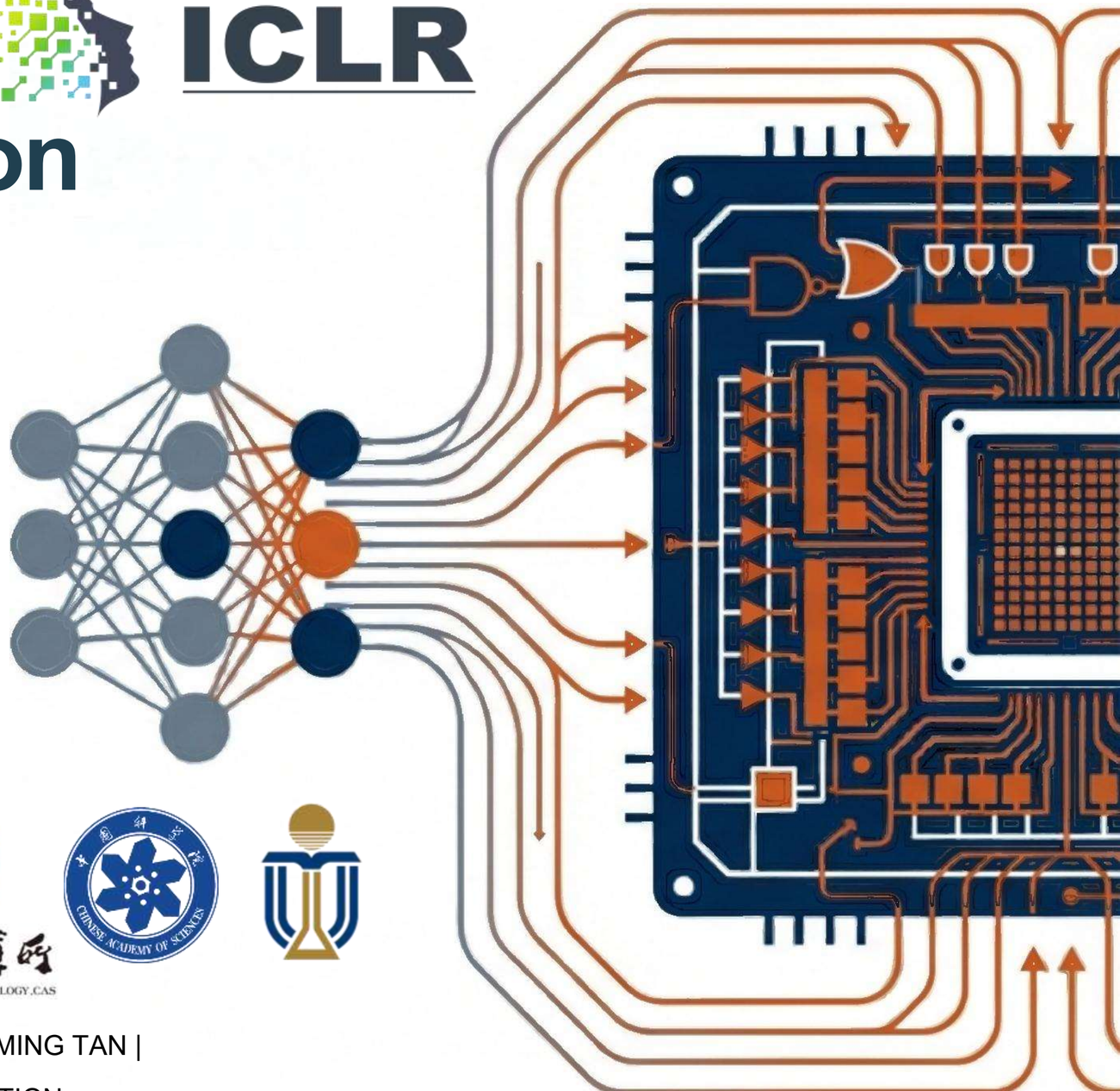
SparseRL:Mastering Sparse CUDA Generation Through Pretrained Models and Deep Reinforcement Learning

ICLR



Bridging the gap between LLM code generation and Hardware-Aware High-Performance Computing

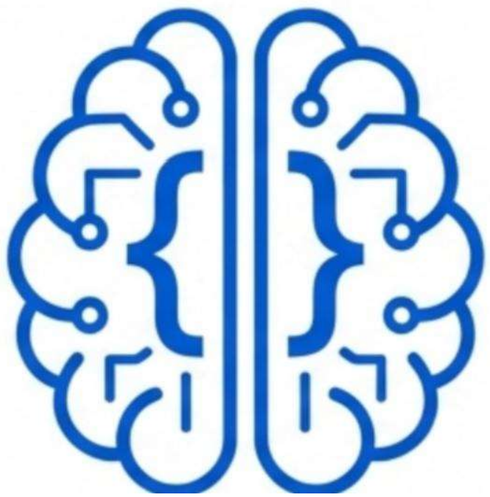
Presentor: YAOYU WANG

Authors: YAOYU WANG HANKUN DAI, ZHIDONG YANG, JUNMIN XIAO, GUANGMING TAN |

INSTITUTE OF COMPUTING TECHNOLOGY,CAS | ICLR 2026 ORAL PRESENTATION

The Performance Gap

Standard LLM Coding



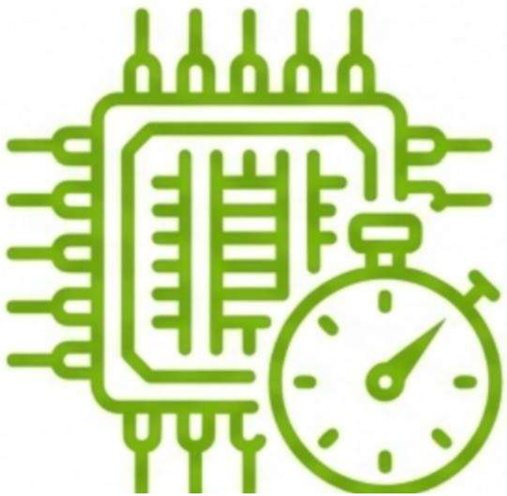
Focus: Logical Correctness

Metric: Pass@k

Blind to Hardware

VS

HPC / Sparse Coding



Focus: Execution Latency

Metric: GFLOPS

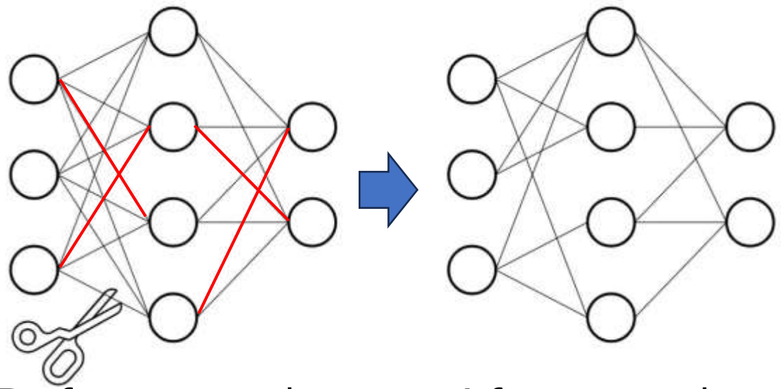
Hardware-Aware

In High-Performance Computing, the code not only needs to be correct, but also run faster.

Wide range of applications of sparsity

Sparse Weight

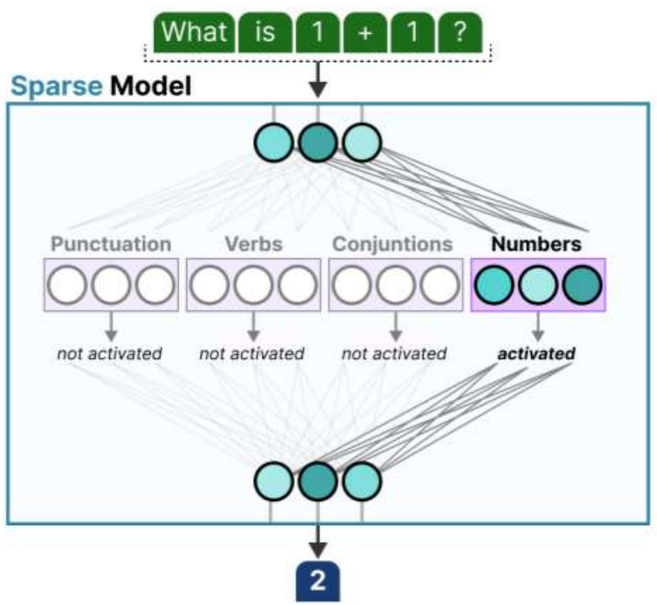
Sparsification (pruning) can set the values of some network weights to zero while maintaining the accuracy of the output.



Before pruning After pruning

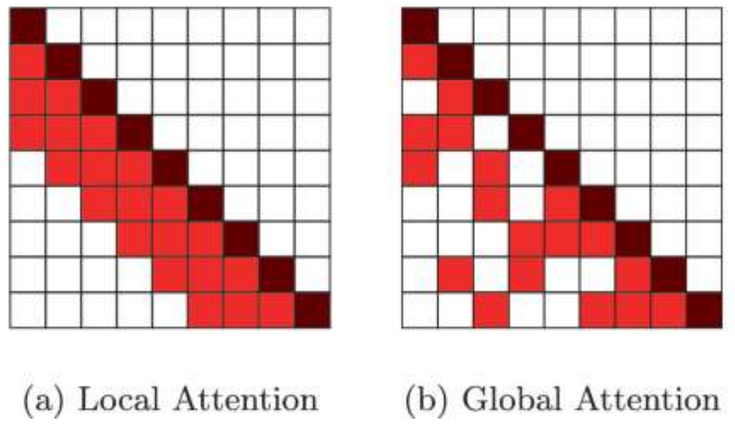
Sparse MOE

Activating only a portion of the linear layers reduces computational load. This involves an imbalance between communication and computation.



Sparse Attention

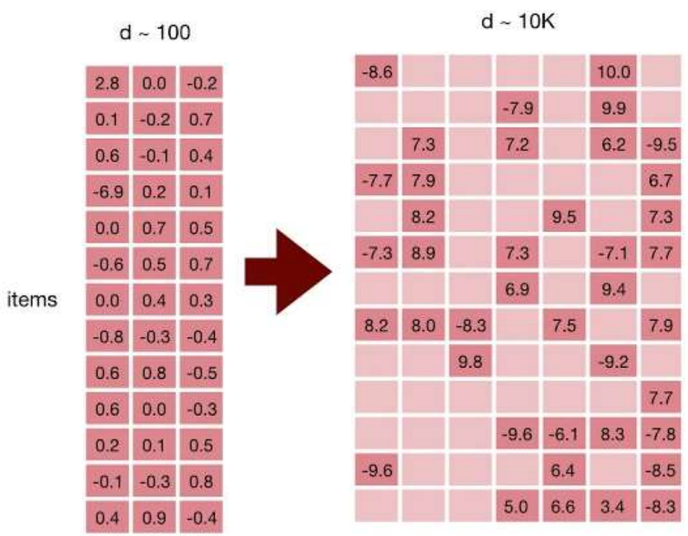
Attention mechanism is only calculated between a subset of tokens.



(a) Local Attention (b) Global Attention

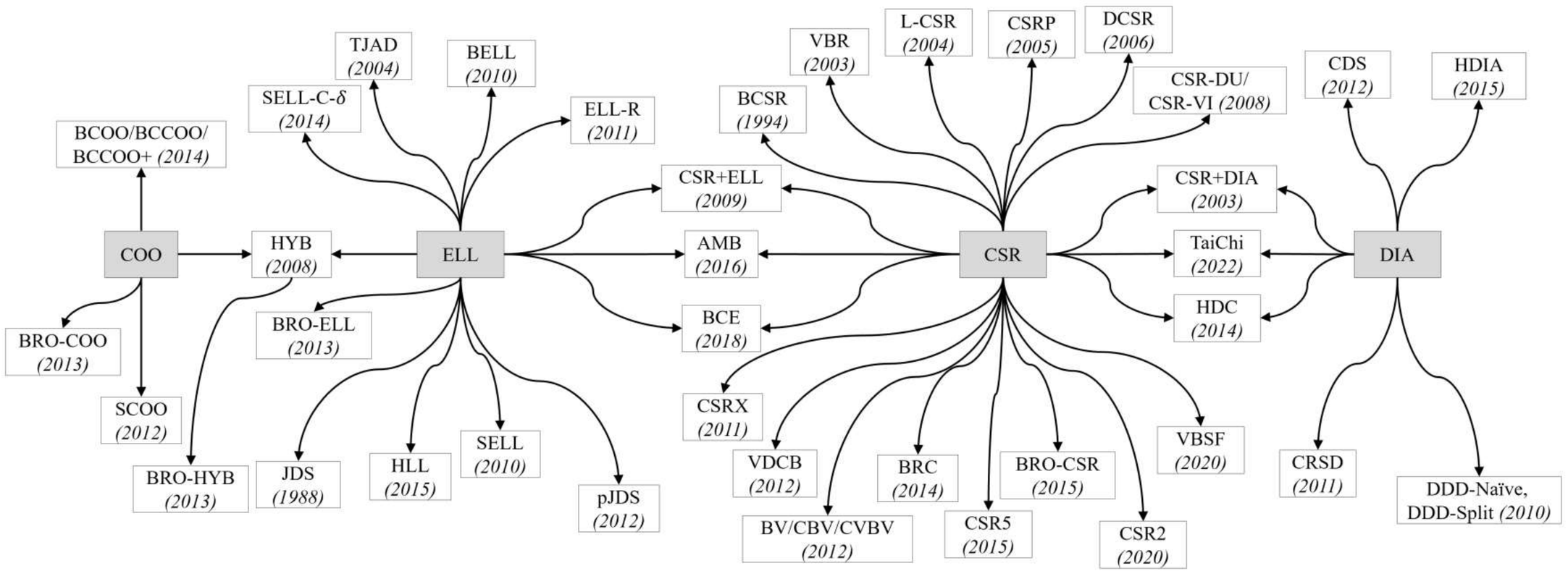
Sparse Embedding

Sparse embedding



Related work of HPC(High-Performance Computing)

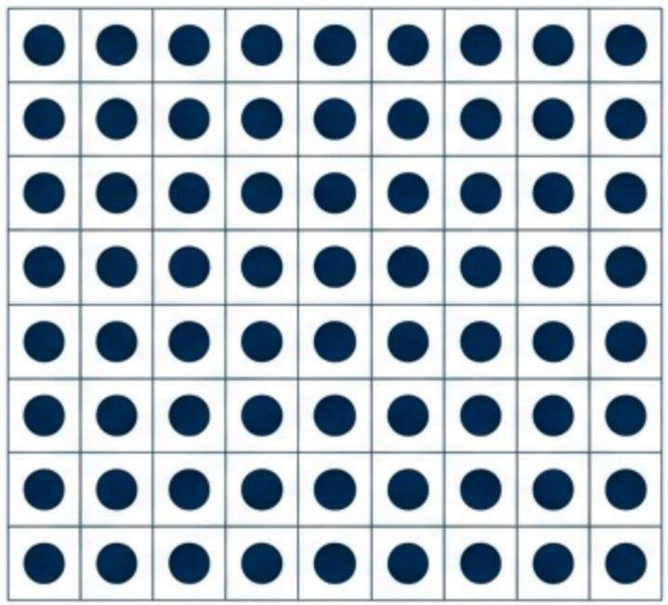
Numerous Studies and Large Time Spans



Gao J, Liu B, Ji W, et al. A systematic literature survey of sparse matrix-vector multiplication[J]. arXiv preprint arXiv:2404.06047, 2024.

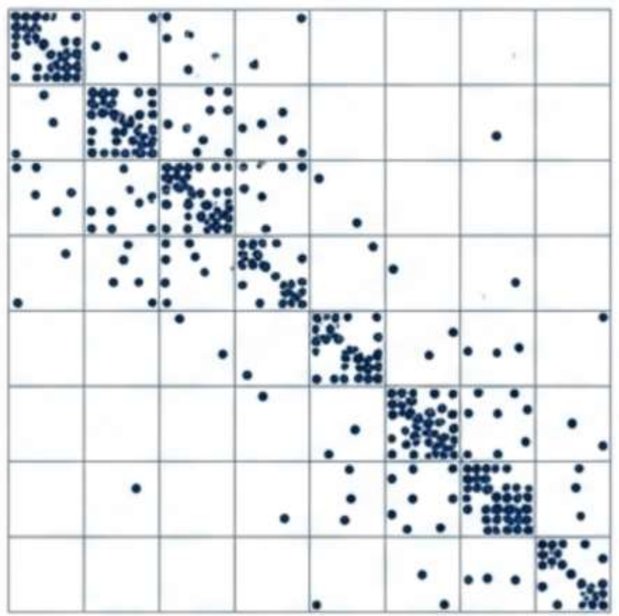
The Reason of Performance Gap In Sparse Computing

Dense Matrix (Static)



Regular Pattern. Memory access is predictable. Static code optimization works perfectly.

Sparse Matrix (Dynamic)



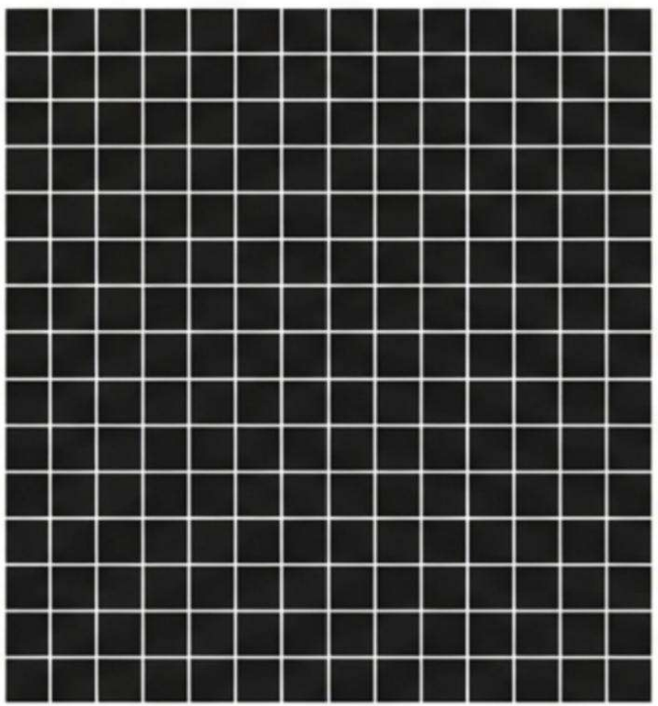
Irregular Pattern. Memory access is random. Requires input-aware dynamic optimization.

Standard LLMs (Next-Token Prediction) fail here because they ignore the geometric structure of the data, leading to code that is valid but slow.

SpMV: The Boss Fight for Code Gen

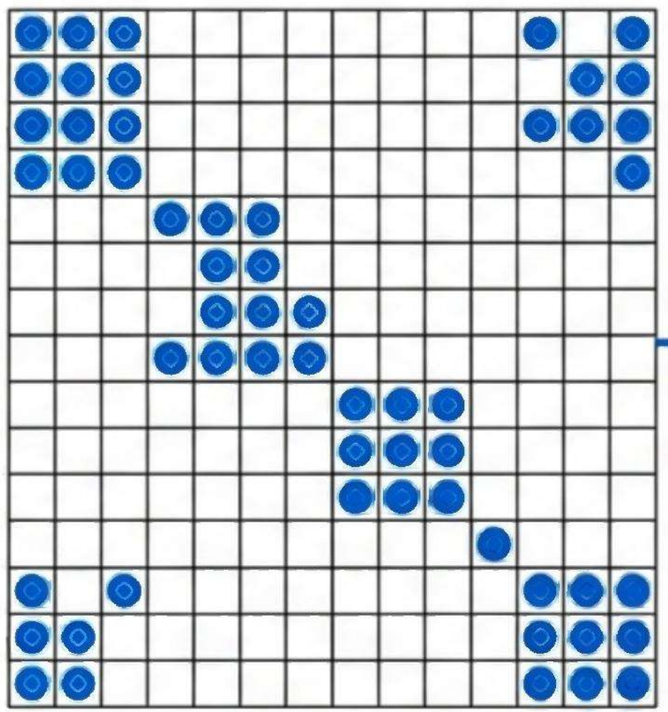
Sparse Matrix-Vector Multiplication requires dynamic adaptation.

Dense Matrix



Static &Predictable

Sparse Matrix



Irregular &Dynamic

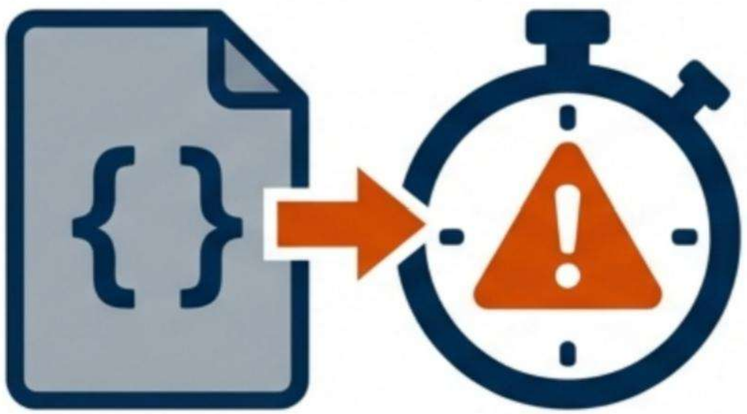
```
for(int j=ptr[row];
        j<ptr[row+1];j++)
```

Runtime dependency:
Code execution path
depends on data shape.

LLMs generate code.SparseRL generates optimized execution.

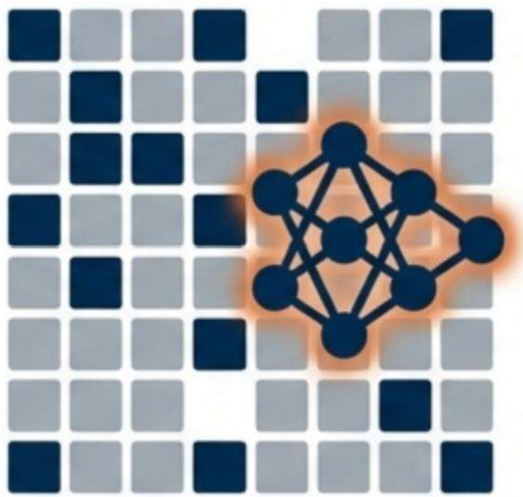
Executive Summary: The SparseRL Advantage.

The Challenge.



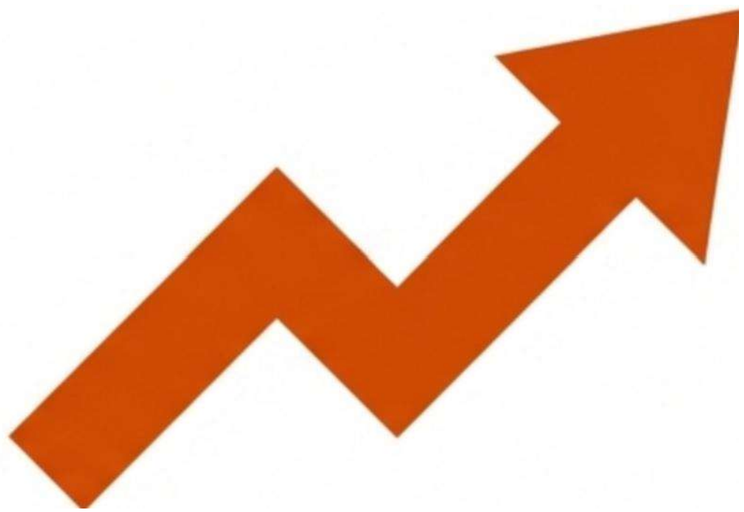
LLMs treat code as text, ignoring hardware reality. Sparse data irregularity causes standard kernels to suffer from load imbalance and memory divergence.

The Solution.



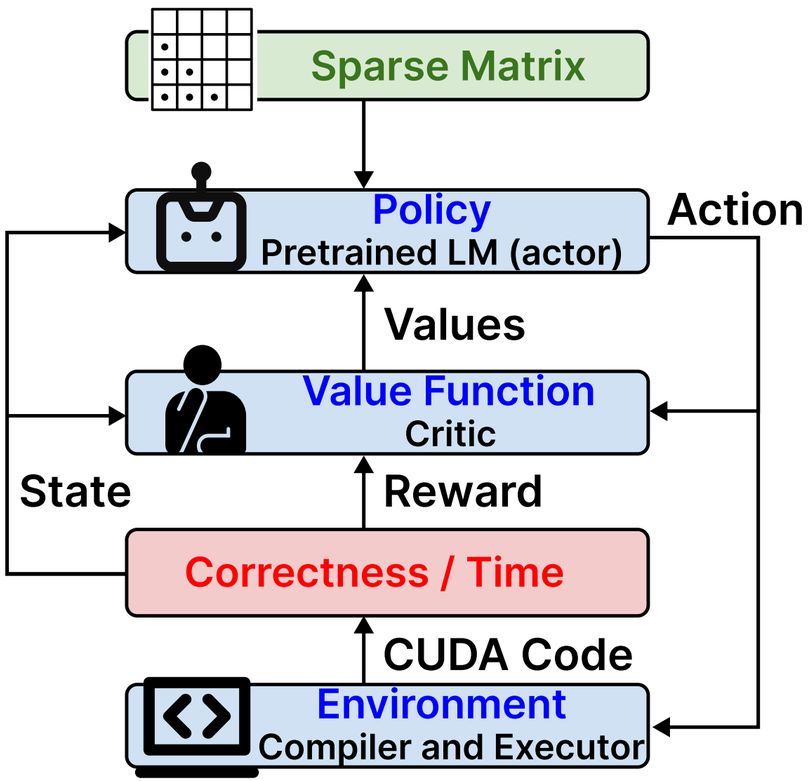
SparseRL treats Matrix Structure as a modality. It combines Sinusoidal Embeddings to 'see' sparsity with a Hierarchical RL Reward system to optimize for execution speed.

The Impact.



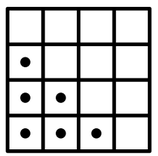
Outperforms NVIDIA cuSPARSE by 1.4x. Beats SOTA code generators By 30%. Improves compilation rates by 20%.

The SparseRL Framework and Target



Sparse Matrix

Row	1	2	2	3	3	3
Col	0	0	1	0	1	2



Dynamic input at runtime

Solution Program1

```
1. __global__ void spmv_kernel1(...)
2. {
3.   int row = blockDim.x * blockIdx.x + threadIdx.x;
4.   float cumu_sum = 0;
5.   for (int j = csr_ptr[row]; j < csr_ptr[row+1]; j++){
6.   cumu_sum += value[j] * x[col_index[j];];
7.   }
8.   y[row_current] = cumu_sum;
9. }
```

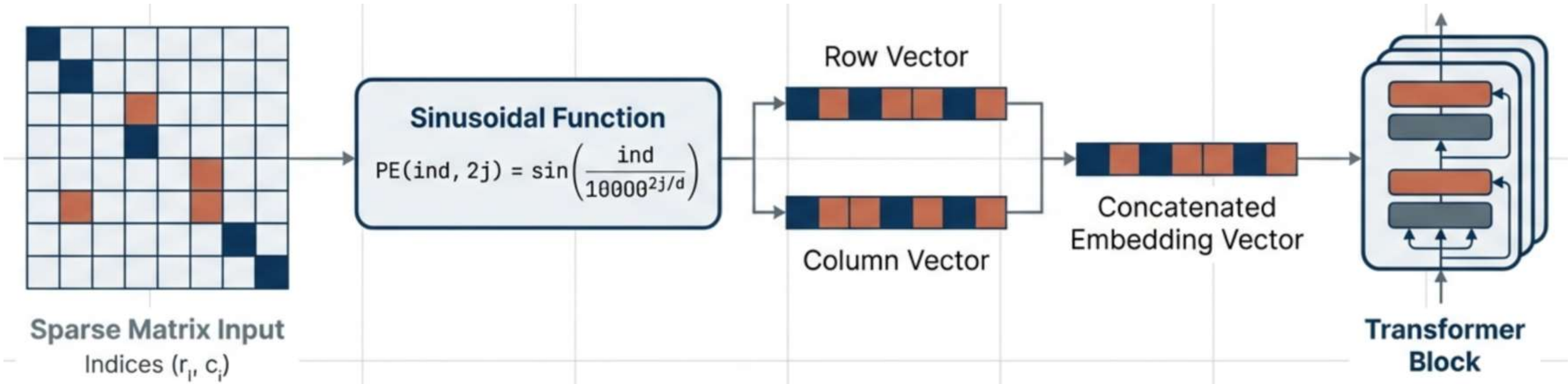
Solution Program2

```
1. __global__ void spmv_kernel2(...)
2. {
3.    int THREADS_PER_ROW = 2;
4.    int rows_per_block = blockDim.x / THREADS_PER_ROW;
5.    int row = blockIdx.x * rows_per_block
6.                         + threadIdx.x / THREADS_PER_ROW;
7.    int offset = threadIdx.x % THREADS_PER_ROW;
8.    float cumu_sum = 0;
9.    for (int i = csr_ptr[row] + offset; i < csr_ptr[row + 1]; ...) {
10.   •••
```

Input: Sparse Matrix → Generate → Output: Solution Program1, Solution Program2 → Execute → Execution Time/Efficiency: 21.95 s, 10.62 s ✓

Innovation 1:Treating Matrix Structure as a Modality

How do we feed a geometric shape into a Language Model?



Scale Invariant: Handles millions of rows without numerical instability.

Locality Aware: Captures relative positions (diagonal vs. scattered).

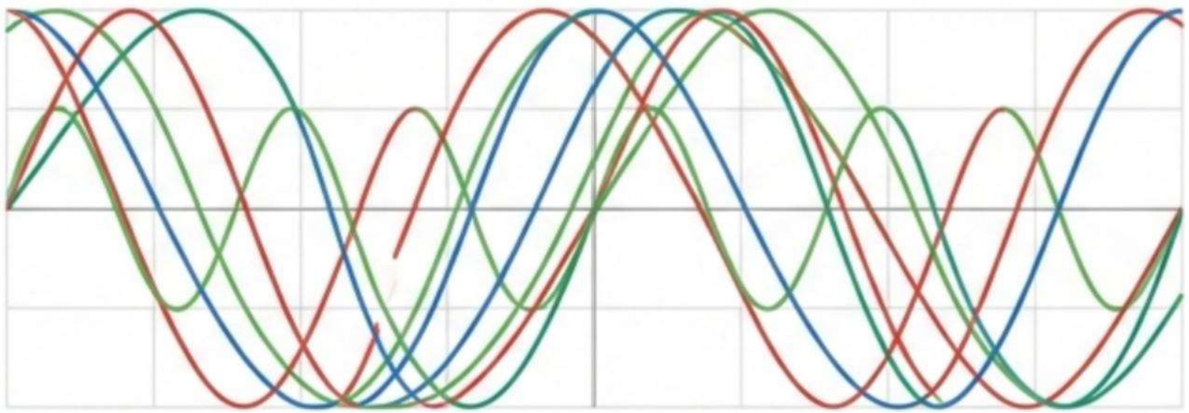
Innovation 1: Treating Matrix Structure as a Modality

From raw indices to geometric embeddings.

Rows:
[0, 1, 5. . .]

Cols:
[2, 4, 8. ⋯.]

Sinusoidal Embedding[1]



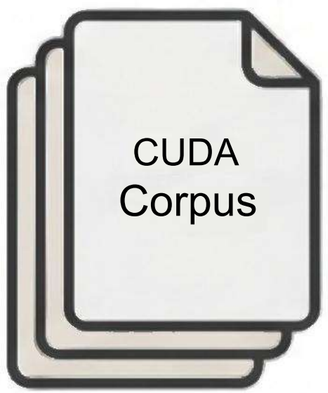
$$PE(ind, 2j) = \sin(ind/10000^{2j/d_{model}})$$

Why? Enables the model to 'see' clusters and diagonals for memory coalescing.

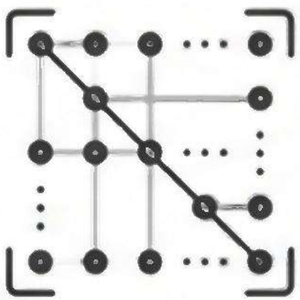
[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ŁukaszKaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processingsystems, 30, 2017

Innovation 2:The Training Pipeline

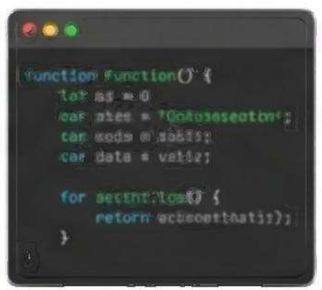
Pre-training



Supervised Fine-Tuning (SFT)



Sparse Matrix



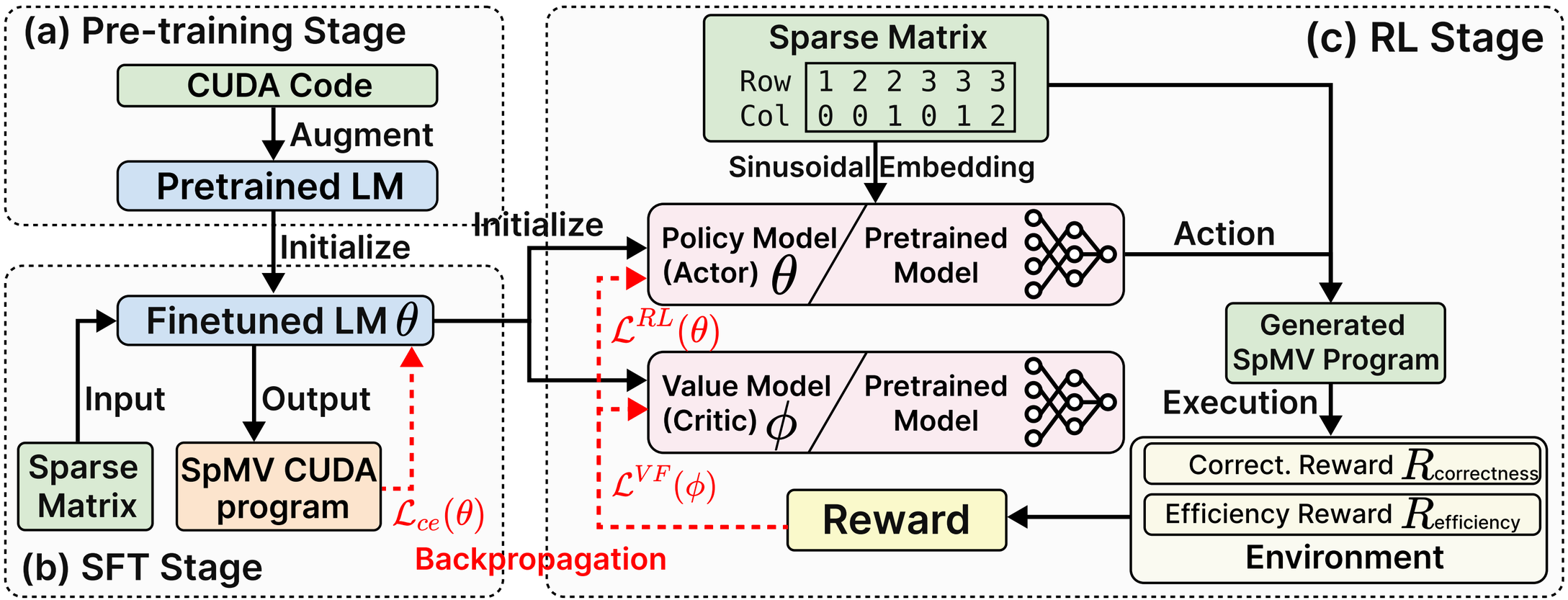
Code Output

Modality Alignment

Reinforcement Learning(RL)

Critical Differentiator

Innovation 2: The Training Pipeline Detail



Innovation 3: Designing the Reward Function: The Hierarchy of Needs

Efficiency(Refficiency)

Speedup vs.Baseline (cuSPARSE)

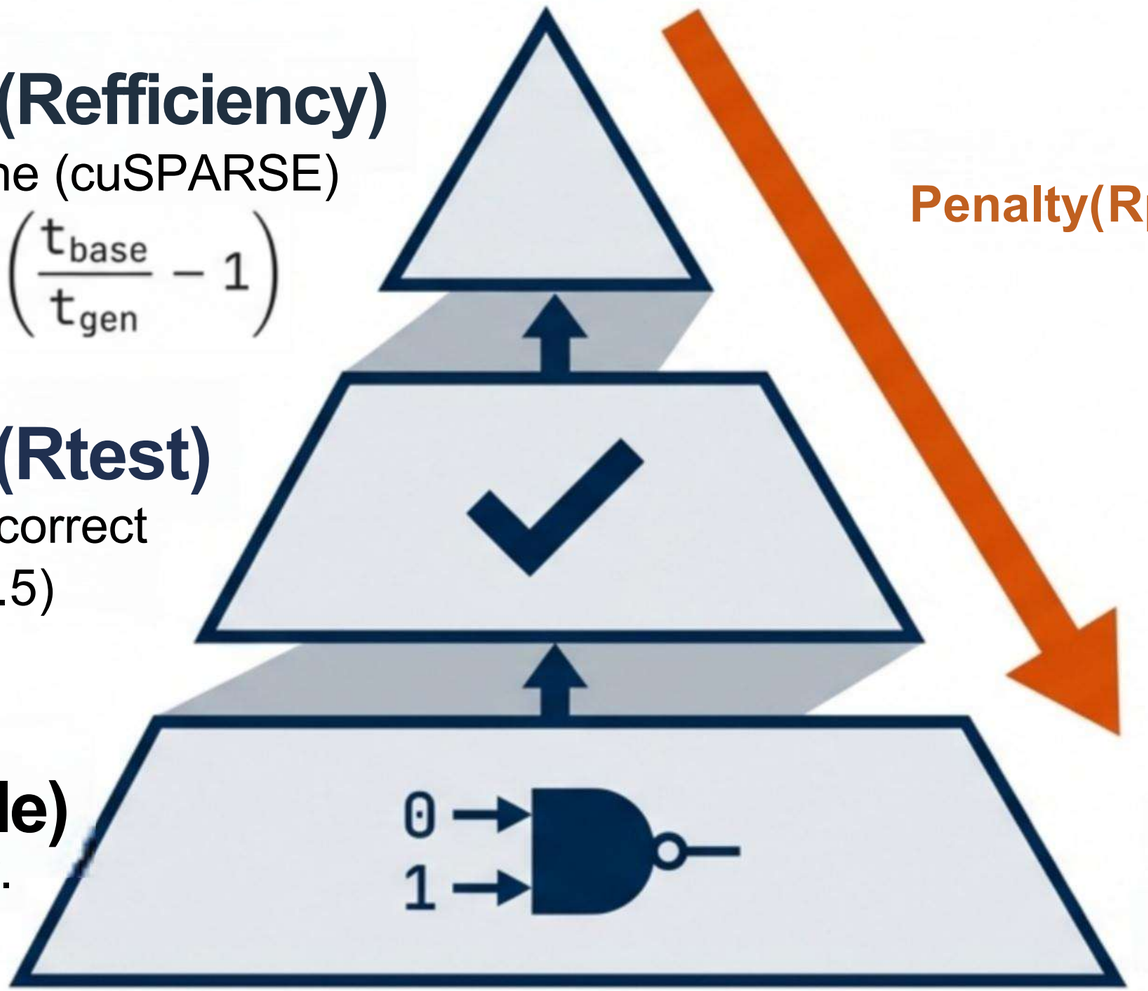
$$r_{eff} \times \left(\frac{t_{base}}{t_{gen}} - 1 \right)$$

Correctness(Rtest)

Must produce correct math(Ax=y).(+/-0.5)

Compilation(Rcompile)

Must compile successfully. (+/−0. 5)

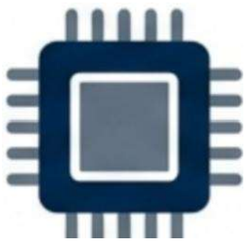


Penalty(Rpenalty)

Excessive Shared Memory usage (>48KB) is penalized.

Experimental Setup & Baselines

The Testbed



Hardware: NVIDIA V100 & A100 GPUs (Volta &Ampere)

Datasets:

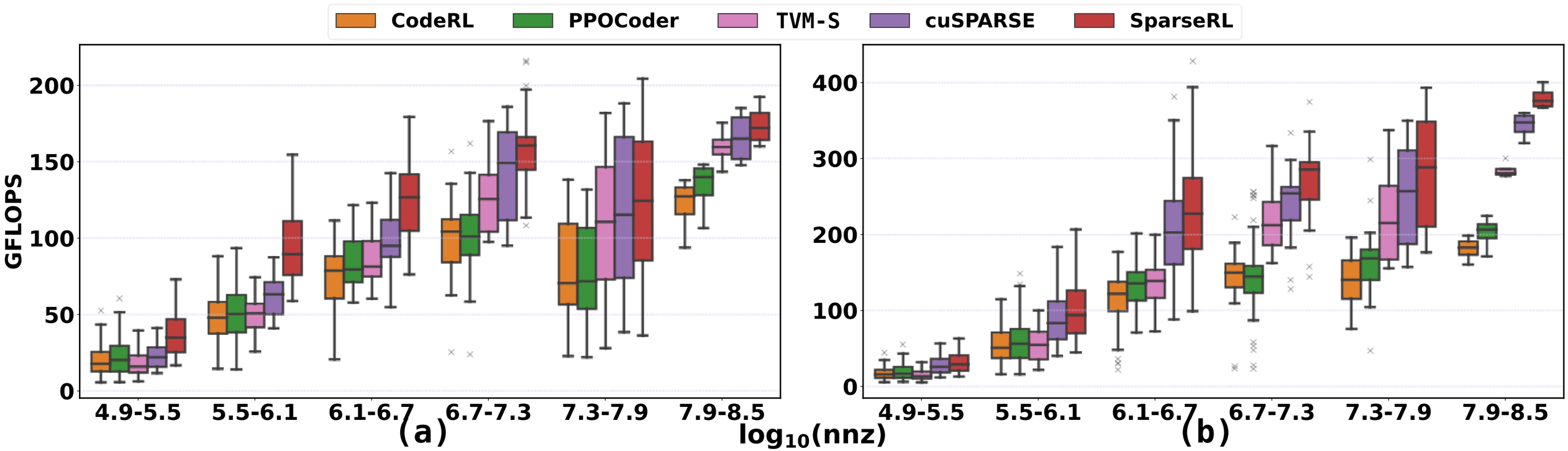
- SuiteSparse(1,100 matrices, diverse domains)
- DLMC (Deep Learning Matrix Collection)

The Competition

- **LLM Baselines:** CodeRL, PPOCoder (SOTA Code Gen)
- **Vendor Baselines:** cuSPARSE (NVIDIA Proprietary Library)
- **Auto-Tuners:** TVM-S (Sparse Compiler)

Primary Metric: GFLOPS (Giga Floating Point Operations Per Second)

Main Result: Beating Vendor Libraries (SpMV)

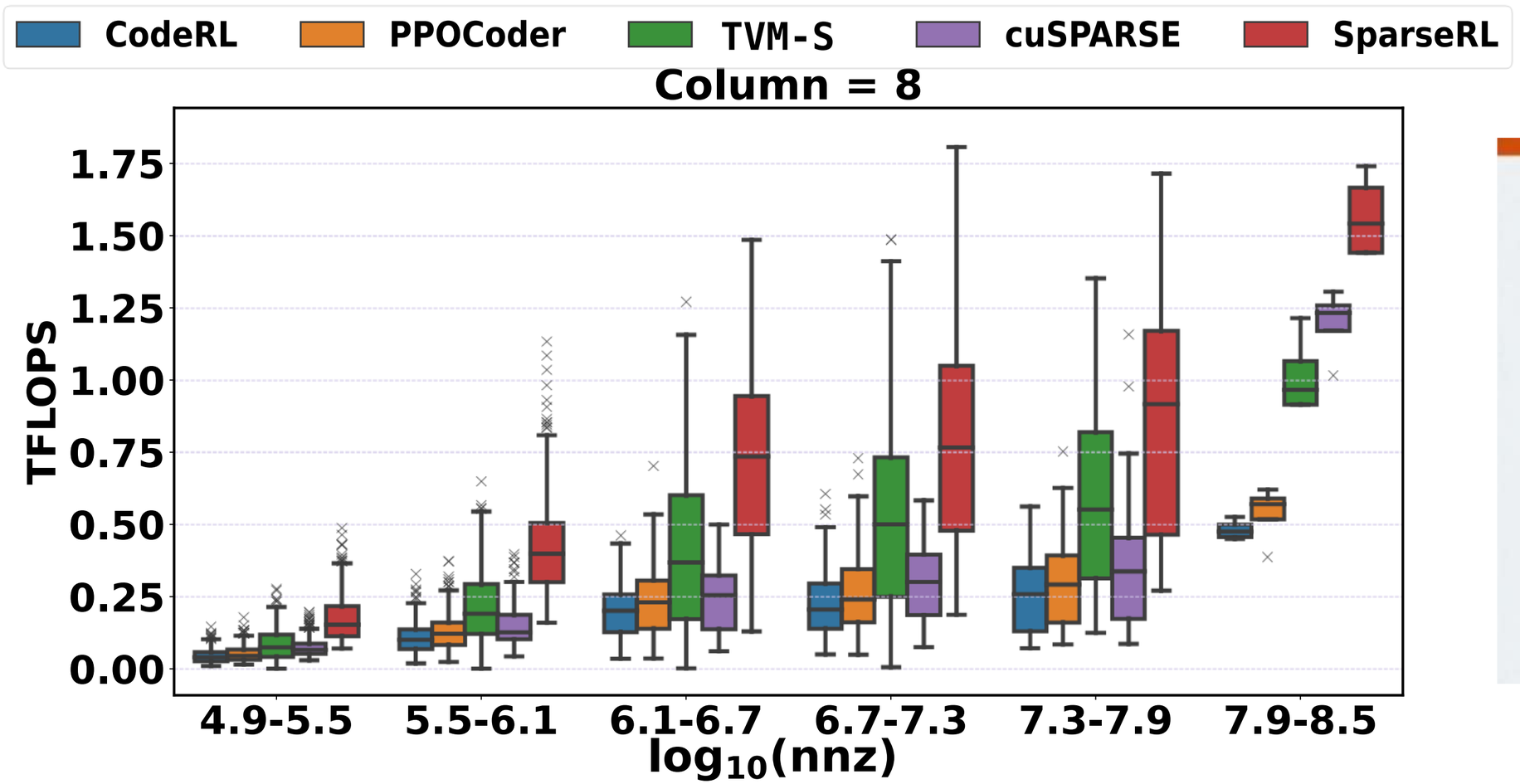


NVIDIA V100

NVIDIA A100

Generalization:Extending Dominance to SpMM

Sparse Matrix-Dense Matrix Multiplication (Critical for GNNs)



Key Insights

- Adapts to higher arithmetic intensity.
- 6.39x speedup vs. cuSPARSE(Col=8).
- 2.32x speedup vs. Sputnik (SOTA SpMM).

Correct functionality and Compilation Rates

Table 1: Correct functionality ($pass@k$) and Compilation Rates (CR) under $k = 1000$.

Model	Size	SpMV				SpMM (col=8)				SpMM (col=32)			
		$pass@1$	$pass@5$	$pass@1000$	CR	$pass@1$	$pass@5$	$pass@1000$	CR	$pass@1$	$pass@5$	$pass@1000$	CR
Qwen3	8B	8.00	12.50	–	–	6.75	12.00	–	–	7.00	10.75	–	–
DeepSeek-R1	671B	15.00	22.50	–	–	16.50	20.50	–	–	15.50	22.75	–	–
DeepSeek-R1-Distill -Qwen	7B	9.50	16.50	–	–	10.00	16.75	–	–	10.50	16.25	–	–
CodeT5	770M	4.75	7.50	30.25	38.00	1.75	2.25	28.00	36.00	2.00	2.25	30.00	36.25
CodeRL+CodeT5	770M	5.25	8.50	36.50	39.50	2.50	5.25	30.50	40.50	2.50	5.50	32.50	39.75
PPOCoder+CodeT5	770M	5.75	10.00	35.50	40.75	3.50	5.00	35.50	45.50	3.50	4.75	36.00	46.75
GPT-o3-pro	-	25.25	32.75	-	-	24.25	31.50	-	-	25.00	32.00	-	-
GPT-o4-mini	-	23.50	30.00	-	-	22.00	28.25	-	-	21.25	27.25	-	-
GPT-5	-	27.00	36.50	-	-	29.75	32.25	-	-	26.50	31.75	-	-
Claude-sonnet-4	-	28.25	36.75	-	-	24.25	31.50	-	-	26.00	31.50	-	-
SparseRL+CodeT5	770M	9.25	15.75	48.75	56.50	9.00	15.00	45.25	56.75	8.25	14.75	47.00	57.25
SparseRL+Qwen2.5	7B	9.75	16.00	48.75	57.00	10.00	15.00	46.50	57.00	8.50	15.00	46.75	58.00
SparseRL+Qwen2.5	14B	10.25	16.50	49.25	57.50	10.25	16.00	47.50	58.75	9.25	15.25	47.75	59.00
SparseRL+Qwen3	14B	16.25	21.00	–	–	18.25	21.00	–	–	18.25	22.25	–	–
SparseRL+GLM-Z1	9B	15.50	19.75	–	–	16.50	20.00	–	–	16.00	21.00	–	–
SparseRL+DeepSeek -Coder	6.7B	17.75	20.25	–	–	17.50	21.00	–	–	18.75	22.25	–	–

Under the Hood:What optimizations did RL discover?

Analysis of RL-driven code improvements in CUDA kernels.

Standard LLM Code

```
__global__ void standard_spmm_kernel(int* row_ptr, int* col_idx,
float* values, float* dense_matrix, float* output, int num_rows,
int dense_cols) {

  int row = blockIdx.x * blockDim.x + threadIdx.x;
  int col = blockIdx.y * blockDim.y + threadIdx.y;

  if (row < num_rows && col < dense_cols) {
     float sum = 0.0f;
     int start_idx = row_ptr[row];
     int end_idx = row_ptr[row + 1];

     for (int i = start_idx; i < end_idx; ++i) {
        int c = col_idx[i];
        float val = values[i];
        sum += val * dense_matrix[c * dense_cols + col];
     }
     output[row * dense_cols + col] = sum;
  }
}
```

SparseRL Optimized Code

```
__global__ void sparserl_spmm_kernel(int* row_ptr, int* col_idx,
float* values, float* dense_matrix, float* output, int num_rows,
int dense_cols) {
  int warp_id = threadIdx.x / 32;
  int lane_id = threadIdx.x % 32;
  int row = blockIdx.x * blockDim.y + warp_id;

  if (row < num_rows) {
    int start_idx = row_ptr[row];
    int end_idx = row_ptr[row + 1];
    int nnz = end_idx - start_idx;

    // Warp Padding and Coalescing Optimizations
    #pragma unroll
    for (int i = start_idx + lane_id; i < end_idx
        + (32 - (nnz % 32)) % 32; i += 32) {
      if (i < end_idx) {
        int c = col_idx[i];
        float val = values[i];
        // ... computation ...
      }
    }
    // ... output ...
}
```

Warp Padding

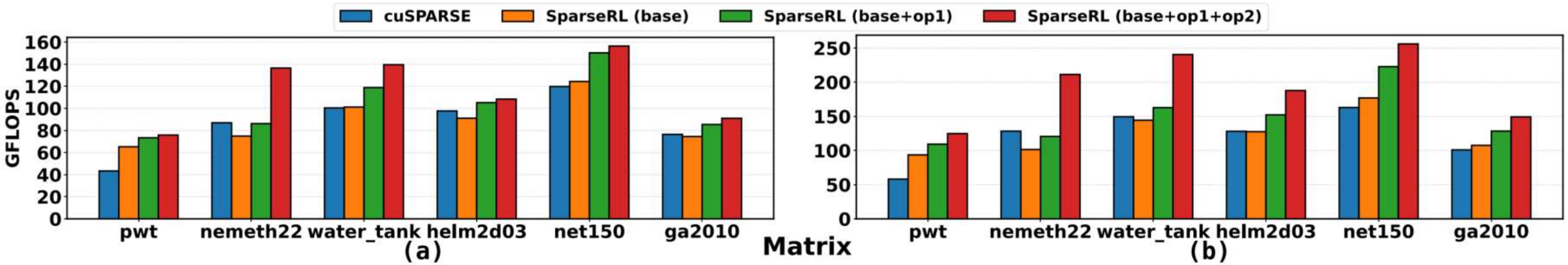
Learned to pad rows to prevent thread divergence.

Memory Coalescing

Reduced global memory access by 20-50%.

SM Occupancy increased from 71% to **89%**.

Ablation Study: Deconstructing the Gains



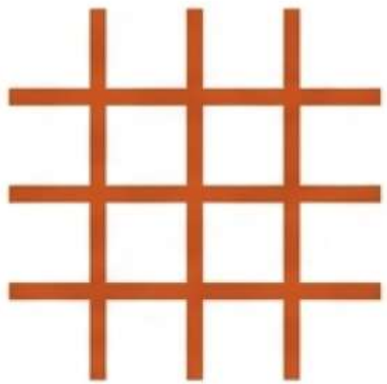
Ablation Studies on (a) V100 and (b) A100.

- (i) cuSPARSE, which is the NVIDIA library for sparse matrix operations;
- (ii) SparseRL (base) means the supervised fine-tuning (SFT);
- (iii) Op1 represents the utilization of RL with only the correctness reward;
- (iv) Op2 represents the addition of execution efficiency reward.

Limitations & Future Work

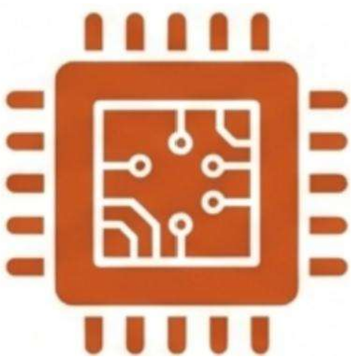
Training Cost

RL loop is expensive due to compilation/execution overhead.Best suited for reusable kernels (e.g., Simulation/AI Inference).



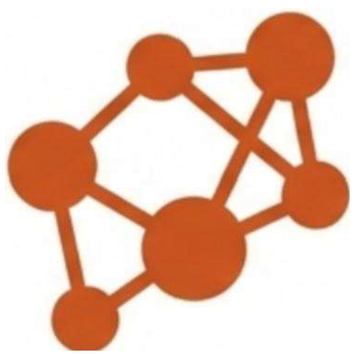
Extreme Sparsity

Struggles with 'Hyper-sparse' matrices(>99.9%zeros)where manual heuristics still win.



Hardware Scope

Currently CUDA-focused. Future work:HIP(AMD)and OpenCL.



Input Modality

Plan to integrate Graph Neural Networks(GNNs)for richer structural embedding.

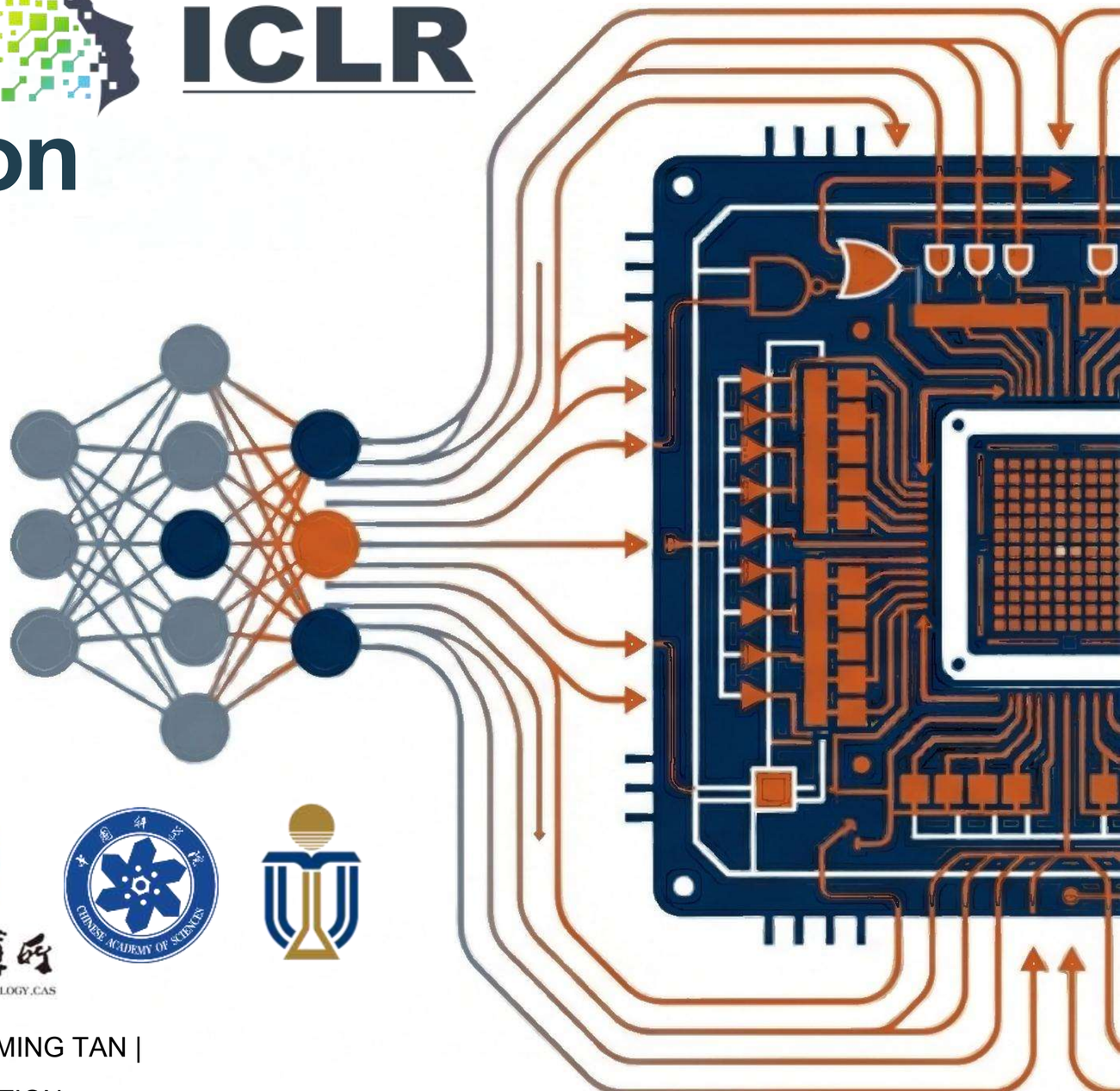
From Code Generation to Code Optimization

SparseRL bridges the gap between AI reasoning and Hardware reality.

- Embeds Matrix Structure as input.
- Uses Compiler-Feedback RL to optimize for speed.
- Beats Vendor Libraries by 1.4x.
- We provide the minimal code pipeline.

Code available: https://github.com/QiWu-NCIC/SparseRL

SparseRL:Mastering Sparse CUDA Generation Through Pretrained Models and Deep Reinforcement Learning



Thanks | Question & Answer

Presentor: YAOYU WANG

Authors: YAOYU WANG HANKUN DAI, ZHIDONG YANG, JUNMIN XIAO, GUANGMING TAN |

INSTITUTE OF COMPUTING TECHNOLOGY,CAS | ICLR 2026 ORAL PRESENTATION

Question & Answer

Question: Dataset-related issues

- How was the dataset collected?
- Are you considering open-sourcing it?

Answer:

- Regarding dataset collection:

We have long been deeply involved in the field of sparse computing, publishing papers on manual optimization of sparse operators since 2012, and accumulated over ten years of research experience in this area and have simultaneously built a continuous collection of sparse operator datasets.

- Regarding open-source plans:

We will open-source the datasets in the form of a Benchmark competition ("QiWu" project).

Question & Answer

Qusetion: Comparison with related work

- For example, comparison with CUDA-Agent, kevin, etc.

Answer:

- **SparseRL** is a customization expert in the field of sparse operators;
- **Kevin** is an iterator focused on learning the logic of human "code modification";
- **CUDA Agent** is an all-around agent with operator analysis and fusion capabilities supported by large-scale data.

Question & Answer

Qusetion: Method Details

- What framework/method was used for training/fine-tuning?

Answer:

- In the initial stages of training and fine-tuning:

  we used a hand-written Python framework for reinforcement learning training, specifically choosing PPO and MCTS algorithms as the core training methods. For distributed training, we initially adopted the lightweight torch.distributed solution.

- As the research progressed:

  we gradually expanded to mainstream framework systems, introducing Ray Rllib.

Question & Answer

Qusetion: More

Answer:
    [TODO]