# Accurate and Efficient CUDA Generation through Reinforcement Learning

Ao Li, Kexin Li, Weizhe Chen

ECE1508 Reinforcement Learning - Fall 2025

# Problem & Motivation: The CUDA Gap

**Problem**: GPU kernel implementation is crucial for optimizing ML performance, and **CUDA** is a powerful but low-level programming model that is **specialized, error-prone and time consuming** to implement (i.e., engineers must manage memory hierarchy, warp divergence, performance tuning at hardware level, etc.).

**Solution**: High-level compiler-based abstraction allows writing GPU code in a Python-like language instead of CUDA C/C++ (e.g., **Triton**).

**Problem**: Compiler-based GPU programming does not abstract away substantial domain expertise.

| | CUDA | TRITON |
|---|---|---|
| Memory Coalescing | Manual | Automatic |
| Shared Memory Management | Manual | Automatic |
| Scheduling (Within SMs) | Manual | Automatic |
| Scheduling (Across SMs) | Manual | Manual |

# Problem with LLM Kernel Generation

**Problem**: Compiler-based GPU programming does not abstract away substantial domain expertise.

**Solution**: Large Language Model (LLM) for GPU kernel generation.

**Problem**: Inefficient and Inaccurate automatic Triton code generation.

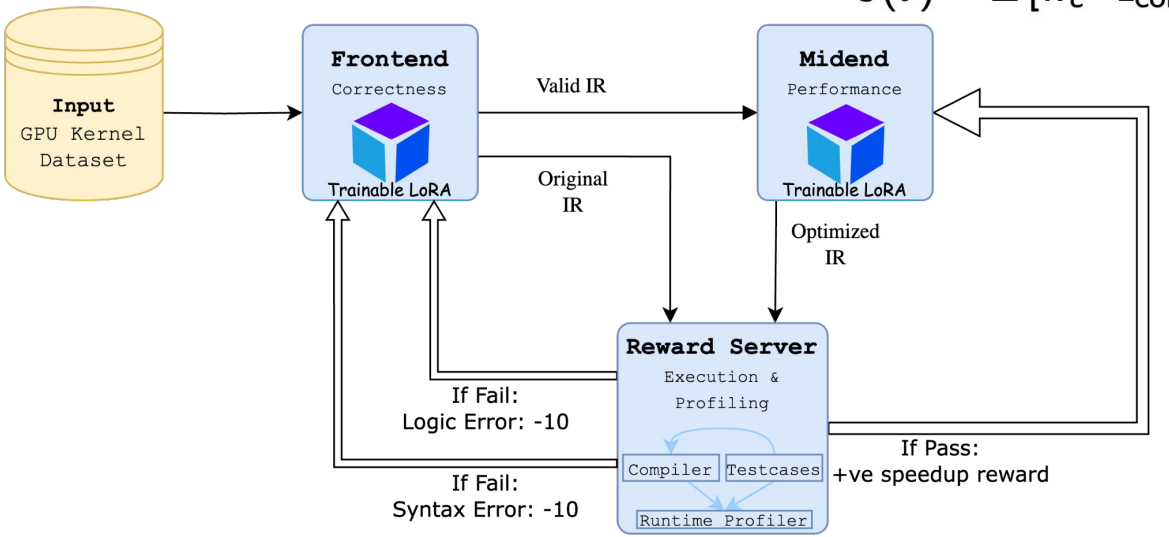
**Solution**: **Supervised Finetuning (SFT) + Reinforcement Learning (RL)**

- **SFT** teaches LLM the correct kernel patterns with high-quality code samples.
- **RL** uses CUDA compiler's runtime feedback (e.g., compilation errors, performance metrics) to refine kernel and hyperparameters (e.g., threads per block, block partitioning sizes, shared memory allocation).

# System Design: Multi-Agent Architecture

Our pipeline mimics a compiler stack by decoupling **functional correctness** (Frontend) from **runtime optimization** (Mid-end), employing Group Relative Preference Optimization (GRPO) for stable RL agent training.

$$J(\theta) = \mathbb{E}\left[w_c \cdot \mathbb{I}_{\text{compile}} + w_f \cdot \mathbb{I}_{\text{correct}} + w_p \cdot \text{Speedup}\right]$$



# Evaluation Results

Kernel Bench: Benchmark designed to evaluate t ability of LLMs to generate efficient GPU kernels

- Target to Level 1 of Kernel Bench: Simple Kernels

```python
import torch
import torch.nn as nn

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()

    def forward(self, predictions, targets):
        return torch.mean((predictions - targets) ** 2)
```

1.5x speed up

```python
import torch
import torch.nn as nn
import triton
import triton.language as tl
@triton.jit
def mse_loss_kernel(predictions_ptr, targets_ptr, output_ptr, n_elements, BLOCK_SIZE:tl.constexpr
):
    block_start = tl.program_id(0) * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
    predictions = tl.load(predictions_ptr + offsets, mask=mask, other=0.0)
    targets = tl.load(targets_ptr + offsets, mask=mask, other=0.0)
    diff = predictions - targets
    squared_diff = diff * diff
    tl.store(output_ptr + offsets, squared_diff, mask=mask)

def triton_mse_loss(predictions: torch.Tensor, targets: torch.Tensor):
    predictions = predictions.contiguous()
    targets = targets.contiguous()
    n_elements = predictions.numel()
    output = torch.empty_like(predictions)
    BLOCK_SIZE = 1024
    grid = lambda meta: ((n_elements + meta["BLOCK_SIZE"] - 1) // meta["BLOCK_SIZE"],)
    mse_loss_kernel[grid](predictions, targets, output, n_elements, BLOCK_SIZE=BLOCK_SIZE)
    return torch.mean(output)

class ModelNew(nn.Module):
    def __init__(self):
        super(ModelNew, self).__init__()

    def forward(self, predictions, targets):
        return triton_mse_loss(predictions, targets)
```

We observed that some kernels failed to compile in the latest version of PyTorch, as it introduced a major, non–backward-compatible API change earlier this year, affecting many of the available data samples.

**Future work**: finetune the reward function and LLM prompt

```python
import torch
from torch._inductor.select_algorithm import extern_kernels
import triton
import triton.language as tl
from torch._inductor.runtime.triton_heuristics import grid❌
from torch._C import _cuda_getCurrentRawStream as get_raw_stream
import torch.nn as nn
```

Ouyang, Anne et al. “KernelBench: Can LLMs Write Efficient GPU Kernels?” *ArXiv* abs/2502.10517 (2025): n. pag.