Optimizing PyTorch Inference with LLM-Based Multi-Agent Systems

Kirill Nagaitsev[1], Luka Grbcic[2], Samuel Williams[2], Costin Iancu[2]

[1] Northwestern University, [2] Lawrence Berkeley National Laboratory

Northwestern

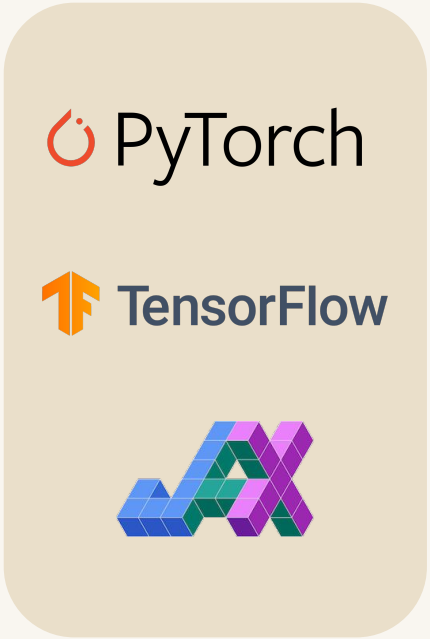
BERKELEY LAB

DOE CSGF

Upcoming Takeaways

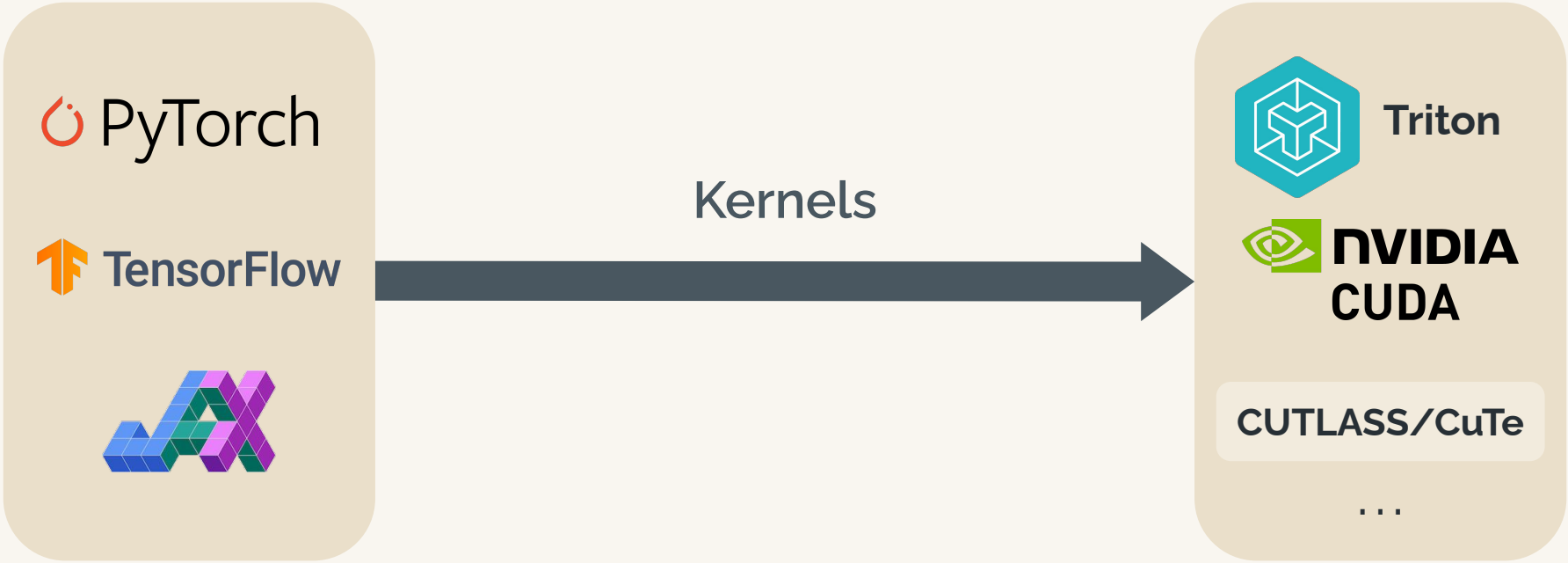
1. **ML model compilers (e.g. torch.compile) are easily outperformed by our methods**
2. Designating some budget to error-fixing loops is critical
3. Use exploit-oriented strategies at budgets of 100s of queries

Optimizing for Target Hardware



Unoptimized

Optimizing for Target Hardware



Optimizing for Target Hardware

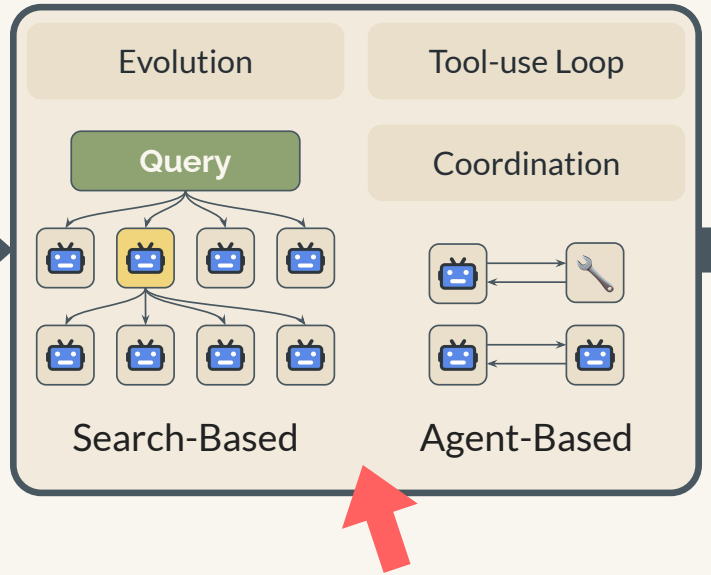
Optimizing for Target Hardware

PyTorch

```
# ReLU self-attention
def forward(self, x):
    # ... project x → q, k, v

    s = 1.0 / math.sqrt(k.size(-1))
    att = (q @ k.transpose(-2, -1)) * s
    att = att.masked_fill(...)
    att = F.relu(att)
    y = att @ v
    # ...
```

Unoptimized



Dynamics largely unexplored -
How to minimize cost here?

Triton

```
@triton.autotune(...)
@triton.jit
def fused_relu_att_kernel(...):
    # ... tiled, fused kernel

def relu_attention(q, k, v):
    # ... kernel setup

    fused_relu_att_kernel[grid](...)
    # ...
```

...

Optimized


```
# ReLU self-attention
def forward(self, x):
    # ... project x → q, k, v

    s = 1.0 / math.sqrt(k.size(-1))
    att = (q @ k.transpose(-2, -1)) * s
    att = att.masked_fill(...)
    att = F.relu(att)
    y = att @ v
    # ...
```

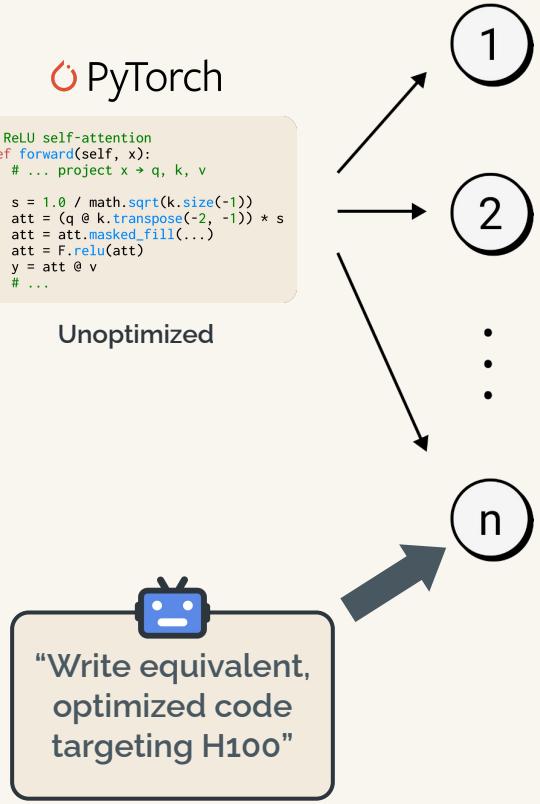
Unoptimized

PyTorch

```
# ReLU self-attention
def forward(self, x):
    # ... project x -> q, k, v

    s = 1.0 / math.sqrt(k.size(-1))
    att = (q @ k.transpose(-2, -1)) * s
    att = att.masked_fill(...)
    att = F.relu(att)
    y = att @ v
    # ...
```

Unoptimized

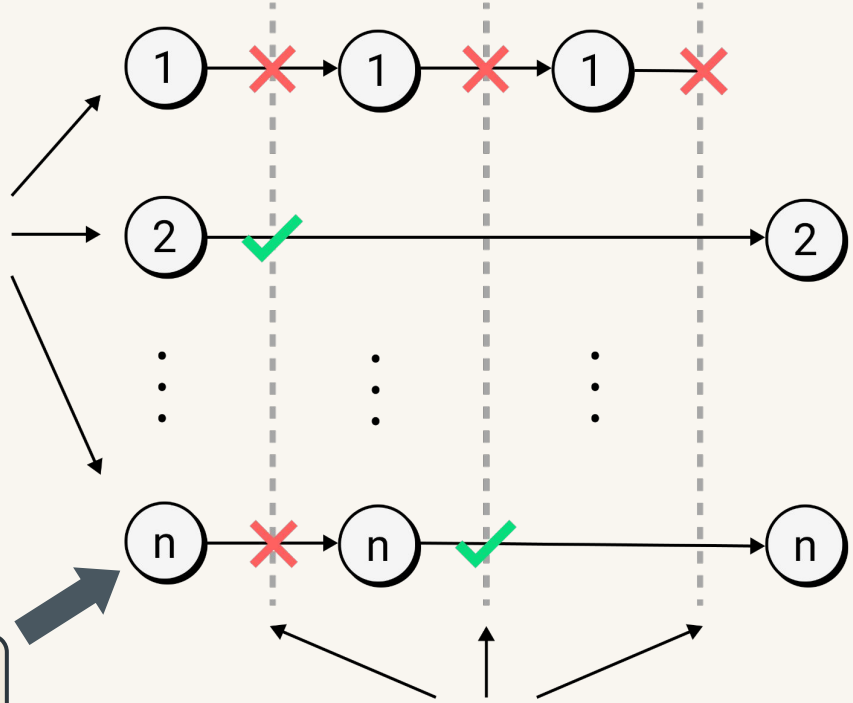


PyTorch

```
# ReLU self-attention
def forward(self, x):
    # ... project x -> q, k, v

    s = 1.0 / math.sqrt(k.size(-1))
    att = (q @ k.transpose(-2, -1)) * s
    att = att.masked_fill(...)
    att = F.relu(att)
    y = att @ v
    # ...
```

Unoptimized



"Write equivalent, optimized code targeting H100"

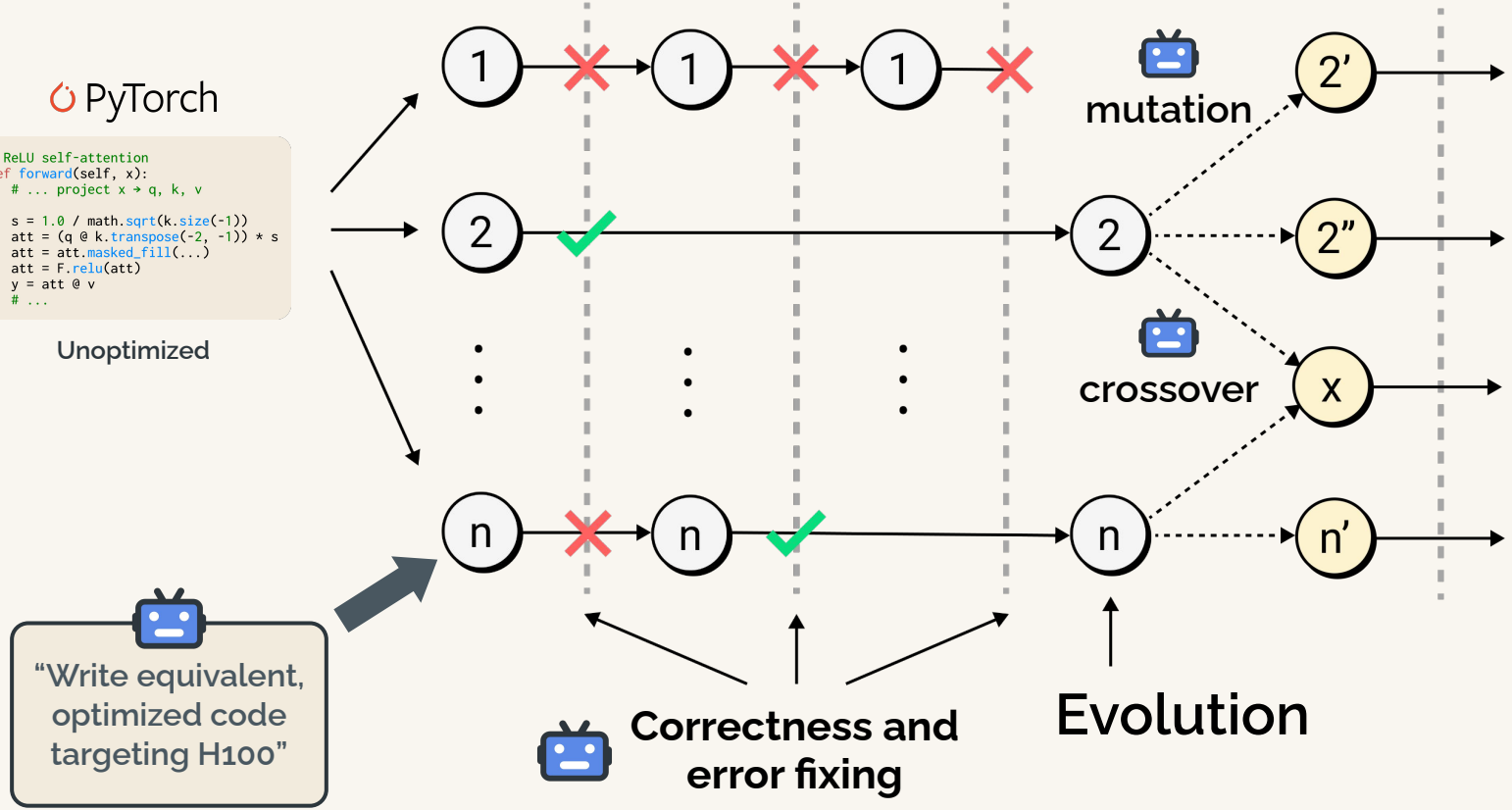
Correctness and error fixing

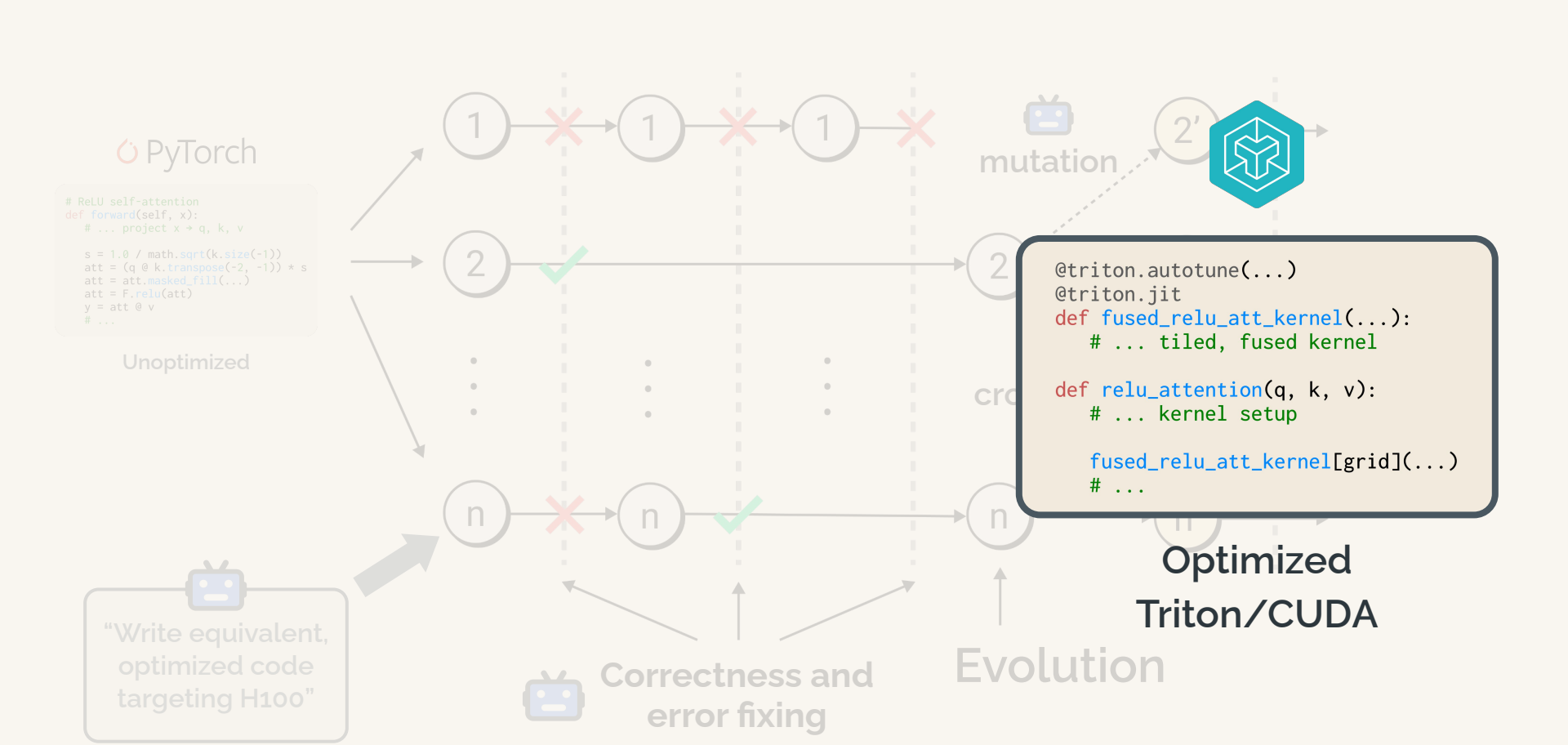
PyTorch

```
# ReLU self-attention
def forward(self, x):
    # ... project x → q, k, v

    s = 1.0 / math.sqrt(k.size(-1))
    att = (q @ k.transpose(-2, -1)) * s
    att = att.masked_fill(...)
    att = F.relu(att)
    y = att @ v
    # ...
```

Unoptimized

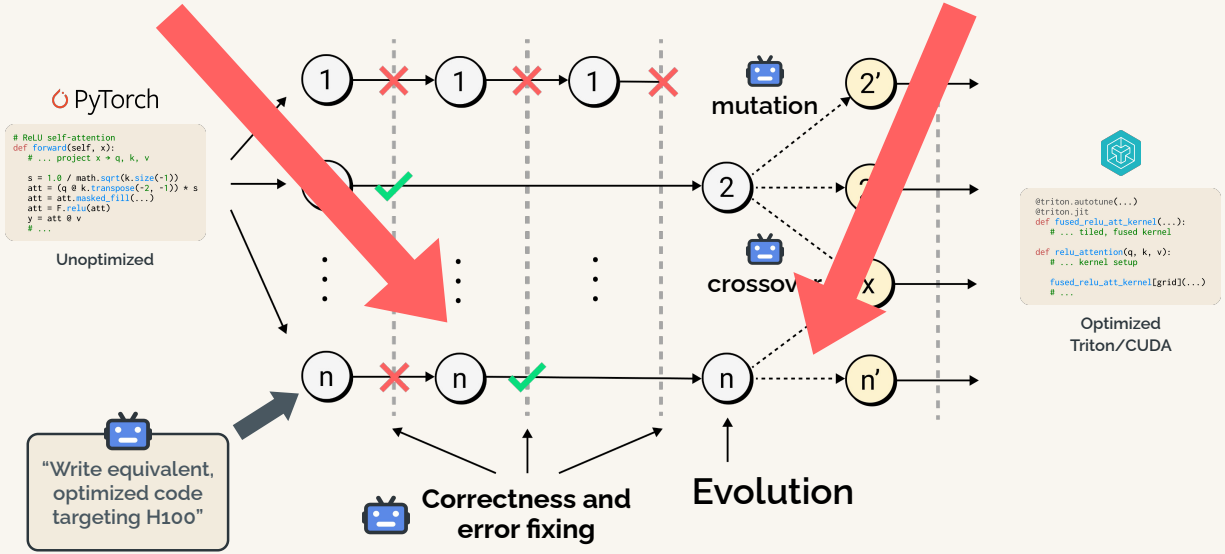



PyTorch
# ReLU self-attention
def forward(self, x):
# ... project x → q, k, v
s = 1.0 / math.sqrt(k.size(-1))
att = (q @ k.transpose(-2, -1)) * s
att = att.masked_fill(...)
att = F.relu(att)
y = att @ v
# ...
Unoptimized
1
1
1
2
n
n
n
mutation
2'
@triton.autotune(...)
@triton.jit
def fused_relu_att_kernel(...):
# ... tiled, fused kernel
def relu_attention(q, k, v):
# ... kernel setup
fused_relu_att_kernel[grid](...)
...
Optimized
Triton/CUDA
"Write equivalent, optimized code targeting H100"
Correctness and error fixing
Evolution

Say we have a budget of 300 LLM queries for this full process:

Designate some to error fixing, or focus entirely on optimization steps?

Focus all effort on top-performing code, or maintain a wide search base?

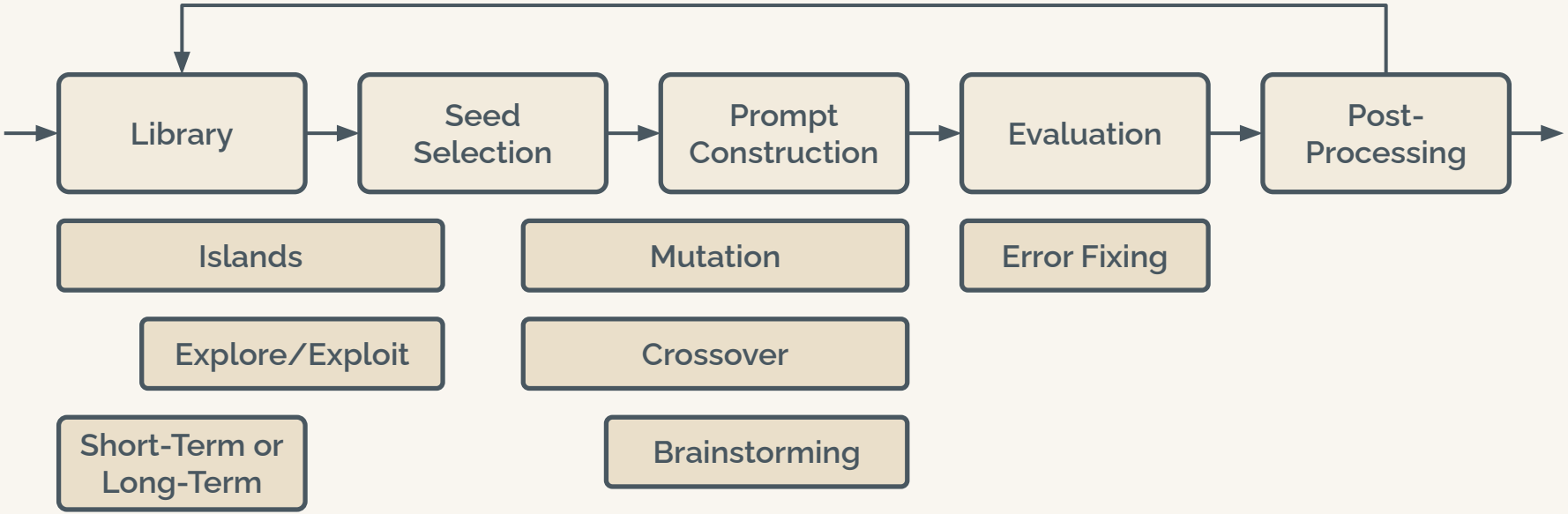


PIKE: Multi-Agent Evolutionary Framework

Logical Steps

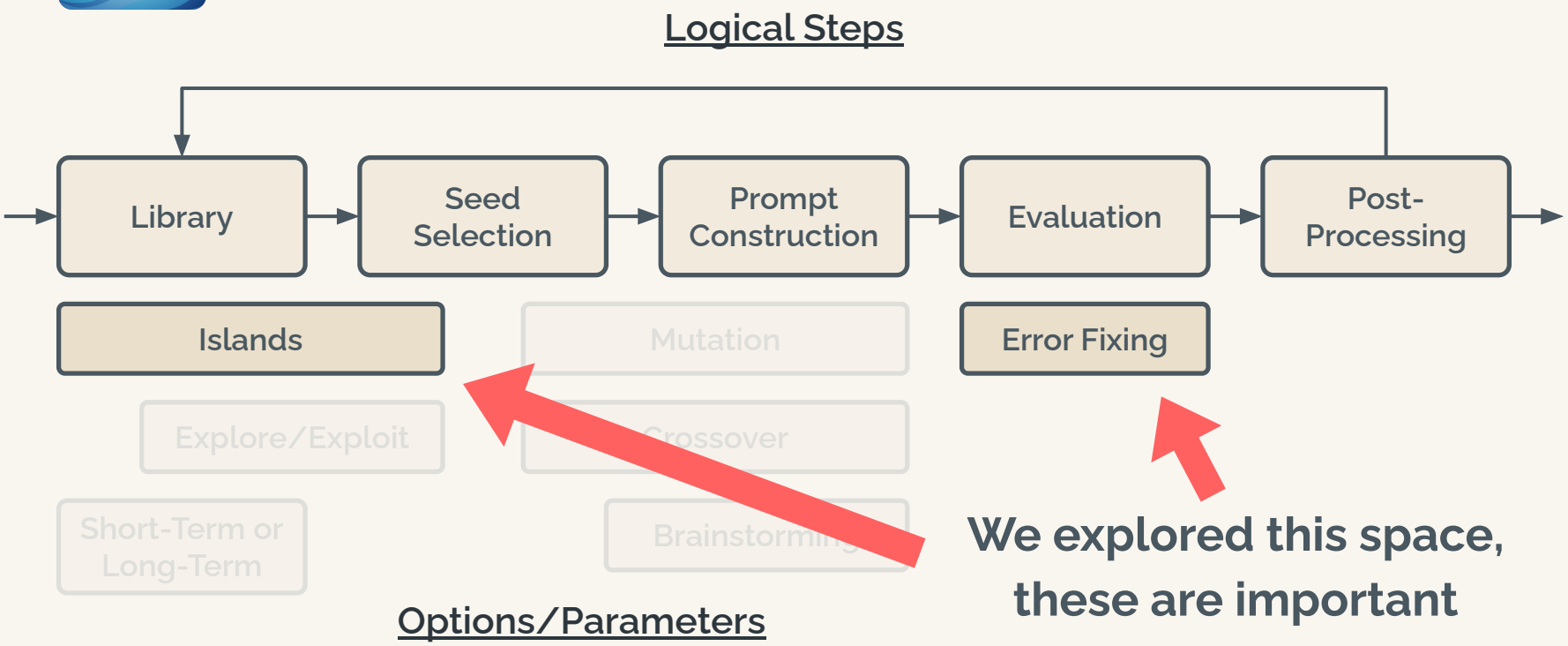
PIKE: Multi-Agent Evolutionary Framework

Logical Steps



Options/Parameters

PIKE: Multi-Agent Evolutionary Framework



Setup

- **KernelBench** suite
  - 250 self-contained PyTorch tasks
  - Varying difficulty levels, some 100+ LoC
- **OpenEvolve** extended with agents & evaluator
  - OpenEvolve - LLM-based evolutionary framework, no distinct agents by default

Benchmark: Extended KernelBench

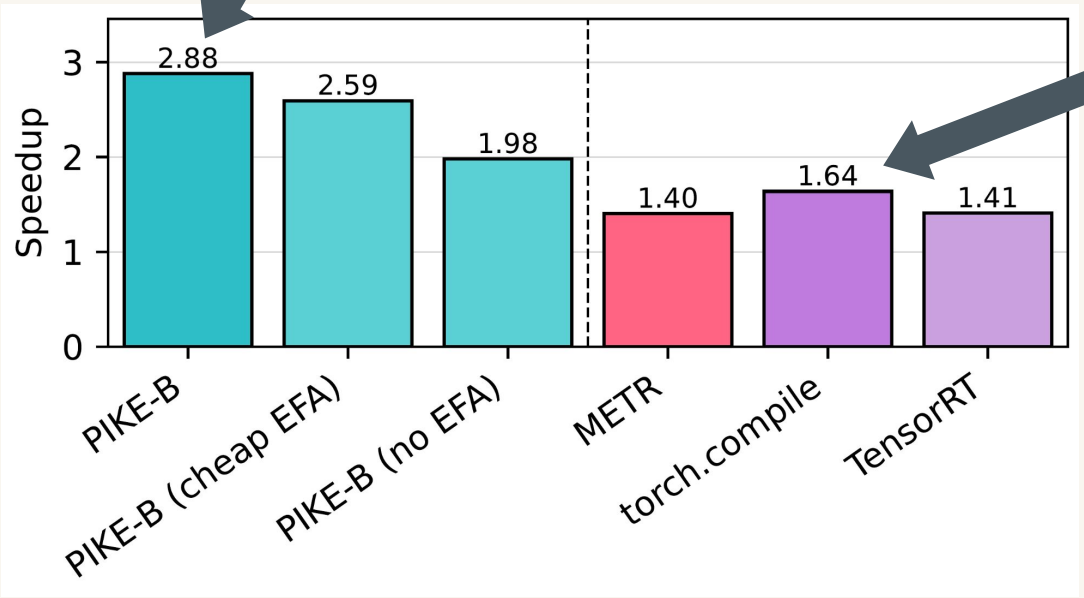
Level	Description	Examples	Tasks
3 (filtered)	Prev. generation AI models	MLP, CNN, RNN, Transformers/Attn.	30
5	2024-25 SOTA Vision/Language Blocks	DeepSeek-V3, Llama 3, RWKV, SD3, Mamba-2, S4, Hunyuan Video	14

Setup (cont.)

- Multiple models tested (focus on Gemini 2.5 Pro)
- 300 LLM queries/task (~$30-50/task)
- Target: NVIDIA H100 GPU

Agent-based evolutionary strategies work well: ~3x over PyTorch eager

Higher is better

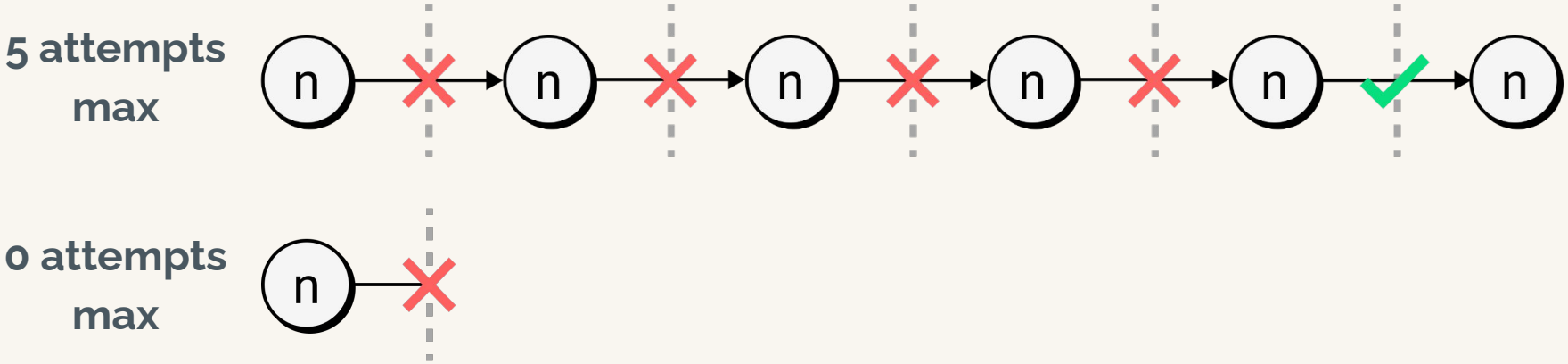


Significantly beating torch.compile

Geomean speedup over PyTorch eager, Level 3

We have a budget of 300 LLM queries for this full process:

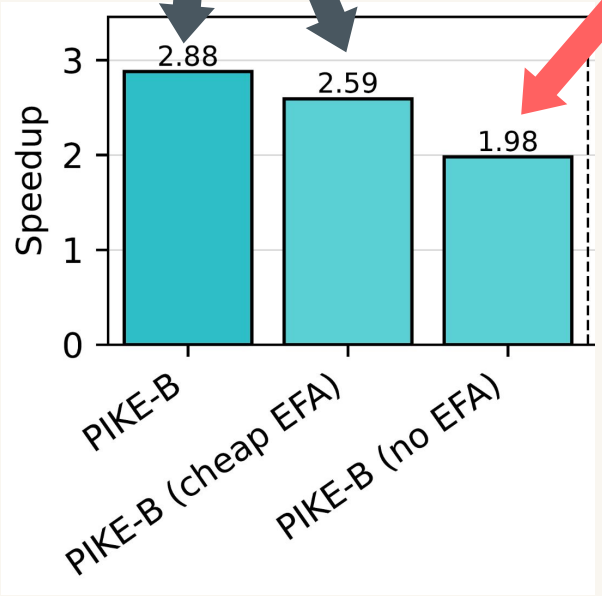
Designate some to explicit error fixing, or
focus entirely on optimization steps?



Give max error fixing budget for any new solution

5 attempts max per broken solution for error fixing

NO budget used for error fixing



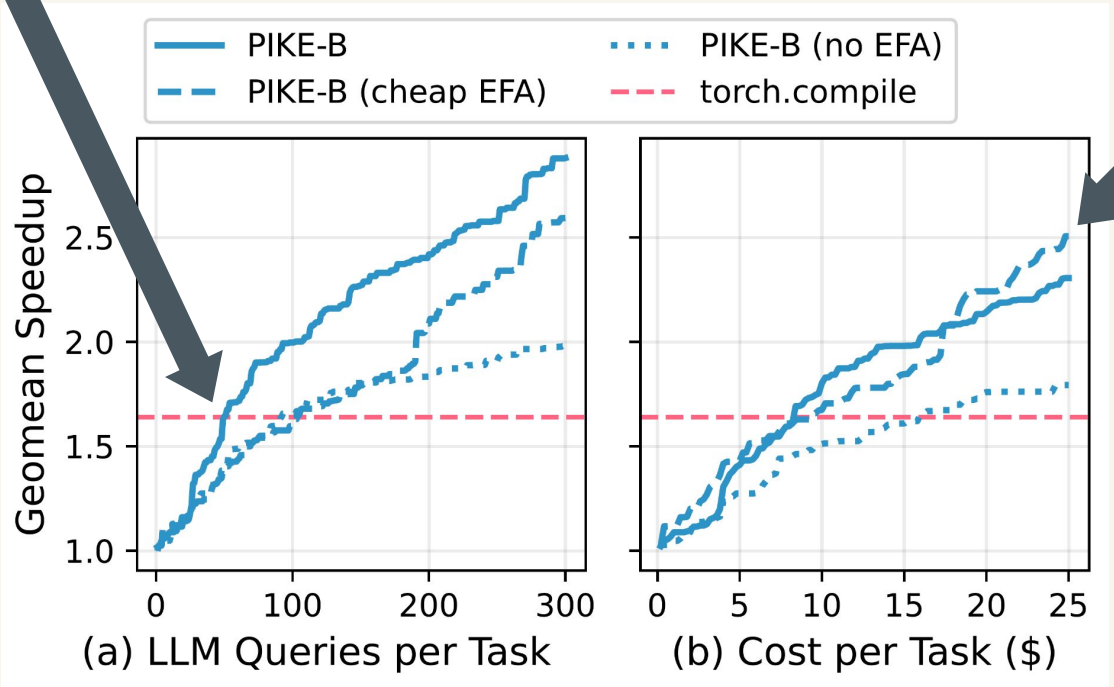
(70-80% of solutions fixed within 5 attempts)

Error fixing is critical: initially broken solutions are worth it to fix

Geomean speedup over PyTorch eager, Level 3

torch.compile beaten within 100 queries

Cheap error fixing wins at $25 budget



Higher is better

Geomean speedup over PyTorch eager under budget, Level 3

We have a budget of 300 LLM queries for this full process:

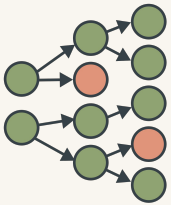
Focus all effort on top-performing code, or maintain wide search base?

Exploit-oriented evolution

- Focus search on exploiting elite solutions
- Minimal maintenance of population diversity

Explore-oriented evolution

- Uses multiple islands - parallel subpopulations for maintaining diversity
- Samples from large library, including non-elite solutions

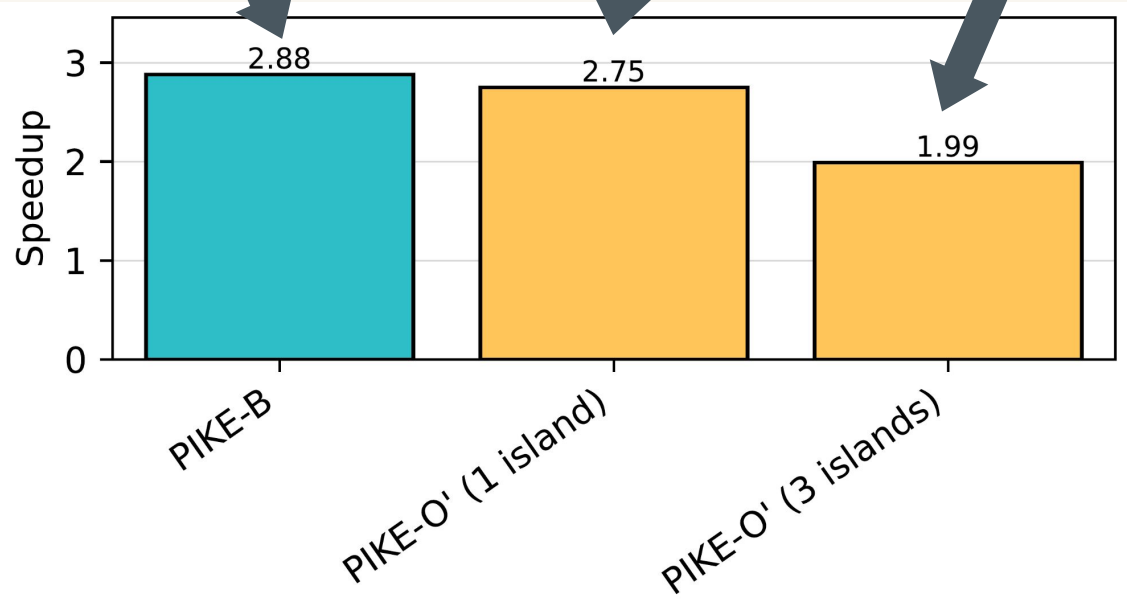


1 island, top-3 sampling

1 island, broader sampling

3 islands

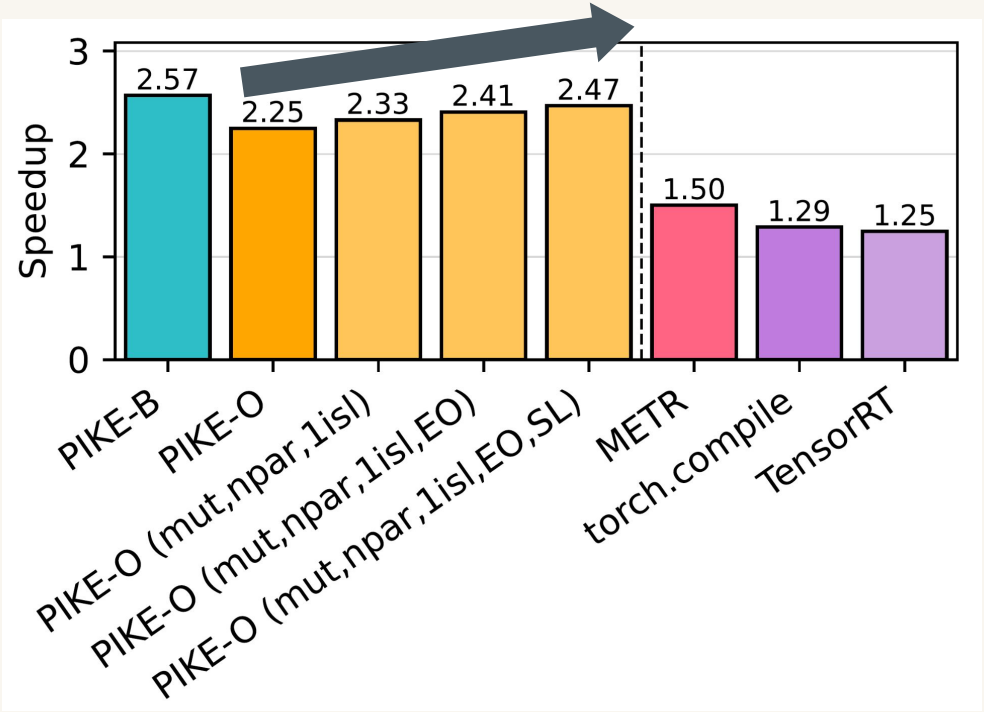
(all error-fixing configurations)



Geomean speedup over PyTorch eager, Level 3

Diversity maintenance from multiple islands not worth it at 100s of queries budget

Reduce islands from 3 to 1, sample elite solutions more heavily, shorter library



Geomean speedup over PyTorch eager, Level 5

Best to use exploit-oriented strategies at budgets of 100s of queries

Key Takeaways

1. **ML model compilers (e.g. torch.compile) are easily outperformed by our methods**
2. Designating some budget to error-fixing loops is critical
3. Use exploit-oriented strategies at budgets of 100s of queries

github.com/pike-project/pike

Paper Preprint

Kirill Nagaitsev

knagaitsev@u.northwestern.edu

https://kirn.io

DOE CSGF

Northwestern

BERKELEY LAB