ITC 2/55
Information Technology and Control
Vol. 55 / No. 2/ 2026
pp. 671-686
DOI 10.5755/j01.itc.55.2.44070

CoDSA: A Hybrid Tensor Compilation Framework Integrating Coordinate Descent with Dynamic Simulated Annealing

Received 2026/01/13 | Accepted after revision 2026/05/22

HOW TO CITE: Sun, R., He, H., Zhang, G., Zhou, W. (2026) CoDSA: A Hybrid Tensor Compilation Framework Integrating Coordinate Descent with Dynamic Simulated Annealing. *Information Technology and Control*, 55(2), 671-686. https://doi.org/10.5755/j01.itc.55.2.44070

CoDSA: A Hybrid Tensor Compilation Framework Integrating Coordinate Descent with Dynamic Simulated Annealing

Ruiting Sun, Honglu He, Guanwen Zhang*, Wei Zhou

School of Electronics and Information, Northwestern Polytechnical University, Xi'an, Shaanxi, 710129, China; e-mails: sunrt@mail.nwpu.edu.cn (R.S.); hehl@mail.nwpu.edu.cn (H.H.); guanwen.zh@nwpu.edu.cn (G.Z.); zhouwei@nwpu.edu.cn (W.Z.)

Corresponding author: guanwen.zh@nwpu.edu.cn

The deployment of Deep Learning models on heterogeneous hardware necessitates optimized tensor programs to utilize computational resources. While auto-tuning frameworks like Ansor employ evolutionary search strategies, they suffer from limitations, including slow convergence, memory consumption, and a tendency to stagnate in local optima within high-dimensional, non-convex search spaces. To address these challenges, this paper proposes CoDSA (Coordinate Descent with Dynamic Simulated Annealing), a hybrid auto-tuning framework that integrates an enhanced Droplet Search algorithm into the Ansor ecosystem. We introduce three improvements to the droplet coordinate descent strategy: 1) a Dynamic Simulated Annealing mechanism to escape local optima; 2) an Adaptive Step-Size Control modulated by an exploration factor to balance global traversal and local refinement; and 3) a Diversity Control mechanism utilizing cosine similarity to enforce orthogonal exploration. Experiments on NVIDIA GPUs using Vision Transformer, DeiT, and Swin Transformer models demonstrate that CoDSA outperforms the Ansor framework. Specifically, for the ViT-Base model, our method reduces search time by 37.2% and memory usage by 53.1%, while improving inference latency by up to 26.5% on complex architectures like Swin-Small. This work bridges the gap between the global exploration capability of evolutionary methods and the rapid exploitation efficiency of coordinate descent.

KEYWORDS: Tensor Compilers, Auto-scheduling, Coordinate Descent, Simulated Annealing, Vision Transformers.

1. Introduction

The rapid evolution of Deep Learning has catalyzed a paradigm shift in computer vision and natural language processing, exemplified by the dominance of Transformer-based architectures such as Vision Transformers (ViT) [8], Data-efficient Image Transformers (DeiT) [24], and Swin Transformers [17]. To meet the stringent latency and resource constraints of real-world deployment, these models must be executed on a diverse array of heterogeneous hardware backends, ranging from high-performance server-class GPUs and TPUs to resource-constrained edge devices [26, 32]. In emerging industrial scenarios, such as autonomous driving systems and privacy-sensitive IoT home monitoring networks, relying on cloud connectivity or host servers for model compilation is often infeasible. These scenarios demand resource-efficient adaptive compilation, where models must be dynamically re-compiled and fine-tuned locally under strict time and host-memory constraints. This creates a critical requirement for time and memory efficient compilers that can operate within constrained hardware environments. However, the increasing complexity of modern hardware, characterized by intricate memory hierarchies and specialized compute primitives, widens the abstraction gap between high-level model definitions and low-level machine code. Consequently, achieving peak hardware performance requires meticulous optimization of tensor operators (kernels), involving transformations such as loop tiling, vectorization, thread binding, and memory layout permutation.

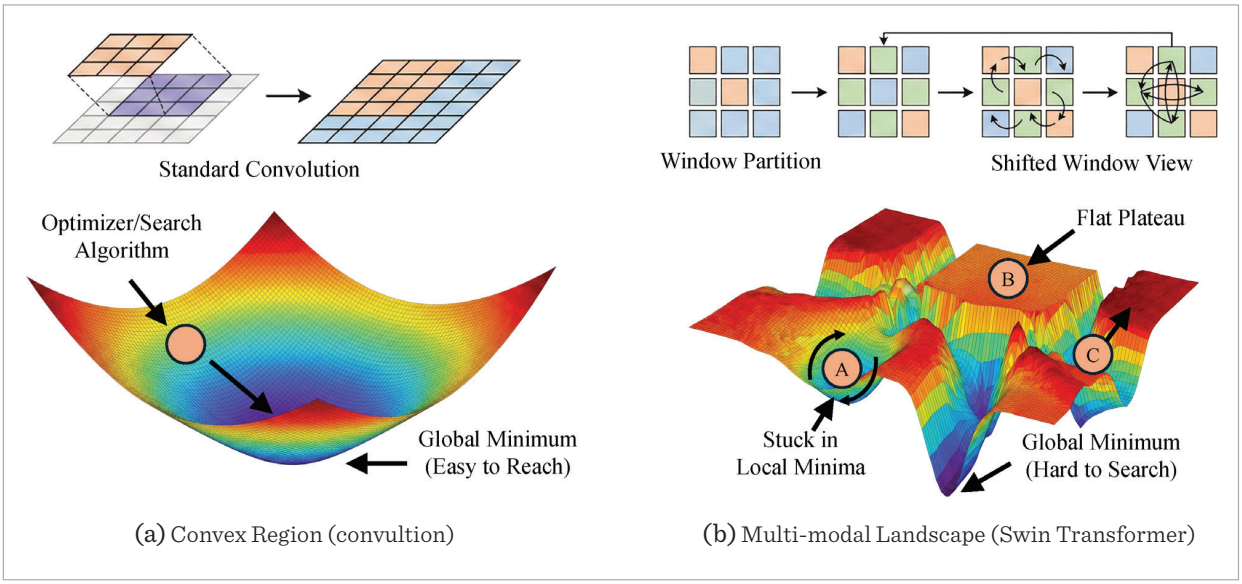
This optimization burden was shouldered by vendor-specific operator libraries like cuDNN [5] or MKL. While effective, these libraries are labor-intensive to maintain and lack the flexibility to optimize novel operators. More critically, manual libraries fail to perform global fusion optimization across operator boundaries. While manual, vendor-specific libraries effectively bridge the abstraction gap for standard, isolated operators, they suffer from a severe scalability bottleneck. Overcoming this fundamental scalability limitation and minimizing the high engineering cost associated with maintaining deep fusion patterns are the primary drivers behind the necessity and widespread adoption of automated tensor schedulers. To address this scalability bottleneck, automated tensor compilers such as Apache TVM [6], Halide [21], and Tensor Comprehensions [27] have emerged as essential infrastructure. These systems frame tensor program generation as a search problem within a massive space of potential schedules.

Unlike predecessors relying on manual templates [7], Ansor [33] automatically constructs a comprehensive search space using hierarchical sketches and employs an evolutionary search algorithm, guided by a learned XGBoost cost model, to identify high-performance schedules. Despite its proven efficacy, Ansor's evolutionary search strategy exhibits intrinsic limitations when applied to the increasingly complex search spaces of modern Transformer workloads. First, the search is susceptible to local optima because the random initialization and mutation operators lack directional guidance; consequently, in the highly non-convex landscapes of complex operators such as shifted window attention in Swin Transformers illustrated in Figure 1, Ansor often stagnates at suboptimal configurations and fails to exploit the hardware's theoretical peak performance. Second, it suffers from low search efficiency, as the evolutionary process relies on large populations and random mutations that typically necessitate thousands of hardware measurements to converge, resulting in tuning sessions that can span hours or days for a single network [33]. Third, the evolutionary approach demands excessive resource consumption. Maintaining large population histories and frequently inferring the XGBoost cost model imposes a substantial memory footprint. Our analysis reveals that tuning a standard ViT-Base model via Ansor consumes approximately 1903 MiB of memory. On edge devices with limited shared RAM, this footprint is prohibitive and frequently leads to out-of-memory compilation crashes. CoDSA aims to reduce the dependence on heavy host-side tuning infrastructure. By reducing the tuning memory footprint to 893 MiB in our experiments, it improves the feasibility of memory-constrained tuning settings.

In response to the inefficiencies of evolutionary methods, the *Droplet Search* algorithm [4] was proposed as a lightweight alternative. By employing a

Figure 1

Conceptual illustration of the optimization landscape contrast between convolution and Swin shifted-window attention. (a) Smooth/near-convex region (convolution). (b) Rugged multi-modal landscape with local minima/plateaus (Swin), motivating barrier-crossing exploration in CoDSA.



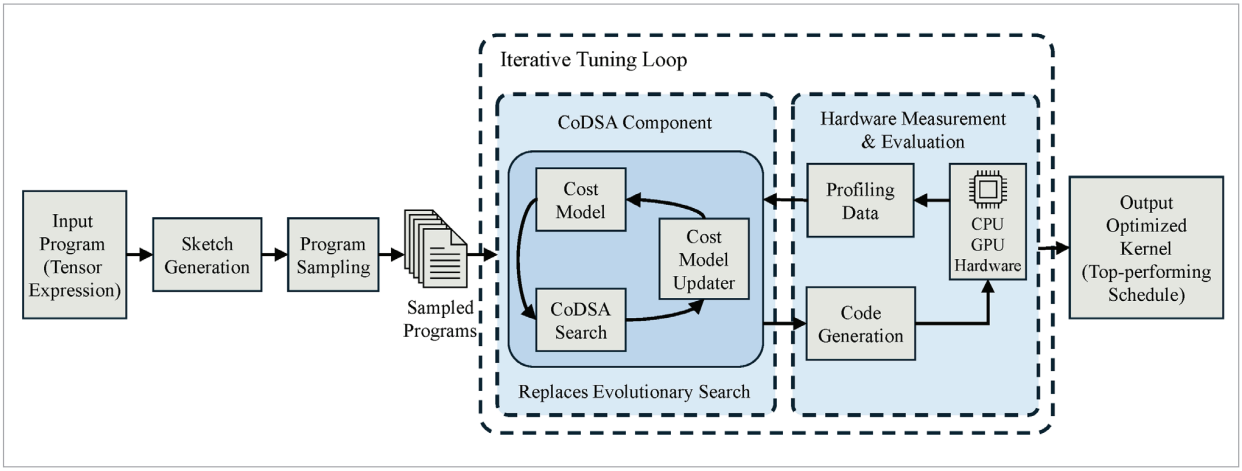
(a) Convex Region (convultion) (b) Multi-modal Landscape (Swin Transformer)

deterministic, greedy Coordinate Descent strategy, it achieves rapid convergence and minimal memory overhead. However, our investigations reveal that this strictly greedy approach often fails on the non-convex optimization landscapes of modern vision backbones. Without the ability to traverse performance barriers, the search frequently terminates prematurely at shallow local minima, yielding performance inferior to fully-tuned Ansor.

To reconcile the global exploration capabilities of Ansor with the rapid exploitation efficiency of Droplet Search, this paper presents **CoDSA** (**Co**ordinate **D**escent with Dynamic **S**imulated **A**nnealing). We integrate an enhanced version of Droplet Search directly into Ansor's tuning loop, using Ansor's sketches as high-quality initialization points while replacing the inefficient evolutionary search with a refined coordinate descent mechanism, illustrated in Figure 2. The novelty of CoDSA lies in its tensor-compilation-specific integration. CoDSA converts Ansor's sketch-derived schedule space into a valid coordinate-wise trajectory search, accepts or rejects candidates using hardware-measured latency rather than a generic objective surrogate, dynamically changes the schedule-step granularity, and filters redundant configurations before measurement using a normalized representation of tensor-schedule knobs. Thus, CoDSA is a schedule-aware search policy for template-free tensor compilation, rather than a generic combination of existing heuristics. To overcome the limitations of the original Droplet algorithm regarding local optima and diversity, we introduce three strategic algorithmic improvements:

- **CoDSA-Core (Dynamic Simulated Annealing):** We incorporate a probabilistic acceptance criterion governed by a dynamically decaying cooling rate. This allows the search to traverse performance barriers and escape local basins during the early exploration phases, addressing the fundamental limitation of greedy descent strategies.
- **CoDSA-Zoom (Adaptive Step Size Control):** Addressing the inefficiency of fixed-step traversal in high-dimensional spaces, we design an adaptive step size strategy modulated by an exploration factor. This enables coarse-grained jumps in initial iterations for global coverage and fine-grained local tuning as convergence approaches.
- **CoDSA-Ortho (Orthogonal Diversity Control):** To mitigate redundant sampling and mode collapse, we implement a diversity check based on cosine similarity. This enforces the exploration

Figure 2
Ansor auto-scheduling pipeline with CoDSA as a drop-in replacement for evolutionary search. CoDSA operates in the measurement loop and feeds profiling results back to the cost model updater.



of orthogonal parameter subspaces, ensuring comprehensive coverage of the optimization landscape and maximizing information gain from each measurement.

Extensive experiments on NVIDIA GPUs with ViT, DeiT, and Swin Transformer models demonstrate that CoDSA significantly outperforms the native Ansor framework. For the ViT-Base model, our method reduces search time by 37.2% and memory usage by 53.1%, while improving inference latency by up to 26.5% on complex architectures like Swin-Small. These results validate that integrating structurally-guided coordinate descent with dynamic exploration strategies offers a superior trade-off between tuning efficiency and code quality.

2. Related Work

The efficient deployment of Deep Learning models on heterogeneous hardware necessitates bridging the significant abstraction gap between high-level computational graphs and low-level machine code. This section reviews the evolution of automated tensor scheduling, analyzing the trade-offs between search space coverage and compilation efficiency. Furthermore, we examine the specific optimization heuristics—Coordinate Descent, Simulated Annealing, and Diversity Control—that constitute the theoretical foundation of our proposed CoDSA framework.

2.1. Auto-Scheduling in Deep Learning Compilers

The core challenge in deep learning compilation is the automated generation of high-performance kernel code such as CUDA or LLVM IR (Intermediate Representation) from high-level operator definitions. Early frameworks like Halide [21] and TVM [6] pioneered the decoupling of algorithm definitions from execution schedules, creating a vast combinatorial space of potential program variants that necessitates automated traversal.

2.1.1. From Template-Based to Template-Free Search

First-generation auto-tuning systems, such as AutoTVM [7], relied on manually designed schedule templates. While effective for standard operators like matrix multiplications, this approach lacks scalability as it requires expert engineering for every new operator or hardware target. To address this rigidity, Ansor [33] introduced a template-free, hierarchical search mechanism. Ansor automatically constructs a comprehensive search space using sketches to define program structure and annotations for low-level details such as tiling and unrolling. It employs an evolutionary search algorithm (genetic algorithm) guided by a learned XGBoost cost model to identify performant schedules. This methodology, extended by FlexTensor [35] and MetaSchedule [22], represents the state-of-the-art in terms of flexibility and search space coverage.

2.1.2. Search Efficiency and Budget Constraints

Despite their flexibility, evolutionary search strategies are inherently sample-inefficient. They typically require thousands of hardware measurements (trials) to converge, creating a significant bottleneck in tuning time and measurement budget. Recent research has sought to mitigate this inefficiency through distinct paradigms:

- **Analytical Construction:** Diverging from stochastic search, white-box compilers such as Rammer [18], Roller [36], and Welder [23] analytically construct schedules based on hardware abstractions. For instance, Roller aligns rTiles with memory bank capacities to maximize throughput without iterative tuning. While these methods offer orders-of-magnitude faster compilation, they often lack the generality to optimize irregular operators or complex fusion patterns where analytical modeling is intractable.
- **Search Space Pruning:** Approaches like Heron [2] and Bolt [29] utilize hardware intrinsics and mathematical constraints to prune invalid regions of the search space. By enforcing a stricter lower bound, they reduce the volume of the design space that the search algorithm must traverse.
- **Learned Policies:** Advanced search policies such as ProTuner [10] (using Monte Carlo Tree Search) and Chameleon [1] (using Reinforcement Learning) attempt to improve exploration efficiency. Similarly, transfer learning approaches like TenSet [34] and TLP [31] pre-train cost models on massive datasets to guide the search from a better initialization.

However, a critical gap remains: purely constructive methods sacrifice the optimality found by exhaustive search, while learning-based policies often incur prohibitive computational overhead and memory usage. Our work addresses this by enhancing lightweight coordinate descent heuristics to achieve the robustness of evolutionary methods without the associated resource cost.

2.2. Local Search with Exploration and Diversity

To operate effectively within a constrained measurement budget, we draw upon and adapt three specific optimization paradigms.

2.2.1. Coordinate Descent and the Droplet Hypothesis

Coordinate Descent minimizes a multivariate objective function by optimizing along one direction at a time [20,28]. In tensor compilation, the Droplet Search algorithm [4] adapts Coordinate Descent to the scheduling space, positing the Droplet Expectation: that high-performance schedules cluster in a convex region around a valid initialization. While Droplet Search offers exceptional speed and minimal memory footprint, it is fundamentally a greedy algorithm. As noted in optimization literature [25], greedy Coordinate Descent is prone to stagnation in local optima, particularly in the high-dimensional, non-convex landscapes of complex operators like the Swin Transformer [17], where performance barriers prevent simple descent algorithms from reaching the global optimum.

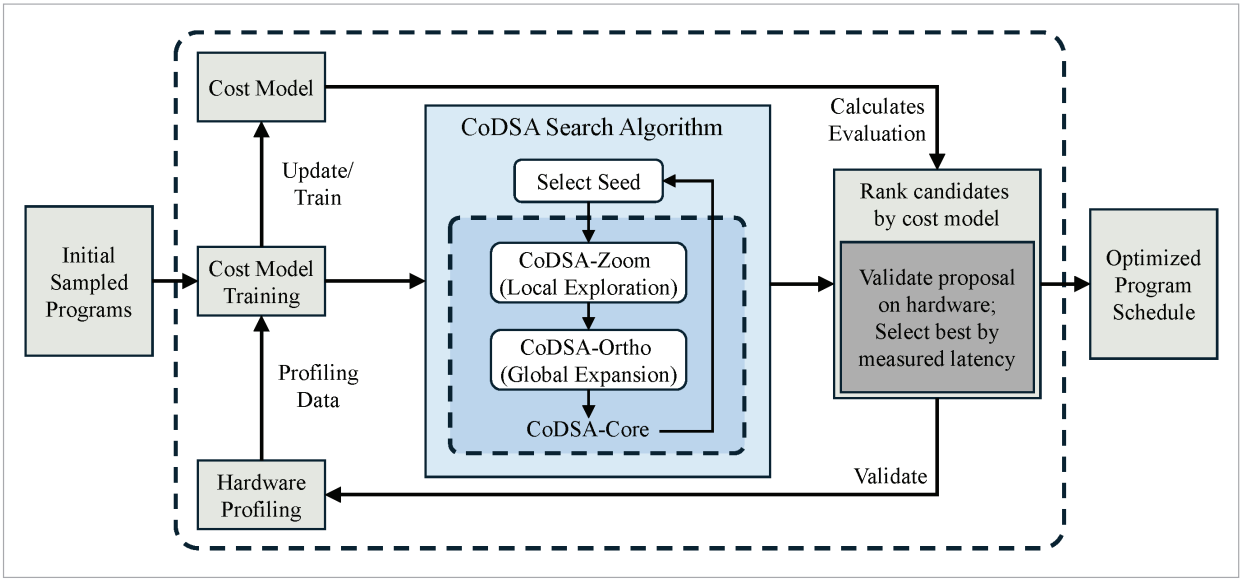
2.2.2. Simulated Annealing for Barrier Traversal

To overcome the limitations of greedy search, we integrate Simulated Annealing (SA) [15]. SA permits the probabilistic acceptance of inferior solutions, enabling the search to traverse performance barriers. While theoretical convergence requires logarithmic cooling [9], practical implementations typically employ Adaptive Simulated Annealing [11-12] to rescale temperature based on parameter sensitivity. In the context of our CoDSA-Core module, we employ a dynamic cooling schedule to balance early-stage global exploration with late-stage convergence, addressing the specific stability variance of tensor parameters [13, 30].

2.2.3. Orthogonal Diversity and Step Control

The efficiency of local search is often compromised by mode collapse, where the algorithm redundantly evaluates syntactically distinct but semantically identical schedules. Quality-Diversity algorithms like MAP-Elites [19] emphasize discovering behaviorally distinct solutions. To enforce this within a coordinate descent framework, we incorporate Orthogonal Exploration [14]. By utilizing Cosine Similarity [3] to penalize candidates colinear with visited states, we maximize the information gain of each trial [16]. Furthermore, addressing the inefficiency of fixed-step traversal in discrete spaces, we adopt adaptive step-size control, allowing the search to transition from coarse-grained jumps to fine-grained refinement, ensuring efficient coverage of the search space under strict budget constraints.

Figure 3
End-to-end CoDSA search loop inside Ansor. Candidates are generated by Zoom, pruned by Ortho for novelty, accepted by Core via simulated annealing, and validated on hardware for cost-model updates.



3. Methodology

This section articulates the design and implementation of CoDSA, a hybrid auto-tuning framework designed to overcome the sample inefficiency and local optima stagnation inherent in evolutionary search methods. As illustrated in Figure 3, CoDSA functions as a plug-and-play search policy within the Ansor ecosystem. By reformulating the tensor program optimization as a trajectory-based search augmented with thermodynamic heuristics, CoDSA achieves a superior balance between exploration and exploitation under strict measurement budgets.

3.1. CoDSA Framework and Ansor Integration

The fundamental challenge in tensor compilation is identifying an optimal configuration vector x from a high-dimensional, discrete, and non-convex search space S, such that the execution time on hardware is minimized.

3.1.1. Problem Formalization

Let G denote a computational subgraph (a fused operator). Operating strictly within a template-free search paradigm, the Ansor framework automatically constructs a comprehensive search space by generating a set of high-level structural skeletons, termed sketches $\Sigma=\{s_1,s_2,...,s_m\}$, which define the structural skeleton of the computation such as multi-level tiling structures and cache hierarchies. For each sketch s_i, there exists a parameter space P_i consisting of tunable knobs such as tile sizes, unroll factors, and thread binding indices. A complete schedule is represented by a configuration vector $v \in P_i$. The optimization objective is defined as:

$$v^* = argmin_{v \in P_i} F_{cost}\left(Lower(s_i, v)\right), \quad (1)$$

where $Lower(\cdot)$ transforms the configuration into executable machine code, and $F_{cost}(\cdot)$ represents the measured latency, which is computationally expensive to obtain.

3.1.2. Integration with Ansor Pipeline

CoDSA is architected as a drop-in replacement for Ansor's evolutionary search module while preserving its overall tuning infrastructure: it takes the sketches generated by Ansor's policy rules, uses Ansor's XGBoost cost model to select the single most promising sketch as the starting point v_{init} rather than initializing a random population, and then performs iterative refinement via three tightly coupled mechanisms—Zoom (step control), Ortho (diversity), and Core (ac-

ceptance)—in contrast to Ansor's genetic operators such as crossover and mutation. Finally, the measured latencies are recorded through Ansor's standard logging/database interface and can optionally be used by the shared cost-model updater, thereby retaining the hardware-in-the-loop tuning workflow. The XGBoost cost model is used only to select the initial sketch and optionally guide candidate prioritization. The CoDSA-Core acceptance decision is made after the candidate has been lowered, built, and measured on the target hardware; it uses the measured latency samples rather than the predicted cost.

The overall workflow is formally described in Algorithm 1.

Algorithm 1 CoDSA Overall Optimization Loop

Input: Sketch Space Σ, Budget N_{trials}, Initial Temp T_0
Output: Optimized Schedule v_{best}
$v_{curr} \leftarrow$ SelectBestSketchZΣ, CostModel)
$v_{best} \leftarrow v_{curr}$
$T \leftarrow T_0$
$H \leftarrow \{v_{curr}\}$ {History for Ortho}
while $n < N_{trials}$ **do**
  {**Phase 1: CoDSA-Zoom**}
  $\Delta \leftarrow$ ComputeAdaptiveStep(n, β)
  for each dimension d in v_{curr} **do**
    {**Phase 2: CoDSA-Ortho**}
    $N \leftarrow$ GenerateNeighbors(v_{curr}, d, Δ)
    $v_{prop} \leftarrow$ FilterDiverse(N, H, θ_{ortho})
    if $v_{prop} \neq$ NULL **then**
      $lat_{prop} \leftarrow$ MeasureOnHardware(v_{prop})
      $n \leftarrow n + 1$
      {**Phase 3: CoDSA-Core**}
      $Accept \leftarrow$ MetropolisCheck(lat_{curr}, lat_{prop}, T)
      if $Accept$ **then**
        $v_{curr} \leftarrow v_{prop}$
        $lat_{curr} \leftarrow lat_{prop}$
        $H \leftarrow H \cup \{v_{curr}\}$
        if $lat_{curr} < lat_{best}$ **then**
          $v_{best} \leftarrow v_{curr}$
        end if
      end if
    else
      $n \leftarrow n + 1$ {Penalize budget to prevent livelock}
    end if
  end for
  $T \leftarrow$ UpdateTemperature(T, n)
end while
return v_{best}

3.2. CoDSA-Core: Dynamic Simulated Annealing

Standard greedy descent algorithms are mathematically incapable of surmounting the performance ridges ($\Delta E > 0$) often found in the non-convex landscapes of complex operators like Shifted Window Attention (Figure 1). To address this, CoDSA-Core incorporates a Dynamic Simulated Annealing mechanism that employs a thermodynamic criterion to probabilistically accept temporary performance degradations, effectively allowing the search agent to climb out of suboptimal basins.

3.2.1. Probabilistic Acceptance Criterion

Because the objective in Eq. (1) is latency minimization, a smaller latency is always preferred. Let v_k denote the current accepted schedule and $\tilde{v}_k$ denote a proposed schedule. After the proposed schedule is lowered and measured on the target hardware, its mean measured latency is denoted as $Lat(\tilde{v}_k)$. We define the latency degradation as:

$$\Delta_k = Lat(\hat{v}_k) - Lat(v_k), \tag{2}$$

Thus, $\Delta_k \leq 0$ means that the proposed schedule is no slower than the current schedule and is accepted deterministically. When $\Delta_k > 0$, the proposed schedule is temporarily worse but may still be accepted to escape local optima:

$$P(accept) = \begin{cases} 1 & if\ \Delta_k \leq 0 \\ exp\left(-\frac{\Delta_k}{T_k}\right) & if\ \Delta_k > 0 \end{cases}. \tag{3}$$

Here, T_k is the annealing temperature at iteration k. The previous Boltzmann-like constant is absorbed into the temperature scale, so no separate physical constant is required.

3.2.2. Dynamic Cooling Schedule

The temperature is initialized as T_0=1000 in our experiments. After each search iteration, the temperature is updated by multiplicative cooling:

$$T_{k+1} = \rho_k T_k, \tag{4}$$

where the initial cooling rate is ρ_0=0.95. For the dynamic cooling variant, the cooling rate is updated as:

$$\rho_{k+1} = max(0.85, 0.99\rho_k). \quad (5)$$

This schedule allows relatively permissive acceptance in early iterations and gradually suppresses the acceptance of latency-degrading candidates as the search converges.

3.3. CoDSA-Zoom: Adaptive Step Size Control

Traversing a large parameter space with tile sizes spanning 1 to 1024 using a fixed unitary step is computationally prohibitive, while large fixed steps risk overshooting narrow regions of hardware efficiency. To resolve this conflict, CoDSA-Zoom implements a coarse-to-fine strategy that dynamically modulates the search stride Δ_k, ensuring that sampling density matches the granularity of the optimization phase.

3.3.1. Adaptive Step Decay

We introduce an Adaptive Step Size Δ_k modulated by an exploration factor $\beta \in 0,1$:

$$\Delta_k = \max\left(1, \left\lfloor \Delta_{init} \cdot \beta^{\lfloor k/S \rfloor} \right\rfloor\right), \quad (6)$$

where S represents the epoch length.

- **Exploration Phase (Low *k*):** Δ_k is large. The algorithm performs long jumps across the coordinate axes, effectively sampling distinct regions of the search space.
- **Exploitation Phase (High *k*):** $\Delta_k \to 1$. The algorithm converges to a local tuner, refining parameters to match exact hardware constraints, such as aligning vector lengths to 32 for warp efficiency.

3.3.2. Multi-Scale Neighborhood

Combined with CoDSA-Core, Zoom creates a dynamic neighborhood definition. At high temperatures and large steps, the algorithm behaves like a random global search; as temperature and step size decrease, it seamlessly transitions into a greedy local hill-climber. We also enforce boundary constraints: if $v_d + \Delta_k$ exceeds the parameter's valid range, the step is clipped to the nearest valid boundary.

3.4. CoDSA-Ortho: Orthogonal Diversity Control

In high-dimensional spaces, single-trajectory methods are susceptible to mode collapse, where the search repeatedly evaluates functionally equivalent or colinear parameter configurations. To maximize the marginal information gain of each measurement trial, CoDSA-Ortho explicitly penalizes redundancy by enforcing orthogonality in the search direction.

Before computing cosine similarity, each schedule candidate is converted into a reduced knob-index vector. CoDSA-Ortho does not compare raw TVM schedule traces or heterogeneous schedule objects directly. Ansor first produces a valid sketch, and CoDSA keeps this sketch structure fixed while tuning only the numerical schedule knobs inside the trace. We extract two types of tunable knobs: split/tile knobs SP and unroll pragma knobs PR. For each knob i, a finite legal candidate set S_i is constructed. For SP knobs, S_i is selected from {2, 4, 8, 16, 24, 32, 48, 64} together with the original Ansor seed value, filtered by the loop extent. For PR knobs, S_i contains candidate values of auto unroll max step, such as {64, 128, 256, 512}, together with the original seed value when applicable.

A schedule configuration is then represented as $x = [x_1, x_2, \ldots, x_d]$, where x_i is the integer index of the selected value in S_i. Thus, heterogeneous schedule choices are mapped into a fixed-length integer vector over legal knob choices. Hierarchical and constrained dependencies are handled during candidate-set construction and schedule reconstruction: invalid values are excluded before vectorization, and each vector is converted back to a valid TVM schedule trace before measurement.

3.4.1. Feature Representation and Similarity

We map each configuration vector v to a normalized feature space.

To quantify redundancy, we utilize Cosine Similarity between a candidate proposal v_{prop} and the history set H:

$$Sim(v_{prop}, v_h) = \frac{v_{prop} \cdot v_h}{\|v_{prop}\| \, \|v_h\|}, \ \forall v_h \in H. \quad (7)$$

In our implementation, the orthogonality threshold is fixed to τ = 0.8 for all workloads. A candidate is re-

jected when its cosine similarity with any vector in the current comparison set is larger than τ. This value corresponds to an angular separation of approximately 36.9 degrees, which filters out nearly collinear schedule configurations while still allowing moderately similar candidates to be explored.

CoDSA-Ortho does not maintain an unbounded global history buffer. The comparison set is the batch-local candidate buffer generated during the current search step. The batch size is B = max (number of CPU cores, 16). In addition, an index-based visited set is maintained to avoid exact duplicate measurements. Therefore, the Ortho module controls local redundancy within the generated candidate batch without introducing a large memory-resident history archive.

3.4.2. Rejection Sampling with Orthogonality

Before performing a hardware measurement, CoDSA-Ortho checks the maximum similarity:

$$S_{max} = \max_{v_h \in H} \mathrm{Sim}\left(v_{prop}, v_h\right). \tag{8}$$

If $S_{max} > \theta_{ortho}$, the candidate is rejected regardless of its predicted cost. This Hard Rejection forces the Coordinate Descent algorithm to choose an alternative direction—specifically, one that is orthogonal to the current trajectory. This effectively prunes the search space, guiding the algorithm towards unexplored regions. This mechanism complements CoDSA-Zoom: while Zoom allows the algorithm to jump far, Ortho ensures those jumps land in novel territory.

4. Experiments

In this section, we present a systematic evaluation of the proposed CoDSA framework. We systematically analyze its performance against state-of-the-art auto-scheduling baselines across a diverse set of ViT architectures. Our experimental design is structured to rigorously validate three core hypotheses: (1) CoDSA significantly reduces search time and memory overhead compared to evolutionary search methods while maintaining or exceeding code quality; (2) The integration of Dynamic Simulated Annealing and Adaptive Step Size allows the search agent to escape local optima in the rugged, non-convex optimization landscapes typical of complex operators like Swin Transformers; and (3) Orthogonal Diversity Control effectively mitigates the mode collapse problem inherent in single-trajectory search methods, ensuring robust exploration under strict measurement budgets.

4.1. Setup

To ensure the reproducibility of our results and provide a fair comparison, we conducted all experiments on a high-performance heterogeneous computing platform designed to simulate realistic deployment scenarios. The host system is powered by an Intel Xeon Silver 4110 CPU, while the target accelerator is an NVIDIA GeForce RTX 2080 Ti GPU. Our software stack is built on Apache TVM (v0.13.x), utilizing CUDA 11.7 and cuDNN 8.5.0 as the backend. The environment runs on Ubuntu 20.04 LTS with Python 3.8.

For workload evaluation, we selected six representative Vision Transformer models: ViT (Small, Base) [8], DeiT (Small, Base) [24], and Swin Transformer (Tiny, Small) [17]. These models cover a broad spectrum of computational complexities and memory access patterns. Specifically, the Swin Transformer serves as a stress test due to its Shifted Window Attention mechanism, which creates a highly non-convex optimization landscape with complex dependency chains.

We compare CoDSA against a robust set of baselines, including template-based AutoTVM (Random and Grid Search), a standard Genetic Algorithm without a learned cost model, the Ansor framework [33], and the original greedy Droplet search [4]. Grid Search in this paper refers to a budgeted reduced-grid baseline rather than exhaustive enumeration of the full Ansor tensor scheduling space. The grid is constructed only from selected Ansor schedule knobs: split/tile parameters (SP) and the unroll pragma (auto_unroll_max_step, PR). For each SP knob, the candidate values are selected from {1, 2, 4, 8, 16, 24, 32, 48, 64} plus the original Ansor seed value, filtered by the corresponding loop extent. For each PR knob, the candidate values are {64, 128, 256, 512}. The baseline then enumerates the Cartesian product of these reduced knob dimensions under the same measurement budget. We use Grid Search to denote this reduced-grid baseline. Performance is evaluated based on three key metrics: total search time (including compila-

tion and measurement), peak host memory usage, and the final inference latency measured over up to 10,000 synchronized runs after a warm-up phase.

4.2. Main Results

4.2.1. End-to-End Performance and Efficiency Analysis

We first present a comprehensive comparison of search efficiency and model performance on the ViT-Base model, a widely used benchmark in the tensor compilation literature. Table 1 summarizes the quantitative results.

Table 1

Search efficiency and resource footprint on ViT-Base. We report total search time (compile+measure), final inference latency, and peak host RAM usage.

Method	Search (s)	Infer. (ms)	RAM (MiB)
Ansor (XGB)	950	17.38	1903
Random	1631	17.79	1190
Grid	1823	16.64	1334
Genetic (Ga)	2417	22.88	4070
Droplet	641	17.24	942
CoDSA (Ours)	**597**	**15.32**	**893**

Search Time Reduction: CoDSA demonstrates a measured reduction in tuning time, completing the search in 597 seconds compared to Ansor's 950 seconds—a reduction of 37.2%. This acceleration stems fundamentally from CoDSA's trajectory-based search nature. Unlike Ansor, which must maintain a population history, perform complex crossover/mutation operations, and repeatedly query a computationally expensive XGBoost cost model for thousands of candidates in every generation, CoDSA generates and evaluates candidates sequentially using lightweight coordinate transformations. The elimination of the heavy evolutionary meta-heuristics allows CoDSA to spend a larger fraction of the time budget on actual hardware measurements rather than search overhead.

Memory Footprint and Resource Footprint: The resource efficiency of CoDSA is even more pronounced. It requires only 893 MiB of memory, whereas Ansor consumes 1903 MiB—a substantial 53.1% reduction. Ansor's high memory usage is driven by the need to store feature vectors for the entire history of explored schedules to train its cost model. In contrast, CoDSA's search policy is lightweight (it does not maintain an evolutionary population), and the additional algorithmic state it introduces is small—primarily a short history buffer for the Orthogonal Diversity check (CoDSA-Ortho). For clarity, the reported peak memory footprint still includes Ansor's shared tuning infrastructure such as logging, the global schedule database, and cost-model feature extraction/updating, which is common across the compared methods. The lower memory footprint suggests that CoDSA is more compatible with memory-constrained tuning environments than Ansor under the evaluated setting.

Inference Latency and The Grid Search Trade-off: Crucially, this efficiency does not come at the cost of code quality. CoDSA achieves an inference latency of 15.32 ms, surpassing Ansor (17.38 ms) and the reduced-grid Grid Search baseline (16.64 ms). It is pertinent to discuss the trade-off presented by Grid Search. While Grid Search achieves a respectable latency of 16.64 ms, better than the random baseline, its search time (1823 s) is nearly triple that of CoDSA. Grid Search suffers from the curse of dimensionality; it effectively wastes the vast majority of its budget evaluating invalid or low-performance regions of the search space that are theoretically reachable but practically useless. CoDSA starting from a high-quality sketch (initialized via Ansor's policy) and performing guided descent, effectively navigates to the high-performance manifold that Grid Search hits only by brute force. This establishes CoDSA as a favorable trade-off, achieving better latency than the reduced-grid baseline while requiring only a fraction of its search cost.

4.2.2. Cross-Model Generalization and Robustness

The primary comparative advantage of CoDSA is unleashed in extremely rugged topological spaces. While traditional greedy search methods or standard evolutionary algorithms often suffice for the relatively smooth, continuous optimization landscapes of standard convolutions, they frequently may underperform when confronted with the irregular memory dependencies of Swin Transformers. Our cross-model evaluations demonstrate that CoD-

SA's orthogonal exploration offers significant advantages over standard methods in navigating these volatile architectures.

To verify that our gains are not specific to a single architecture, we extended the evaluation to all six benchmark models. Table 2 presents the inference latency comparison across the varied landscape of ViTs.

The results highlight a strong overall trend: CoDSA-Ortho performs best in most tested cases compared to the baseline Ansor framework and the original Droplet search across the evaluated workloads. While CoDSA-Ortho achieves the best latency on five out of the six models, its gains are marginal or slightly negative in relatively smooth optimization landscapes. For instance, on ViT-Small, Grid Search is marginally better by 0.01 ms (6.59 ms vs. 6.58 ms). This indicates that for simpler architectures with smaller parameter spaces, exhaustive enumeration within a reduced grid can sometimes identify the absolute global minimum that heuristic trajectory methods might narrowly miss, albeit at a significantly higher search time cost (as shown in Table 1).

Overall, the results indicate that orthogonal diversity enforcement improves robustness without sacrificing the exploitation efficiency of coordinate descent. The most significant and scientifically interesting gains are observed in the Swin Transformer models. For Swin-Small, CoDSA-Ortho reduces latency to 13.53 ms, representing a 26.5% speedup over Ansor's 18.41 ms. This result is important for illustrating the limitations of evolutionary search. The Swin Transformer's shifted window attention involves complex loop nests with conditional masking and cyclic shifting. These structures create a highly volatile performance landscape where a small mutation in tile size can lead to invalid memory accesses or severe bank conflicts.

Ansor's evolutionary mutation operators often break the delicate loop structure required for optimal tensor core alignment. In contrast, CoDSA's coordinate descent preserves the structural integrity of the high-quality initial sketch while fine-tuning the numerical parameters (tile sizes, unroll factors). Furthermore, the superior performance of CoDSA-Ortho over Droplet (15.80 ms) on Swin-Small confirms that the simple greedy approach is insufficient for these rugged landscapes; the barrier-crossing capability of Simulated Annealing and the diversity enforcement of Ortho are essential to escape the numerous local optima introduced by the window shifting mechanism.

4.3. Ablation

To rigorously quantify the contribution of each proposed component and dissect the source of our performance gains, we analyze the incremental improvements shown in table 2. We visualize the marginal benefit of each module in the context of the Swin-Small model, which represents the most complex optimization challenge.

4.3.1. Impact of CoDSA-Core (Dynamic Simulated Annealing)

The transition from Droplet to CoDSA-Core introduces the probabilistic acceptance criterion. On DeiT-Base, the Droplet performed worse than Ansor (16.61 ms vs 15.43 ms), indicating that the greedy

Table 2

Inference latency across ViT/DeiT/Swin models with CoDSA component ablations. CoDSA-Core/Zoom/Ortho progressively add dynamic SA, adaptive step control, and diversity enforcement (lower is better).

Model	Ansor	Droplet	CoDSA-Core	CoDSA-Zoom	CoDSA-Ortho	Grid
DeiT-Small	7.86	7.74	7.68	7.19	**7.11**	7.21
DeiT-Base	15.43	16.61	15.34	16.24	**14.72**	16.32
ViT-Small	8.49	8.21	7.74	6.82	6.59	**6.58**
ViT-Base	17.38	17.24	15.80	15.68	**15.32**	16.64
Swin-Tiny	10.37	8.37	8.74	8.58	**7.96**	8.82
Swin-Small	18.41	15.80	15.69	15.58	**13.53**	15.77

search became trapped in a local optimum—likely a configuration that was locally maximal but globally suboptimal due to memory bank conflicts or poor cache reuse. CoDSA-Core rectifies this, it improves the latency to 15.34 ms. By accepting temporary performance degradations (governed by the dynamic temperature T_k), the search agent was able to traverse the performance barrier surrounding the local basin and converge to a deeper minimum. Here, a performance degradation refers specifically to a candidate whose measured latency is higher than that of the current accepted schedule, i.e., $\Delta_k > 0$. This effectively validates the theoretical motivation that tensor schedule spaces are inherently non-convex and require non-greedy exploration strategies to find global optima.

4.3.2. Impact of CoDSA-Zoom (Adaptive Step Size)

The addition of CoDSA-Zoom yields the most prominent gains on smaller, more regular models like ViT-Small (7.74 ms → 6.82 ms) and DeiT-Small (7.68 ms → 7.19 ms). In these smaller search spaces, the optimal parameters often lie at the boundaries of the valid range defined by limits like the maximum vectorization length allowed by the hardware. A fixed step size of 1 makes traversing from a default value of 16 to a boundary such as 128 prohibitively slow, wasting the limited trial budget on intermediate, insignificant states. CoDSA-Zoom's adaptive stride allows the agent to make long jumps early in the search, rapidly identifying the promising region (macro-tuning), and then seamlessly decay the step size for precise optimization (micro-tuning). The result is a significantly more efficient utilization of the 10000-trial budget, ensuring that the final solution is finely tuned to the hardware intrinsics.

It is important to note that intermediate variants do not guarantee monotonic improvements across all models. As observed in Table 2, while CoDSA-Core improves DeiT-Base latency to 15.34 ms, the addition of the Zoom module (CoDSA-Zoom) temporarily regresses performance to 16.24 ms, underperforming Ansor (15.43 ms). This occurs because adaptive step sizing in isolated intermediate configurations can overshoot narrow high-performance regions if diversity constraints are not enforced. Stable and optimal performance across diverse workloads is only achieved when all three components (Core, Zoom, and Ortho) operate collectively.

4.3.3. Impact of CoDSA-Ortho (Orthogonal Diversity)

The final integration of CoDSA-Ortho provides the critical edge for large, complex models. On Swin-Small, it improves performance from 15.58 ms to 13.53 ms. For high-dimensional spaces, distinct parameter configurations can map to functionally identical machine code or exhibit high collinearity in performance characteristics. Without diversity control, the search agent may redundantly explore a subspace of similar schedules where, for instance, only the tile size of the innermost loop varies while the outer loops are neglected. CoDSA-Ortho enforces exploration in directions orthogonal to the history of visited states. This ensures that the search trajectory covers a wider volume of the design space. It is worth noting a minor observation: for the smallest model, DeiT-Small, CoDSA-Ortho demonstrated robust performance compared to Zoom (7.11 ms vs 7.19 ms). This suggests that even in small, convex spaces, enforcing diversity does not hinder the search from refining the global optimum, indicating the robustness of diversity threshold θ across varying search space complexities.

Although the diversity threshold in CoDSA-Ortho remains stable across the evaluated ViT, DeiT, and Swin workloads on the tested NVIDIA GPU backend, it should not be regarded as a universal hardware-independent constant. Its optimal value may vary across hardware targets because architectural factors can affect the relationship between schedule-space similarity and actual performance diversity. Therefore, for substantially different backends such as CPUs, mobile GPUs, FPGAs, TPUs, or NPUs, the threshold should be treated as a backend-level hyperparameter and calibrated using a small set of initial measurements. A threshold that is too small may over-prune useful candidates, whereas one that is too large may weaken the diversity constraint. Systematic cross-backend threshold adaptation is left for future work.

4.4. Sensitivity and Convergence Analysis

To rigorously quantify the temporal dynamics and efficiency of the proposed framework, we conduct a detailed analysis into the convergence behavior of the search process. This section examines the trade-off between measurement budget and solution quality, providing empirical evidence for the superior sample efficiency of CoDSA.

4.4.1. Confidence Score and Early Stopping

The stopping criterion for the search is governed by a statistical t-test with a confidence level $1-\alpha$. We systematically explored the trade-off between search cost and result quality by varying this confidence score on the ViT-Base model.

Table 3
Effect of early-stopping confidence level on searching time and final latency (ViT-Base).

Confidence Score (1-α)	Search (s)	Infer. (ms)
0.99	668	15.29
0.95	**597**	**15.32**
0.90	533	15.90
0.85	495	16.27

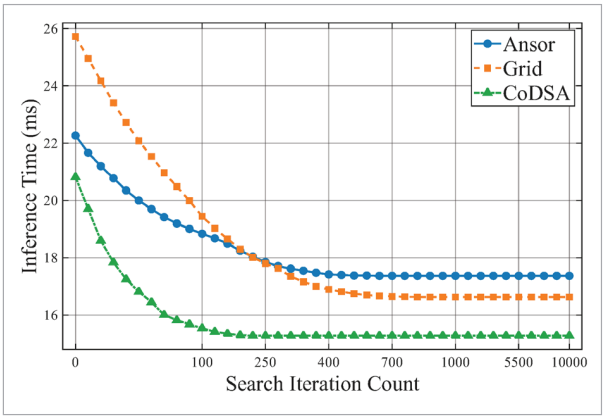
As shown in table 3, increasing the confidence score from 0.85 to 0.99 monotonically increases the search time (from 495s to 668s) as the algorithm requires more statistical evidence of convergence before termination. However, the improvement in inference latency exhibits diminishing returns. While raising the confidence from 0.85 to 0.95 yields a tangible improvement (16.27 ms → 15.32 ms), further increasing it to 0.99 results in a negligible performance gain of only 0.03 ms, yet incurs an approximately 12% penalty in search time. Consequently, we identify 0.95 as the optimal hyperparameter setting for our experiments, offering a highly favorable trade-off between rapid tuning turnaround and high-quality code generation.

4.4.2. Convergence Dynamics and Sample Efficiency

Beyond the static final metrics, the dynamic behavior of the search agent reveals the fundamental algorithmic advantages of CoDSA. Figure 4 visualizes the optimization trajectory, plotting the best-found inference latency against the number of hardware measurement trials. Three distinct optimization signatures are observable:

1. **Evolutionary Stagnation (Ansor):** The baseline Ansor curve begins with high latency (≈ 26 ms) and exhibits a slow, gradual decay. This confirms our hypothesis regarding the inefficiency of evolutionary search in high-dimensional spaces: the random initialization and stochastic mutation operators require extensive sampling to discover valid, high-performance subspaces. Even after 10000 iterations, Ansor fails to reach the performance floor, indicating that its warm-up phase consumes the majority of the budget.

Figure 4
Inference Latency vs. measurement trials on ViT-Base. CoDSA converges rapidly (within the first few hundred trials) and reaches lower latency than Ansor and grid search, demonstrating higher sample efficiency.



2. **Evolutionary Stagnation (Ansor):** The baseline Ansor curve begins with high latency (≈ 26 ms) and exhibits a slow, gradual decay. This confirms our hypothesis regarding the inefficiency of evolutionary search in high-dimensional spaces: the random initialization and stochastic mutation operators require extensive sampling to discover valid, high-performance subspaces. Even after 10000 iterations, Ansor fails to reach the performance floor, indicating that its warm-up phase consumes the majority of the budget.
3. **Rigid Baseline (Grid Search):** The Grid Search maintains a flat trajectory around 16.5 ms. While it avoids the poor initialization of Ansor, it lacks the plasticity tao fine-tune parameters, effectively acting as a performance lower bound that intelligent search must surpass.
4. **Trajectory-Based Descent (CoDSA):** Our method demonstrates a waterfall convergence profile. Starting from a high-quality sketch initialization, CoDSA leverages the *Adaptive Step Size* (CoDSA-Zoom) to perform coarse-grained jumps, precipitously reducing latency within the first 100-200 iterations. Subsequently, the *Dynamic Simulated Annealing* (CoDSA-Core) mechanism

prevents stagnation, allowing the search to settle into a deeper optimum (≈ 15.3 ms) significantly faster than the evolutionary baseline.

This empirical evidence supports the Droplet Expectation: high-performance schedules are indeed accessible via guided trajectory search from a valid seed, provided the search agent possesses the ability to traverse local barriers.

5. Conclusion

In this work, we presented CoDSA, a hybrid auto-tuning framework that reconciles the rapid convergence of coordinate descent with the global exploration capabilities required for modern deep learning workloads. By augmenting the Droplet Search algorithm with Dynamic Simulated Annealing, Adaptive Step Size, and Orthogonal Diversity Control, we successfully addressed the stagnation issues inherent in greedy search strategies while avoiding the prohibitive resource costs of evolutionary methods.

Our empirical evaluation indicates that CoDSA achieves a superior trade-off between search efficiency and target code quality compared to existing evolutionary baselines. Addressing the inherent concerns regarding algorithmic overhead, while the integration of CoDSA-Core, CoDSA-Zoom and CoDSA-Ortho introduces marginal implementation complexity compared to a simplistic greedy Droplet search, the operational returns provide measurable improvements in search efficiency and inference latency. Engineered as a seamless drop-in replacement for the existing Ansor evolutionary policy, CoDSA introduces integration changes for compiler engineers. This architectural integration yields concrete performance gains: a 26.5% reduction in inference latency on complex workloads like Swin-Small, and a 53.1% decrease in memory footprint compared to the Ansor baseline.

These results indicate that CoDSA improves the measured trade-off among search time, inference latency, and peak memory usage on the evaluated NVIDIA GPU platform. Despite these observed improvements, distinct limitations remain regarding the framework's adaptability to extremely irregular, continuously shifting tensor dimensions at runtime. Future work will pivot to extend this hybrid deterministic search strategy to support dynamic shape workloads and large combinatorial scheduling spaces of Large Language Models (LLMs). Adapting this framework to the LLM optimization landscape under strict edge-memory constraints represents an important direction for future research in tensor compilation.

References

1. Ahn, B. H., Pilligundla, P., Yazdanbakhsh, A., Esmaeilzadeh, H. Chameleon: Adaptive Code Optimization for Expedited Deep Neural Network Compilation. arXiv Preprint arXiv:2001.08743, 2020. https://doi.org/10.48550/arXiv.2001.08743
2. Bi, J., Guo, Q., Li, X., Zhao, Y., Wen, Y., Guo, Y., Zhou, E., Hu, X., Du, Z., Li, L., Chen, H., Chen, T. Heron: Automatically Constrained High-Performance Library Generation for Deep Learning Accelerators. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2023, 314-328. https://doi.org/10.1145/3582016.3582061
3. Cai, X., Yang, Z., Fan, Z., Zhang, Q. Decomposition-Based-Sorting and Angle-Based-Selection for Evolutionary Multiobjective and Many-Objective Optimization. IEEE Transactions on Cybernetics, 2017, 47(9), 2824-2837. https://doi.org/10.1109/TCYB.2016.2586191
4. Canesche, M., Rosário, V., Borin, E., Quintão Pereira, F. The Droplet Search Algorithm for Kernel Scheduling. ACM Transactions on Architecture and Code Optimization, 2024, 21(2), 1-28. https://doi.org/10.1145/3650109
5. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E. cuDNN: Efficient Primitives for Deep Learning. arXiv Preprint arXiv:1410.0759, 2014. https://doi.org/10.48550/arXiv.1410.0759
6. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al. TVM: An Automated End-to-End Optimizing Com-

piler for Deep Learning. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, 578-594. https://www.usenix.org/conference/osdi18/presentation/chen

7. Chen, T., Zheng, L., Yan, E., Jiang, Z., Moreau, T., Ceze, L., Guestrin, C., Krishnamurthy, A. Learning to Optimize Tensor Programs. Advances in Neural Information Processing Systems, 2018, 31. https://doi.org/10.48550/arXiv.1805.08166
8. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale. International Conference on Learning Representations, 2021. https://openreview.net/forum?id=YicbFdNTTy
9. Geman, S., Geman, D. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1984, 6, 721-741. https://doi.org/10.1109/TPAMI.1984.4767596
10. Haj-Ali, A., Genc, H., Huang, Q., Moses, W., Wawrzynek, J., Asanović, K., Stoica, I. ProtoTuner: Tuning Programs with Monte Carlo Tree Search. arXiv Preprint arXiv:2005.13685, 2020. https://doi.org/10.48550/arXiv.2005.13685
11. Ingber, L. Very Fast Simulated Re-Annealing. Mathematical and Computer Modelling, 1989, 12(8), 967-973. https://doi.org/10.1016/0895-7177(89)90202-1
12. Ingber, L. Adaptive Simulated Annealing (ASA): Lessons Learned. Control and Cybernetics, 1996, 25(1), 33-54. http://control.ibspan.waw.pl:3000/contents/show/105?year=1996
13. Ingber, L. Simulated Annealing: Practice Versus Theory. Mathematical and Computer Modelling, 1993, 18(11), 29-57. https://doi.org/10.1016/0895-7177(93)90204-C
14. Kifetew, F. M., Panichella, A., De Lucia, A., Oliveto, R., Tonella, P. Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation. Proceedings of the 2013 International Symposium on Software Testing and Analysis, 2013, 257-267. https://doi.org/10.1145/2483760.2483789
15. Kirkpatrick, S., Gelatt Jr, C. D., Vecchi, M. P. Optimization by Simulated Annealing. Science, 1983, 220(4598), 671-680. https://doi.org/10.1126/science.220.4598.671
16. Kulesza, A., Taskar, B. Determinantal Point Processes for Machine Learning. Foundations and Trends in Machine Learning, 2012, 5(2-3), 123-286. https://doi.org/10.1561/2200000044
17. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, 10012-10022. https://doi.org/10.1109/ICCV48922.2021.00986
18. Ma, L., Xie, Z., Yang, Z., Xue, J., Miao, Y., Cui, W., Hu, W., Yang, F., Zhang, L., Zhou, L. Rammer: Enabling Holistic Deep Learning Compiler Optimizations With rTasks. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, 881-897. https://www.usenix.org/conference/osdi20/presentation/ma
19. Mouret, J. B., Clune, J. Illuminating Search Spaces by Mapping Elites. arXiv Preprint arXiv:1504.04909, 2015. https://doi.org/10.48550/arXiv.1504.04909
20. Nesterov, Y. Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems. SIAM Journal on Optimization, 2012, 22(2), 341-362. https://doi.org/10.1137/100802001
21. Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., Amarasinghe, S. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. ACM SIGPLAN Notices, 2013, 48(6), 519-530. https://doi.org/10.1145/2491956.2462176
22. Shao, J., Zhou, X., Feng, S., Hou, B., Lai, R., Jin, H., Lin, W., Masuda, M., Yu, C. H., Chen, T. Tensor Program Optimization with Probabilistic Programs. Advances in Neural Information Processing Systems, 2022, 35, 35783-35796. https://doi.org/10.52202/068431-2593
23. Shi, Y., Yang, Z., Xue, J., Ma, L., Xia, Y., Miao, Z., Guo, Y., Yang, F., Zhou, L. Welder: Scheduling Deep Learning Memory Access via Tile-Graph. 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), 2023, 701-718. https://www.usenix.org/conference/osdi23/presentation/shi
24. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H. Training Data-Efficient Image Transformers & Distillation Through Attention. International Conference on Machine Learning, 2021, 10347-10357. https://proceedings.mlr.press/v139/touvron21a.html

25. Tseng, P. Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization. Journal of Optimization Theory and Applications, 2001, 109, 475-494. https://doi.org/10.1023/A:1017501703105

26. Van Delm, J., Vandersteegen, M., Burrello, A., Sarda, G. M., Conti, F., Pagliari, D. J., Benini, L., Verhelst, M. HTVM: Efficient Neural Network Deployment on Heterogeneous TinyML Platforms. 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023, 1-6. https://doi.org/10.1109/DAC56929.2023.10247664

27. Vasilache, N., Zinenko, O., Theodoridis, T., Goyal, P., DeVito, Z., Moses, W. S., Verdoolaege, S., Adams, A., Cohen, A. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. arXiv Preprint arXiv:1802.04730, 2018. https://doi.org/10.48550/arXiv.1802.04730

28. Wright, S. J. Coordinate Descent Algorithms. Mathematical Programming, 2015, 151(1), 3-34. https://doi.org/10.1007/s10107-015-0892-3

29. Xing, J., Wang, L., Zhang, S., Chen, J., Chen, A., Zhu, Y. Bolt: Bridging the Gap Between Auto-Tuners and Hardware-Native Performance. Proceedings of Machine Learning and Systems, 2022, 4, 204-216. https://proceedings.mlsys.org/paper_files/paper/2022/hash/1f8053a67ec8e0b57455713cefdd8218-Abstract.html

30. Yarmohamadi, H., Kabudian, J., Mirhosseini, S. A New Dynamic Simulated Annealing Algorithm for Global Optimization. Journal of Mathematical and Computer Science, 2015, 14, 16-23. https://doi.org/10.22436/jmcs.014.01.02

31. Zhai, Y., Zhang, Y., Liu, S., Chu, X., Peng, J., Ji, J., Zhang, Y. TLP: A Deep Learning-Based Cost Model for Tensor Program Tuning. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2023, 833-845. https://doi.org/10.1145/3575693.3575737

32. Zhang, G., Zhou, S., Duan, Z., Zhou, W. FPGA Accelerator for CNN: An Exploration of the Kernel Structured Sparsity and Hybrid Arithmetic Computation. Journal of Electronic Imaging, 2021, 30(3), 033034. https://doi.org/10.1117/1.JEI.30.3.033034

33. Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., Gonzalez, J. E., Stoica, I. Ansor: Generating High-Performance Tensor Programs for Deep Learning. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, 863-879. https://www.usenix.org/conference/osdi20/presentation/zheng

34. Zheng, L., Liu, R., Shao, J., Chen, T., Gonzalez, J. E., Stoica, I., Ali, A. H. TenSet: A Large-Scale Program Performance Dataset for Learned Tensor Compilers. Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/a684eceee76fc522773286a895bc8436-Paper-round1.pdf

35. Zheng, S., Liang, Y., Wang, S., Chen, R., Sheng, K. Flextensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, 859-873. https://doi.org/10.1145/3373376.3378508

36. Zhu, H., Wu, R., Diao, Y., Ke, S., Li, H., Zhang, C., Xue, J., Ma, L., Xia, Y., Cui, W., Yang, F., Yang, M., Zhou, L., Cidon, A., Pekhimenko, G. Roller: Fast and Efficient Tensor Compilation for Deep Learning. 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022, 233-248. https://www.usenix.org/conference/osdi22/presentation/zhu