S_i = d). \tag{22}$$

The clipping is implemented by folding the two Gaussian tails into the endpoint probabilities, as described after Equation 3.2. Writing $(x)_+ = \max\{x, 0\}$, define

$$m_d(\boldsymbol{\theta}) = \sum_{v_i \in V} r_i P_i^d, \qquad M(\boldsymbol{\theta}) = \max_{d \in [D]} m_d(\boldsymbol{\theta}), \tag{23}$$

$$A_+(\boldsymbol{s}) = \sum_{(v_i, v_j) \in E} (s_j - s_i)_+, \qquad \widehat{A}(\boldsymbol{\theta}) = \mathbb{E}[A_+(\boldsymbol{S})], \tag{24}$$

where $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$. The relaxed Form-A objective is

$$\widehat{C}_\tau(\boldsymbol{\theta}) = \tau \log \sum_{d=0}^{D-1} \exp\left(\frac{m_d(\boldsymbol{\theta})}{\tau}\right) + \alpha \widehat{A}(\boldsymbol{\theta}). \tag{25}$$

This is a smooth maximum of expected per-step resource loads, rather than the expected value of the discrete peak load. Section H.4 bounds the difference between the two.

The unit-latency dependency-violation count and its expectation are

$$\mathcal{V}_{Dep}(\boldsymbol{S}) = \sum_{(v_i, v_j) \in E} \mathbb{I}[S_j \leq S_i], \tag{26}$$

$$\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}) = \sum_{(v_i, v_j) \in E} \sum_{d_i=0}^{D-1} \sum_{d_j=0}^{d_i} P_i^{d_i} P_j^{d_j}. \tag{27}$$

In particular, equal-step producer–consumer placements are violations when operator latency is one.

Theorem H.1 (NP-hardness despite dense feasibility)**.** *Exact minimization of C is NP-hard even when $E = \varnothing$, $D = 2$, and therefore every assignment in $[D]^n$ is feasible.*

Proof. Given a PARTITION instance Karp [1972] with positive integer weights $a_1, \dots, a_n$, create one operator per item, set $r_i = a_i$, use two steps, and include no edges. The communication term is zero and every assignment is feasible. A schedule satisfying

$$R(\boldsymbol{s}) \leq \frac{1}{2} \sum_i a_i$$

exists if and only if the integers admit an equal partition. □

Thus neither a high feasible-set density nor a convex feasible hull makes global optimization easy. The result rules out an unconditional polynomial-time exact guarantee unless $P = NP$, but it does not preclude the additive, instance-dependent guarantees developed below.

H.2 Exact Feasibility and Legalization

Consider the continuous difference-constraint polytope

$$\mathcal{P} = \{\boldsymbol{x} \in [0, D-1]^n : x_j - x_i \geq 1,\ \forall (v_i, v_j) \in E\}\,. \tag{28}$$

Its constraint matrix consists of a directed node–arc incidence matrix and bound rows, and is totally unimodular Schrijver [1998]. Because the right-hand side is integral, every vertex of $\mathcal{P}$ is integral; consequently,

$$\mathcal{P} = \mathrm{conv}(\mathcal{F}). \tag{29}$$

Form-A feasibility therefore already has an exact convex representation.

Theorem H.2 (Feasibility and minimal forward repair)**.** *Let ℓ be the maximum number of edges on a directed path in G. Then:*

(i) $\mathcal{F} \neq \varnothing$ if and only if $\ell \leq D-1$;

(ii) feasibility can be decided and an ASAP schedule can be constructed in $O(|V| + |E|)$ time; and

(iii) if the instance is feasible, Algorithm 2 returns a schedule in $\mathcal{F}$ in $O(|V| + |E|)$ time and never exceeds its ALAP bounds.

After its input $\boldsymbol{b}$ is clamped to the ASAP/ALAP interval, the algorithm's output $\boldsymbol{z}$ is the componentwise smallest feasible schedule satisfying $\boldsymbol{z} \geq \boldsymbol{b}$.

Proof. For each vertex v, let a_v be the maximum path length from any source to v. Dynamic programming in topological order computes all a_v in linear time. The assignment $s_v = a_v$ satisfies every dependency and fits in $[D]$ exactly when $\max_v a_v = \ell \leq D-1$. Necessity follows by summing the unit gaps along a longest path.

Let b_v^{late} be the ALAP bound. After clamping, the topological pass computes

$$z_v = \max\left\{b_v,\ \max_{u \in \mathrm{pred}(v)} (z_u + 1)\right\}. \tag{30}$$

For every edge (u, v), the ALAP recursion gives $b_u^{late} + 1 \leq b_v^{late}$. Topological induction therefore shows $z_v \leq b_v^{late}$, so the pass cannot overflow the horizon and all dependencies are satisfied. If $\boldsymbol{y} \geq \boldsymbol{b}$ is any other feasible schedule, topological induction on Equation 30 gives $y_v \geq z_v$ for every v. Hence $\boldsymbol{z}$ is the componentwise minimal feasible forward repair. □

Equivalently, its displacement can be computed from

$$z_v = \max_{u \preceq v} \left(b_u + \mathrm{dist}(u, v)\right), \tag{31}$$

where $u \preceq v$ includes $u = v$ and a virtual source supplies the ASAP lower bound. The guarantee is stronger than saying that legalization merely attempts to find a feasible schedule. It does not, however, claim that the result is nearest under Euclidean or Hamming distance.

H.3 Deterministic Rounding Certificates

Let

$$\bar{s}_i = \mathrm{clip}_{[D]}\left(\left\lfloor \mu_i + \frac{1}{2} \right\rfloor\right), \qquad \varepsilon_i = \Pr(S_i \neq \bar{s}_i) = 1 - P_i^{\bar{s}_i}. \tag{32}$$

Every ε_i is directly available from the probability matrix. When $\mu_i \in [0, D-1]$ is not a half-integer, let

$$\gamma_i = \frac{1}{2} - |\mu_i - \bar{s}_i| > 0 \tag{33}$$

denote its distance to the nearest rounding boundary. Standard Gaussian tail bounds Boucheron et al. [2013] give

$$\varepsilon_i \le 2\Phi\left(-\frac{\gamma_i}{\sigma_i}\right) \le \exp\left(-\frac{\gamma_i^2}{2\sigma_i^2}\right). \tag{34}$$

Endpoint folding can only reduce this error. Thus a small standard deviation yields a strong rounding certificate when the mean is also separated from a rounding boundary.

Theorem H.3 (Mean-rounding feasibility certificate)**.** *If $E = \varnothing$, $\bar{\boldsymbol{s}}$ is feasible trivially. Otherwise, if*

$$\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}) < \min_{(v_i,v_j)\in E}(1-\varepsilon_i)(1-\varepsilon_j), \tag{35}$$

then $\bar{\boldsymbol{s}} \in \mathcal{F}$.

Proof. Suppose that $\bar{\boldsymbol{s}}$ violates an edge (v_i, v_j). Whenever $(S_i, S_j) = (\bar{s}_i, \bar{s}_j)$, the sampled schedule violates that edge as well. Independence gives

$$\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}) \ge \Pr(S_i = \bar{s}_i, S_j = \bar{s}_j) = (1-\varepsilon_i)(1-\varepsilon_j),$$

which contradicts Equation 35. □

This condition is sufficient but not necessary. When it does not hold, Algorithm 2 remains an unconditional fallback for every feasible regular-scheduling instance.

Let $\deg(i) = \deg^+(i) + \deg^-(i)$ and define the computable concentration charge

$$B(\boldsymbol{\theta}) = \sum_{v_i \in V} \left[r_i + \alpha(D-1)\deg(i)\right]\varepsilon_i. \tag{36}$$

Theorem H.4 (Gaussian relaxation and smoothing error)**.** *If $\bar{\boldsymbol{s}}$ is feasible, then*

$$\left|C(\bar{\boldsymbol{s}}) - \widehat{C}_\tau(\boldsymbol{\theta})\right| \le B(\boldsymbol{\theta}) + \tau \log D. \tag{37}$$

The sharper one-sided quality certificate is

$$C(\bar{\boldsymbol{s}}) \le \widehat{C}_\tau(\boldsymbol{\theta}) + B(\boldsymbol{\theta}). \tag{38}$$

Proof. Let $\bar{R}_d = \sum_i r_i \mathbb{I}[\bar{s}_i = d]$. For every step d,

$$\left|m_d - \bar{R}_d\right| \le \sum_i r_i \varepsilon_i, \tag{39}$$

and therefore

$$|M(\boldsymbol{\theta}) - R(\bar{\boldsymbol{s}})| \le \sum_i r_i \varepsilon_i. \tag{40}$$

For any vector $\boldsymbol{m}$,

$$\max_d m_d \le \tau \log \sum_d e^{m_d/\tau} \le \max_d m_d + \tau \log D. \tag{41}$$

For an edge (v_i, v_j), the function $(s_j - s_i)_+$ lies in $[0, D-1]$ and is unchanged when both endpoints equal their rounded means. Consequently,

$$|\mathbb{E}[(S_j - S_i)_+] - (\bar{s}_j - \bar{s}_i)_+| \le (D-1)(\varepsilon_i + \varepsilon_j). \tag{42}$$

Summing over edges gives the communication part of $B(\boldsymbol{\theta})$. Because $\bar{\boldsymbol{s}}$ is feasible, $(\bar{s}_j - \bar{s}_i)_+ = \bar{s}_j - \bar{s}_i$. Combining these inequalities proves Equation 37. The one-sided result does not incur $\tau \log D$ because the `LogSumExp` term is at least $M(\boldsymbol{\theta})$. □

The guarantee also extends to an infeasible rounded mean. Let

$$\boldsymbol{z} = \mathrm{Legalize}(\bar{\boldsymbol{s}}), \qquad \Delta_i = z_i - \bar{s}_i, \qquad \mathcal{M} = \{i : \Delta_i \ne 0\},$$

where $\mathcal{M}$ includes changes made by the initial ASAP/ALAP clamp, and define the a posteriori repair charge

$$J(\bar{\boldsymbol{s}}) = \sum_{i \in \mathcal{M}} r_i + \alpha \sum_i \deg(i)|\Delta_i|. \tag{43}$$

Moving the nodes in $\mathcal{M}$ can increase peak load by at most their total resource demand. Changing endpoint i by $|\Delta_i|$ changes the positive span of each incident edge by at most the same amount. Theorem H.2 and the one-sided argument above therefore give

$$\boxed{C(\boldsymbol{z}) \le \widehat{C}_\tau(\boldsymbol{\theta}) + B(\boldsymbol{\theta}) + J(\bar{\boldsymbol{s}})}. \tag{44}$$

All terms on the right are observable after optimization.

H.4 Finite-Sample Randomized Decoding

The expected violation has a direct probabilistic interpretation. Because $\mathcal{V}_{Dep}$ is a nonnegative integer, Markov's inequality gives

$$\Pr_{\boldsymbol{\theta}}(\boldsymbol{S} \notin \mathcal{F}) = \Pr(\mathcal{V}_{Dep}(\boldsymbol{S}) \ge 1) \le \widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}). \tag{45}$$

This certifies the Gaussian measure of the feasible region and avoids an assumption about the cardinal fraction $|\mathcal{F}|/D^n$.

Let $r_{\max} = \max_i r_i$. For $\delta_R, \delta_A \in (0,1)$, define

$$q_R(\delta_R) = \sqrt{2r_{\max} M(\boldsymbol{\theta}) \log\left(\frac{D}{\delta_R}\right)} + \frac{2r_{\max}}{3} \log\left(\frac{D}{\delta_R}\right), \tag{46}$$

$$q_A(\delta_A) = (D-1)\sqrt{\frac{1}{2}\left(\sum_i \deg(i)^2\right) \log\left(\frac{1}{\delta_A}\right)}. \tag{47}$$

Theorem H.5 (Feasible randomized rounding with bounded quality)**.** *If*

$$p_{\text{good}} = 1 - \widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}) - \delta_R - \delta_A > 0, \tag{48}$$

then a single independent draw $\boldsymbol{S}$ is feasible and satisfies

$$C(\boldsymbol{S}) \le \widehat{C}_\tau(\boldsymbol{\theta}) + q_R(\delta_R) + \alpha q_A(\delta_A) \tag{49}$$

with probability at least p_{good}. Consequently, among K independent samples, the probability that at least one satisfies Equation 49 is at least

$$1 - (1 - p_{\text{good}})^K. \tag{50}$$

Proof. For a fixed step d, $R_d(\boldsymbol{S})$ is a sum of independent random variables in $[0, r_{\max}]$, has mean m_d, and has variance at most $r_{\max} m_d \le r_{\max} M$. Bernstein's inequality Boucheron et al. [2013] followed by a union bound over the D steps gives

$$\Pr\left(R(\boldsymbol{S}) > M(\boldsymbol{\theta}) + q_R(\delta_R)\right) \le \delta_R. \tag{51}$$

Changing only S_i can change $A_+(\boldsymbol{S})$ by at most $(D-1)\deg(i)$. McDiarmid's bounded-difference inequality McDiarmid [1989] therefore gives

$$\Pr\left(A_+(\boldsymbol{S}) > \widehat{A}(\boldsymbol{\theta}) + q_A(\delta_A)\right) \le \delta_A. \tag{52}$$

Equation 45 bounds the probability of infeasibility. A union bound gives joint success probability at least p_{good}. On the feasible event, $A_+(\boldsymbol{S}) = A(\boldsymbol{S})$, while

$$M(\boldsymbol{\theta}) + \alpha \widehat{A}(\boldsymbol{\theta}) \le \widehat{C}_\tau(\boldsymbol{\theta}),$$

which proves Equation 49. Independent repetition proves Equation 50. □

This theorem describes an optional sampling decoder: draw a small batch, discard infeasible schedules, and retain the best candidate. The deterministic mean-round-and-legalize decoder remains the default and is covered by Equation 44.

H.5 Optimization Guarantees and Their Scope

For fixed ALM multipliers, consider the Form-A penalized objective

$$F_{\lambda,\rho}(\boldsymbol{\theta}) = \widehat{C}_\tau(\boldsymbol{\theta}) + \lambda \widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}) + \frac{\rho}{2}\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta})^2, \qquad \lambda, \rho \geq 0. \tag{53}$$

On a compact parameter region

$$\Omega = \{\mu_i \in [\mu_{\min}, \mu_{\max}],\ \sigma_i \in [\sigma_{\min}, \sigma_{\max}]\}, \qquad \sigma_{\min} > 0, \tag{54}$$

all Gaussian CDF terms are smooth. The finite sums, products, and `LogSumExp` are also smooth, so the Hessian is bounded on Ω and the gradient is L-Lipschitz for some finite L.

Theorem H.6 (First-order stationarity)**.** *Run gradient descent with step size $1/L$ on a fixed $F_{\lambda,\rho}$ and suppose the iterates remain in Ω. If $F_{\inf}$ is a lower bound on $F_{\lambda,\rho}$ in Ω, then*

$$\min_{0 \leq t < T} \|\nabla F_{\lambda,\rho}(\boldsymbol{\theta}_t)\|_2^2 \leq \frac{2L\left(F_{\lambda,\rho}(\boldsymbol{\theta}_0) - F_{\inf}\right)}{T}. \tag{55}$$

Proof. The descent lemma for an L-smooth function and a step of $1/L$ Beck [2023] gives

$$F_{\lambda,\rho}(\boldsymbol{\theta}_{t+1}) \leq F_{\lambda,\rho}(\boldsymbol{\theta}_t) - \frac{1}{2L}\|\nabla F_{\lambda,\rho}(\boldsymbol{\theta}_t)\|_2^2.$$

Summing for $t = 0, \ldots, T-1$, applying the lower bound, and taking the minimum gradient norm proves the result. □

Smoothness alone guarantees convergence only to first-order stationarity, not to a globally optimal schedule. A precise weaker-than-convexity landscape condition can strengthen the rate. Let $\Omega_0 \subseteq \Omega$ be the connected sublevel basin reached by the warm start, and suppose a local Polyak–Łojasiewicz (PL) condition holds Polyak [1963], Karimi et al. [2016]:

$$\frac{1}{2}\|\nabla F_{\lambda,\rho}(\boldsymbol{\theta})\|_2^2 \geq \mu_{\mathrm{PL}}\left(F_{\lambda,\rho}(\boldsymbol{\theta}) - F^\star_{\Omega_0}\right), \quad \forall \boldsymbol{\theta} \in \Omega_0, \tag{56}$$

where $F^\star_{\Omega_0} = \inf_{\boldsymbol{\theta}\in\Omega_0} F_{\lambda,\rho}(\boldsymbol{\theta})$. The PL condition does not require convexity, but excludes suboptimal stationary points within the relevant basin. If the iterates remain in Ω_0, then

$$F_{\lambda,\rho}(\boldsymbol{\theta}_T) - F^\star_{\Omega_0} \leq \left(1 - \frac{\mu_{\mathrm{PL}}}{L}\right)^T \left(F_{\lambda,\rho}(\boldsymbol{\theta}_0) - F^\star_{\Omega_0}\right). \tag{57}$$

The local condition does not control the basin gap

$$\Gamma_{\mathrm{basin}} = F^\star_{\Omega_0} - \inf_{\boldsymbol{\theta}\in\Omega} F_{\lambda,\rho}(\boldsymbol{\theta}) \geq 0. \tag{58}$$

Assuming that the warm-start basin contains a global relaxed minimizer sets this gap to zero; without such a landscape assumption, a global guarantee would conflict with Theorem H.1.

For completeness, the preceding terms combine into an end-to-end conditional bound. Let $\boldsymbol{s}^\star$ be an optimal discrete schedule and encode it by $\boldsymbol{\theta}^\star_\kappa$ with $\mu_i = s^\star_i$ and $\sigma_i = \kappa > 0$. With

$$\varepsilon_\kappa = 2\Phi\left(-\frac{1}{2\kappa}\right), \tag{59}$$

Equation 34 and a union bound give

$$B(\boldsymbol{\theta}^\star_\kappa) \leq \varepsilon_\kappa \left[\sum_i r_i + 2\alpha(D-1)|E|\right], \tag{60}$$

$$\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}^\star_\kappa) \leq 2|E|\varepsilon_\kappa. \tag{61}$$

Theorem H.7 (Conditional end-to-end discrete quality)**.** *Assume $\boldsymbol{\theta}^\star_\kappa \in \Omega$ and the smoothness and local-PL conditions above. Run T fixed-multiplier gradient steps with step size $1/L$, assume the iterates remain in Ω_0, then mean-round $\boldsymbol{\theta}_T$ and apply Algorithm 2 to obtain $\boldsymbol{z}_T$. Then*

$$\begin{aligned} C(\boldsymbol{z}_T) \leq\ & C(\boldsymbol{s}^\star) + \tau \log D \\ & + B(\boldsymbol{\theta}_T) + J(\bar{\boldsymbol{s}}_T) \\ & + B(\boldsymbol{\theta}^\star_\kappa) + \lambda \widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}^\star_\kappa) + \frac{\rho}{2}\widehat{\mathcal{V}}_{Dep}(\boldsymbol{\theta}^\star_\kappa)^2 \\ & + \Gamma_{\mathrm{basin}} + \left(1 - \frac{\mu_{\mathrm{PL}}}{L}\right)^T \left(F_{\lambda,\rho}(\boldsymbol{\theta}_0) - F^\star_{\Omega_0}\right). \end{aligned} \tag{62}$$

Proof. The legalized-decoder certificate gives

$$C(\boldsymbol{z}_T) \le \widehat{C}_\tau(\boldsymbol{\theta}_T) + B(\boldsymbol{\theta}_T) + J(\bar{\boldsymbol{s}}_T) \le F_{\lambda,\rho}(\boldsymbol{\theta}_T) + B(\boldsymbol{\theta}_T) + J(\bar{\boldsymbol{s}}_T).$$

The PL rate and the basin-gap definition bound $F_{\lambda,\rho}(\boldsymbol{\theta}_T)$ by the global relaxed infimum plus the last two terms of Equation 62. That infimum is no greater than $F_{\lambda,\rho}(\boldsymbol{\theta}^\star_\kappa)$. Applying the upper side of Equation 37 to the feasible $\boldsymbol{s}^\star$ and retaining the nonnegative ALM penalty gives the remaining terms. □

Equation 62 makes each source of error explicit: $\tau \log D$ is smoothing error; B measures Gaussian concentration and rounding; J measures legalization movement; the terms at $\boldsymbol{\theta}^\star_\kappa$ vanish exponentially as $\kappa \to 0$; Γ_{basin} is the unavoidable global-landscape term; and the last term is finite-iteration error. The fixed-multiplier gradient-descent results characterize a standard inner loop of ALM. The implementation uses Adam and updates multipliers during optimization, so we do not claim that the stationarity or PL rates apply unchanged to that time-varying practical loop.

I Full Tradeoff Results

I.1 Formulation A

The full tradeoff analysis on formulation A is shown in Figure 10 and 11, GauS is on the Pareto frontier for most of the problem workloads, especially larger ones.

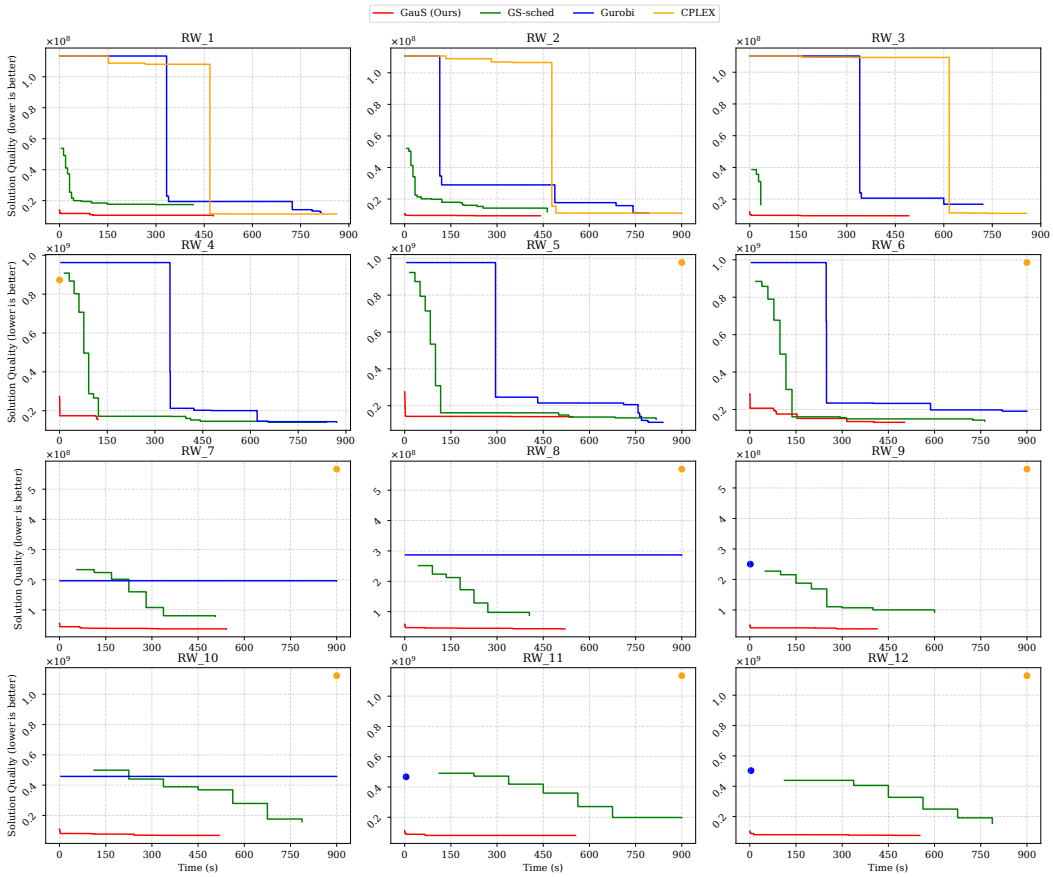

Figure 10: Comprehensive tradeoff comparison of RW on **Formulation B:** *Latency constrained, resource and communication optimization* — A 15-minute time limit is applied to all methods. Heuristics such as FDS and List Scheduling are represented as horizontal lines to reflect their fast convergence to final solution quality.

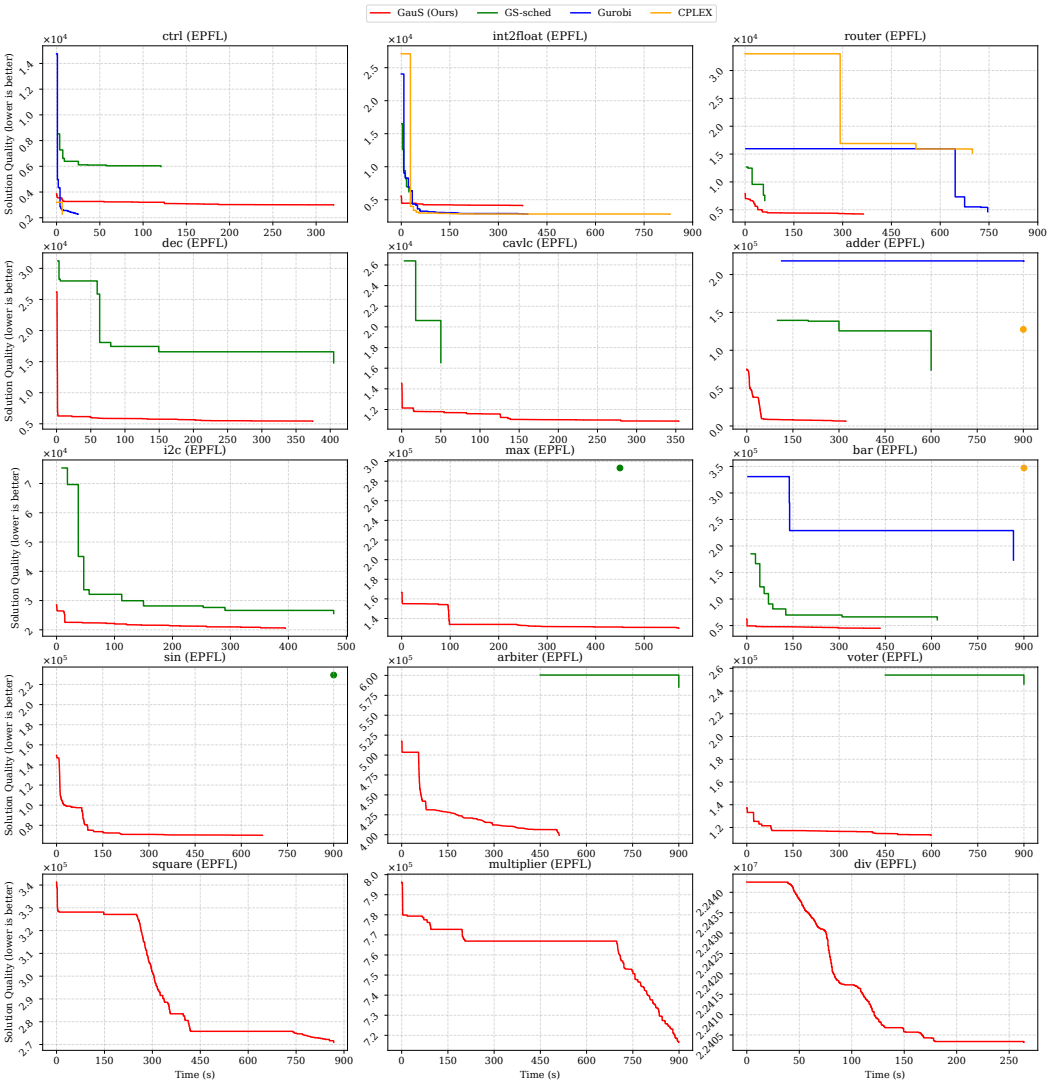

Figure 11: Comprehensive tradeoff comparison of EPFL on **Formulation A:** *Latency constrained, resource and communication optimization* — A 15-minute time limit is applied to all methods.

I.2 Formulation B and C

The comprehensive tradeoff results on formulation B and C are shown in Figure 12, 13, 14 and 15. The results demonstrate that while exact solvers like CPLEX and Gurobi achieve high-quality solutions for smaller problem instances, GauS is better in scalability, delivering good solutions on large-scale benchmarks. For medium-sized cases, our method offers a highly competitive tradeoff, reaching high-quality solutions orders of magnitude faster than exact solvers. Furthermore, GauS consistently outperforms fast heuristics, such as List Scheduling and FDS, which often usually leads to suboptimal results.

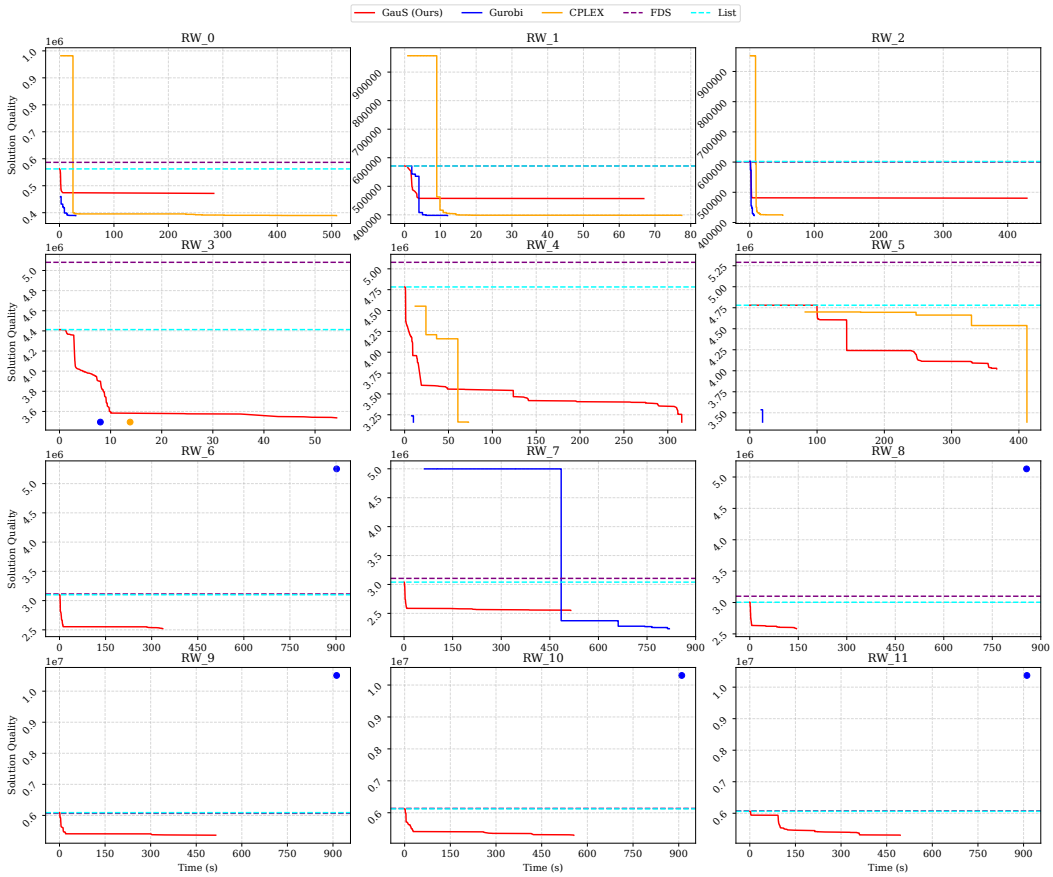

Figure 12: Comprehensive tradeoff comparison of RW on **Formulation B:** *Latency constrained, memory footprint optimization* — A 15-minute time limit is applied to all methods. Heuristics such as FDS and List Scheduling are represented as horizontal lines to reflect their fast convergence to final solution quality.

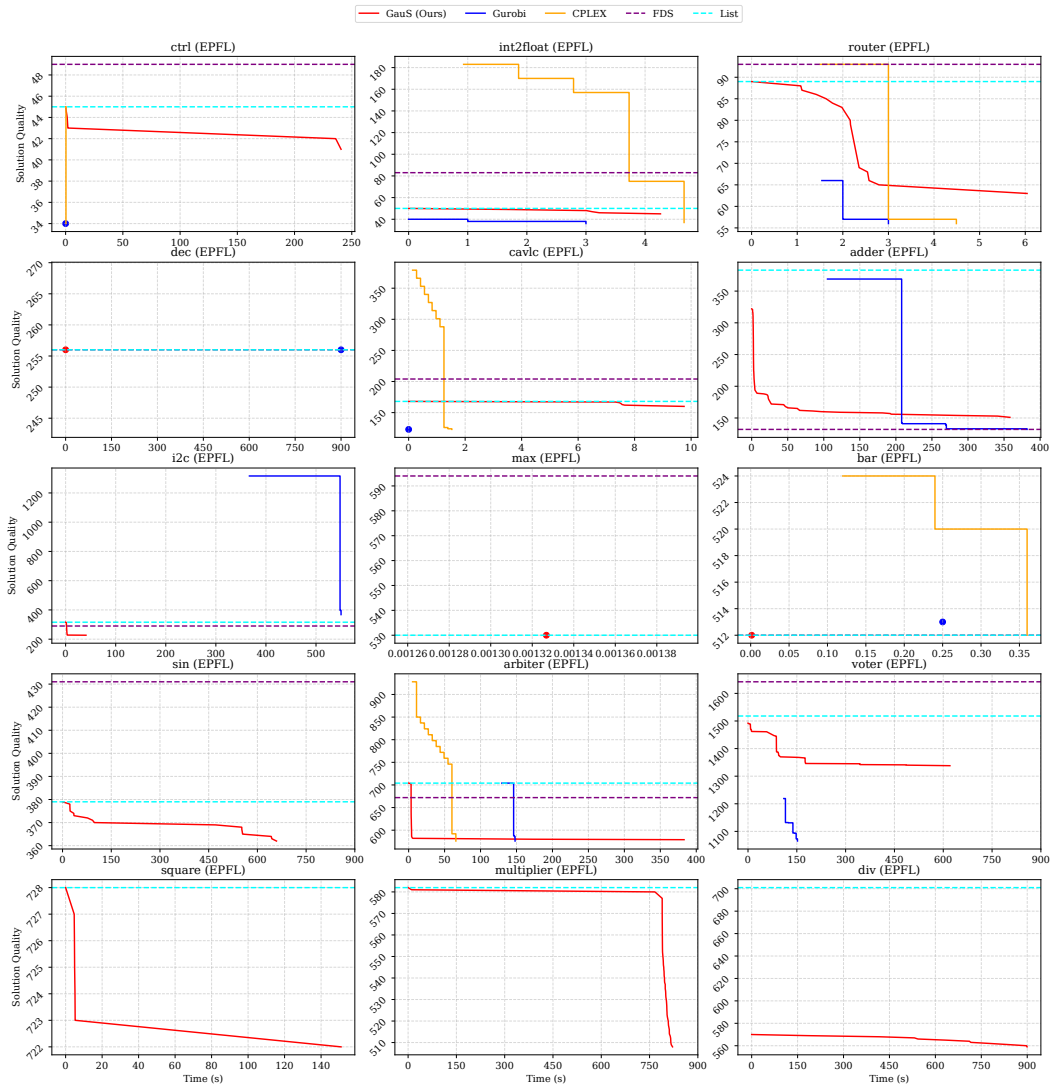

Figure 13: Comprehensive tradeoff comparison of EPFL on **Formulation B:** *Latency constrained, memory footprint optimization* — A 15-minute time limit is applied to all methods. Heuristics such as FDS and List Scheduling are represented as horizontal lines to reflect their fast convergence to final solution quality.

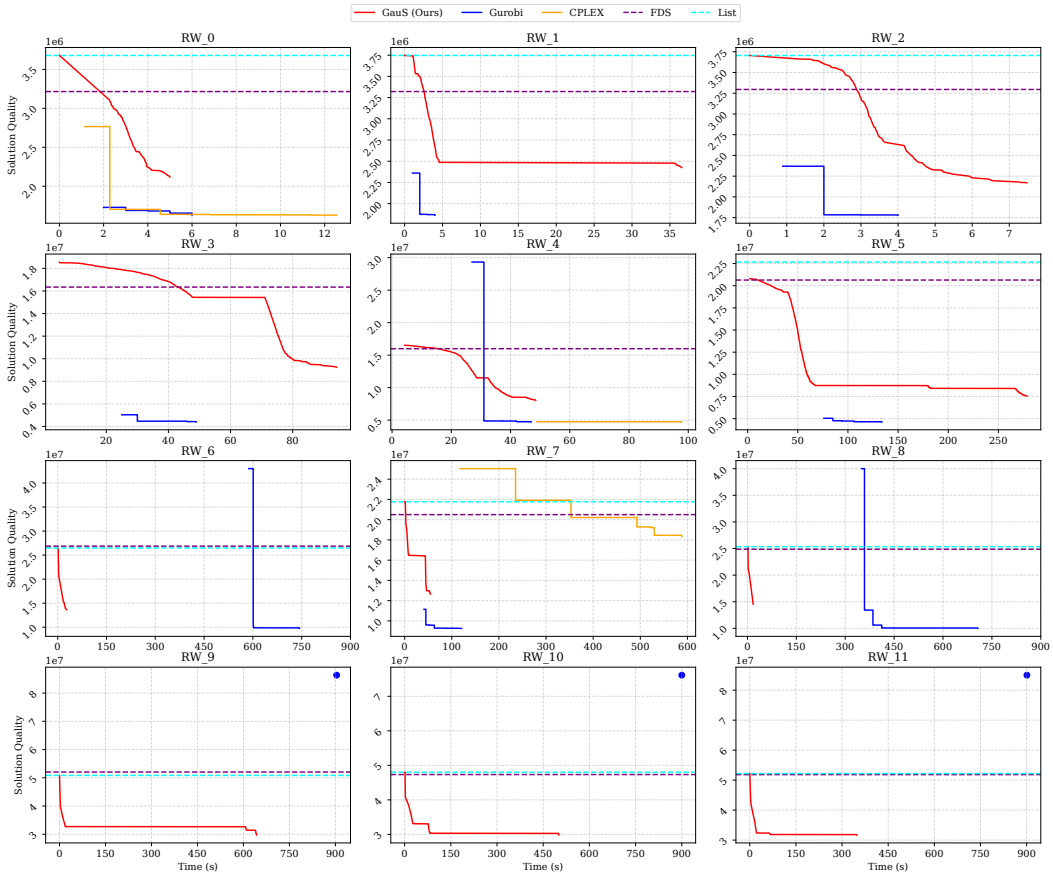

Figure 14: Comprehensive tradeoff comparison of RW on **Formulation C:** *Latency, resource, and recurrences constrained, memory footprint modulo optimization* — A 15-minute time limit is applied to all methods. Heuristics such as FDS and List Scheduling are represented as horizontal lines to reflect their fast convergence to final solution quality.

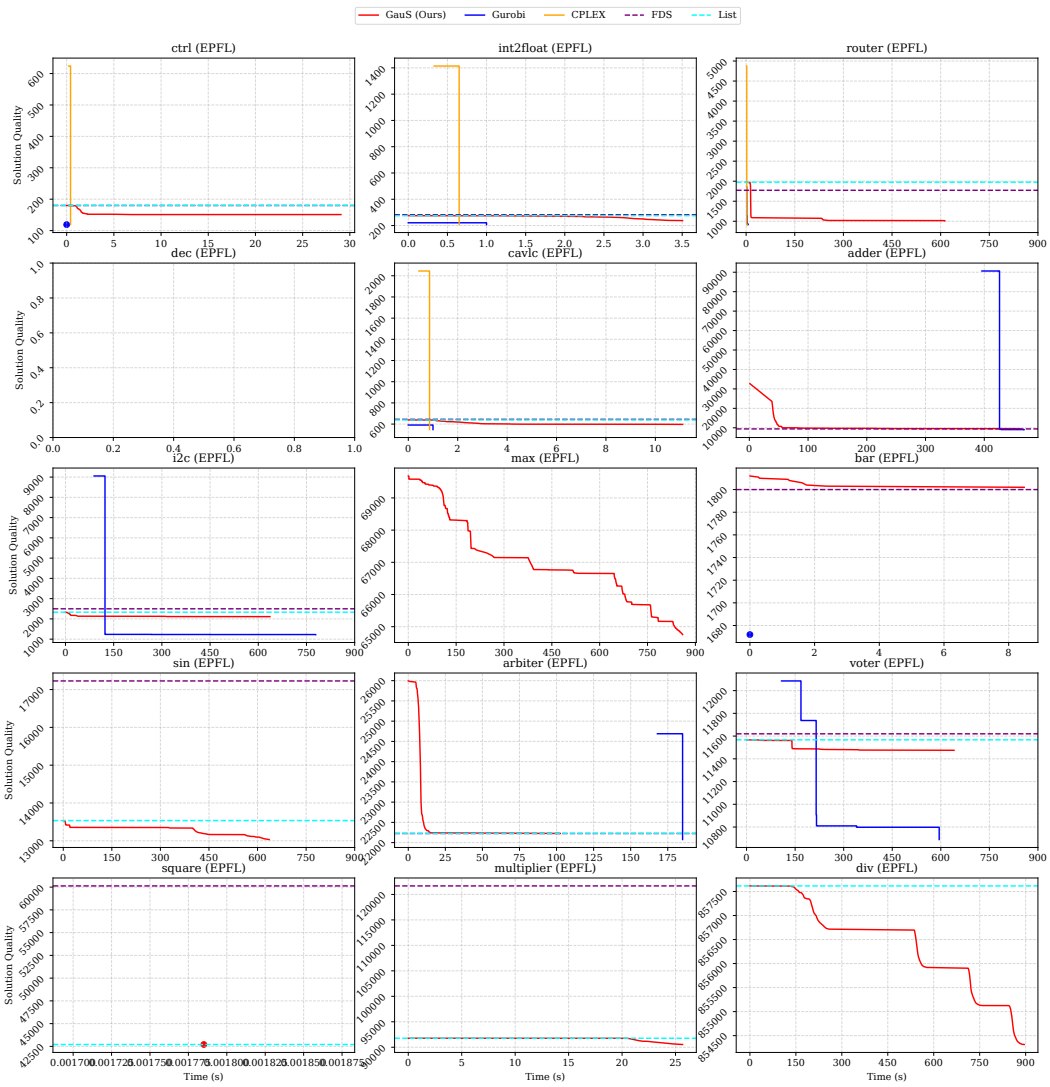

Figure 15: Comprehensive tradeoff comparison of EPFL on **Formulation C:** *Latency, resource, and recurrences constrained, memory footprint modulo optimization* — A 15-minute time limit is applied to all methods. Heuristics such as FDS and List Scheduling are represented as horizontal lines to reflect their fast convergence to final solution quality. There is no feasible solution for dec, thus plot is left empty.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The main claims reflect the papers's contribution and scope and are supported emperically.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

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

   Answer: [Yes]

   Justification: The limitations are discussed in the conclusion and the appendices.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: The assumptions and proofs for the stated regular-scheduling results are given in Appendix H.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: All the experiment details are included in the evaluation and appendices. We also empirically show that the experimental results are robust to parameter selection in general.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

Answer: [Yes]

Justification: The code and data are submitted for experimental results reproduction.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Details can be found in evaluation and appendices D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: The proposed method is deterministic without variance, thus we do not have error bars. We extensively evaluate it on various benchmarks to demonstrate significance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Evaluation, Appendix D, and Appendix I provide all the details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We reviewed the NeurIPS code of Ethics and conform with it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: Our method solely focuses on improving the scalability and efficiency of the scheduling algorithm. It does not impact the society at large.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: We do not add any realistic new data or models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Yes, we comply with the licenses and terms of use as specified by the original creators of the data and code involved in this paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: We introduces new synthetic data. The data generation code is provided and the format is documented. No privacy or copyright issue is involved as the new data is purely synthetic.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: We used LLMs for editing the the paper.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.