Towards Verified and AI-Driven Accelerator Programming

Sahil Bhatia, Alvin Cheung
Jie Qiu, Colin Cai, Charles Hong, Sophia Shao, Sanjit Seshia

Agenda

1. Verified Code **Translation**
   a) Tools: **MetaLift** + **LLMLift**

2. Accelerator Code **Optimization**
   a) Tools: **Autocomp**

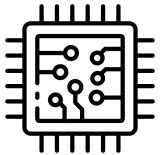
Domain Specific Languages (DSL)

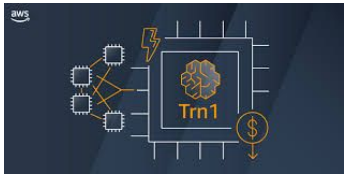
PyTorch

MLX

- Domain-Specific Optimizations
- Better Readability and Maintainability

GraphLab





Code needs to be rewritten!

```
DO k=y_min,y_max
    DO j=x_min,x_max+1
      vol_flux_x(j,k)=0.5_8*dt*x(j,k) &
          *(x(j,k) &
          + x(j,k+1)+x(j,k)+x(j,k+1))
    ENDDO
   ENDDO
```

```
vector<int> normal_blend_8(
  vector<int> b,
  vector<int> a,
  int o
) {
  vector<int> out;
  for (int i = 0; i < base.size(); ++i)
    out.push_back(o * a[i] + (32 - o) * b[i]);
  return out;
}
```

```
public class WordCount {
  private static Map<String, Integer> countWords(
    List<String> words
  ) {
    Map<String, Integer> counts
      = new HashMap<String, Integer>();
    for (int j = 0; j < words.size(); j++) {
      String word = words.get(j);
      Integer prev = counts.get(word);
      if (prev == null)
        prev = 0;
      counts.put(word, prev + 1);
    }
    return counts;
  }
}
```

Scientific Computing

Halide

```
flux(j, k) = 0.5f * dt * x(j, k) * (x(j,
k) + x(j, k+1) + x(j, k) + x(j, k+1));
```

Image Processing


```
out = o* a + (32 - b) * b
```

Big Data


```
words.mapToPair(word -> new
Tuple2<String,
Integer>(word,1)).reduceByKey((c1, c2)
-> c1+c2).collectAsMap();
```

Prior Approaches: Pattern-Driven Compilation

```
matrix<int> screen_blend_8(
  matrix<int> b,
  matrix<int> a
) {
  matrix<int> out;
  for (int i = 0; i < b.size();
    vector<int> row_vec;
    for (int j = 0; j < b[0].si
      int pixel = b[i][j] + a[i
        - (b[i][j] * a[i][j]) /
      row_vec.push_back(pixel);
      }
    out.push_back(row_vec);
  }
}
```

- Brittle
- Difficult to Get Right
- Hard to Maintain

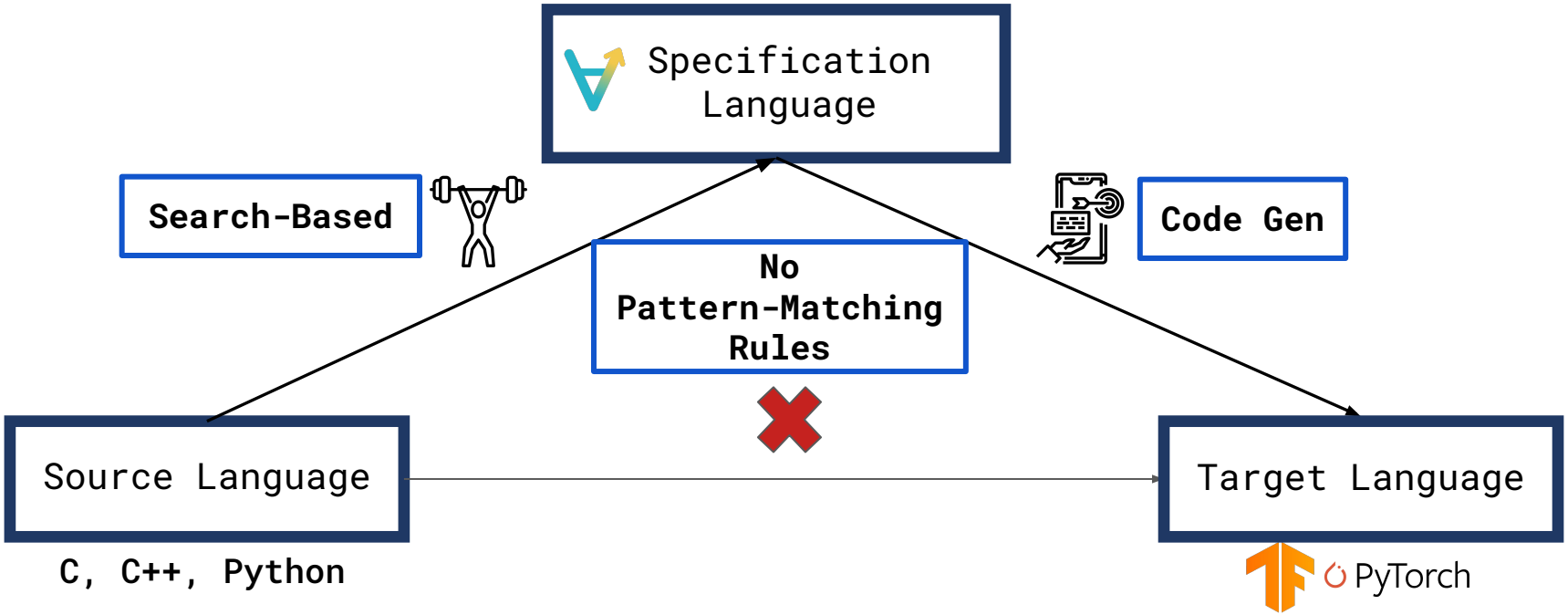
```
torch.sub(
  torch.add(b,a),
  torch.div(
    rch.mul(b,a),
    5
```

```
for (int i = 0; i < b.size(); i++) {
  for (int j = 0; j < b[0].size(); j++) {
    int pixel = $op1(b[i][j], a[i][j]);
  }
}
```

```
torch.$op1(b, a)
```

Translating imperative code to MapReduce. Radio et al. OOPSLA 2014

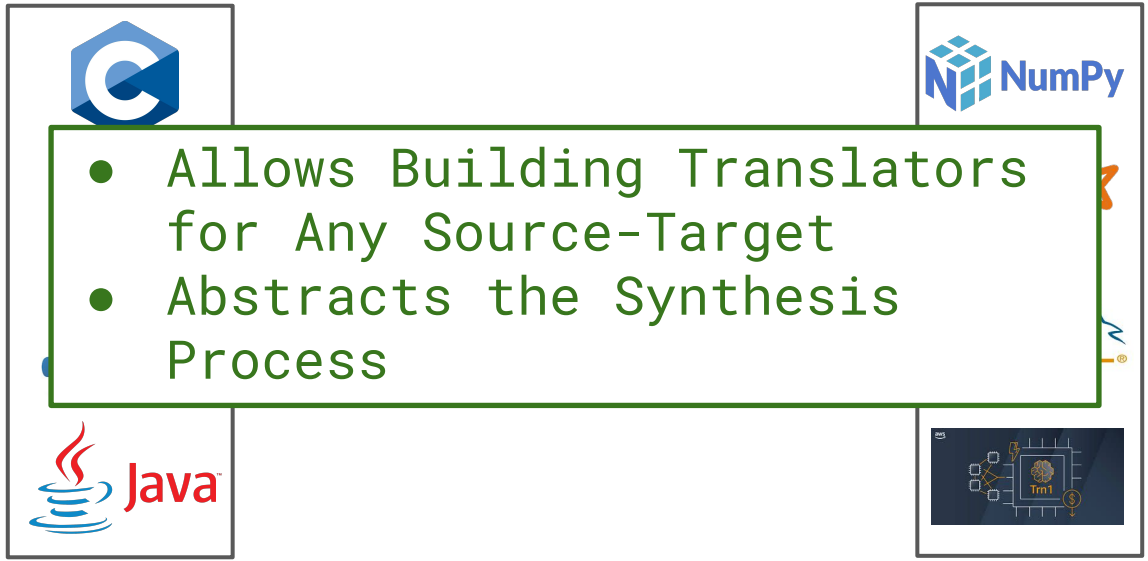
Verified Lifting



Optimizing Database-Backed Applications with Query Synthesis. Cheung at al. PLDI 2013

MetaLift: A Verified Lifting Framework for Building DSL Code Translators

What Does MetaLift Solve?

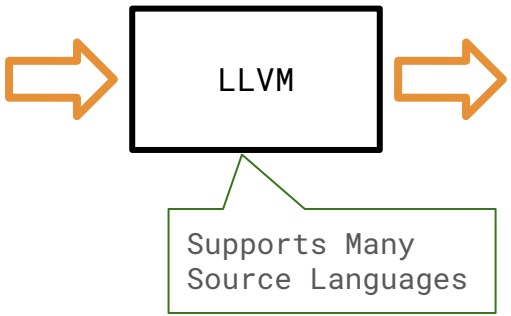


- Allows Building Translators for Any Source-Target
- Abstracts the Synthesis Process

Input Code

Analysis
- VC Generator

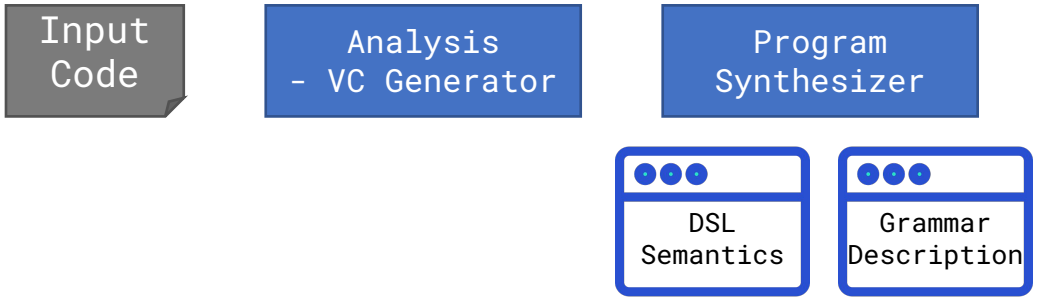
```
matrix<int> screen_blend_8(
  matrix<int> b,
  matrix<int> a
) {
  matrix<int> out;
  for (int i = 0; i < b.size(); i++) {
    vector<int> row_vec;
    for (int j = 0; j < b[0].size(); j++) {
      int pixel = b[i][j] + a[i][j]
        - (b[i][j] * a[i][j]) / 255;
      row_vec.push_back(pixel);
      }
    out.push_back(row_vec);
  }
}
```



Specification

source == target (**PS**)

For loopy programs:
source == target (**PS + Inv**)



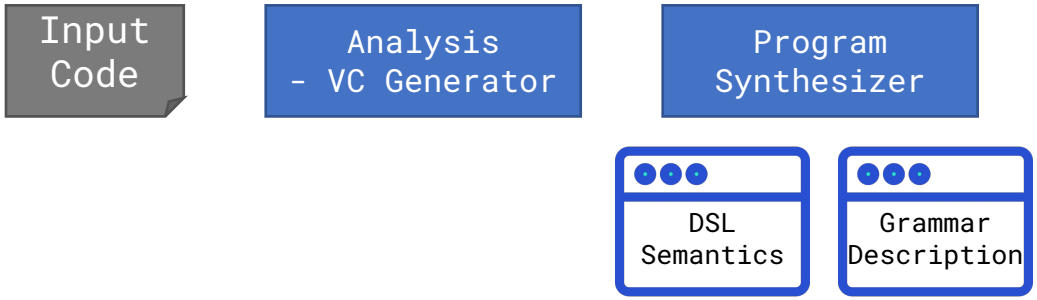

```python
def mat_add(
  matrix_x: List[List[int]],
  matrix_y: List[List[int]]
) -> List[List[int]]:
  return (
    []
    if len(matrix_x) < 1 or not len(matrix_x) == len(matrix_y)
    else [
      vec_add(matrix_x[0], matrix_y[0]),
      *mat_add(matrix_x[1:], matrix_y[1:]),
    ]
  )
```

G

```
out  := exp0
exp0 := mat_add(exp1, exp1) | mat_sub(exp1,exp1)
exp1 := mat_add(var,var) | mat_sub(var,var)
var  := a | b | 255
```

GI

```
inv1 := exp1 and epx2 and exp3
exp1 := i >= 0 | i <= 0 | i == 0
exp2 := i <= b.size() | i <= b.size() | i == b.size()
exp3 := out == mat_add(a[:i], b[:i]) |
        out == mat_sub(a[:i], b[:i])
```




```
matrix<int> screen_blend_8(
  matrix<int> b,
  matrix<int> a
) {
  matrix<int> out;
  inv1 = i >= 0 and i <= b.size() ..
  for (int i = 0; i < b.size(); i++) {
    vector<int> row_vec;
    for (int j = 0; j < b[0].size(); j++) {
      int pixel = b[i][j] + a[i][j]
        - (b[i][j] * a[i][j]) / 255;
      row_vec.push_back(pixel);
      }
    inv1 =
    out.push_back(row_vec);
  }
}
inv1 =
out  = mat_add...
```

G

```
out  := exp0
exp0 := mat_add(exp1, exp1) | mat_sub(exp1,exp1)
exp1 := mat_add(var,var) | mat_sub(var,var)
var  := a | b | 255
```

GI

```
inv1 := exp1 and epx2 and exp3
exp1 := i >= 0 | i <= 0 | i == 0
exp2 := i <= b.size() | i <= b.size() | i == b.size()
exp3 := out == mat_add(a[:i], b[:i]) |
        out == mat_sub(a[:i], b[:i])
```

Input Code

Analysis - VC Generator

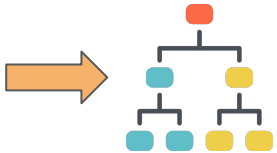
Program Synthesizer

DSL Semantics

Grammar Description

DSL Semantics

Grammar Description



```
result = mat_add(a,b)
result = mat_sub(a, mat_add(a,b))
result = mat_sub(
  mat_add(b,a),
  mat_divide(
    mat_multiply(b,a),255
  )
)
```

```
mat_sub(
  mat_add(b,a),
  mat_divide(
    mat_multiply(b,a),255
  )
)
```

Symbolic Synthesis Engine

Enumerating Expressions

Transpiled Code + Invariant

Input Code

Analysis - VC Generator

Program Synthesizer

Program Verification

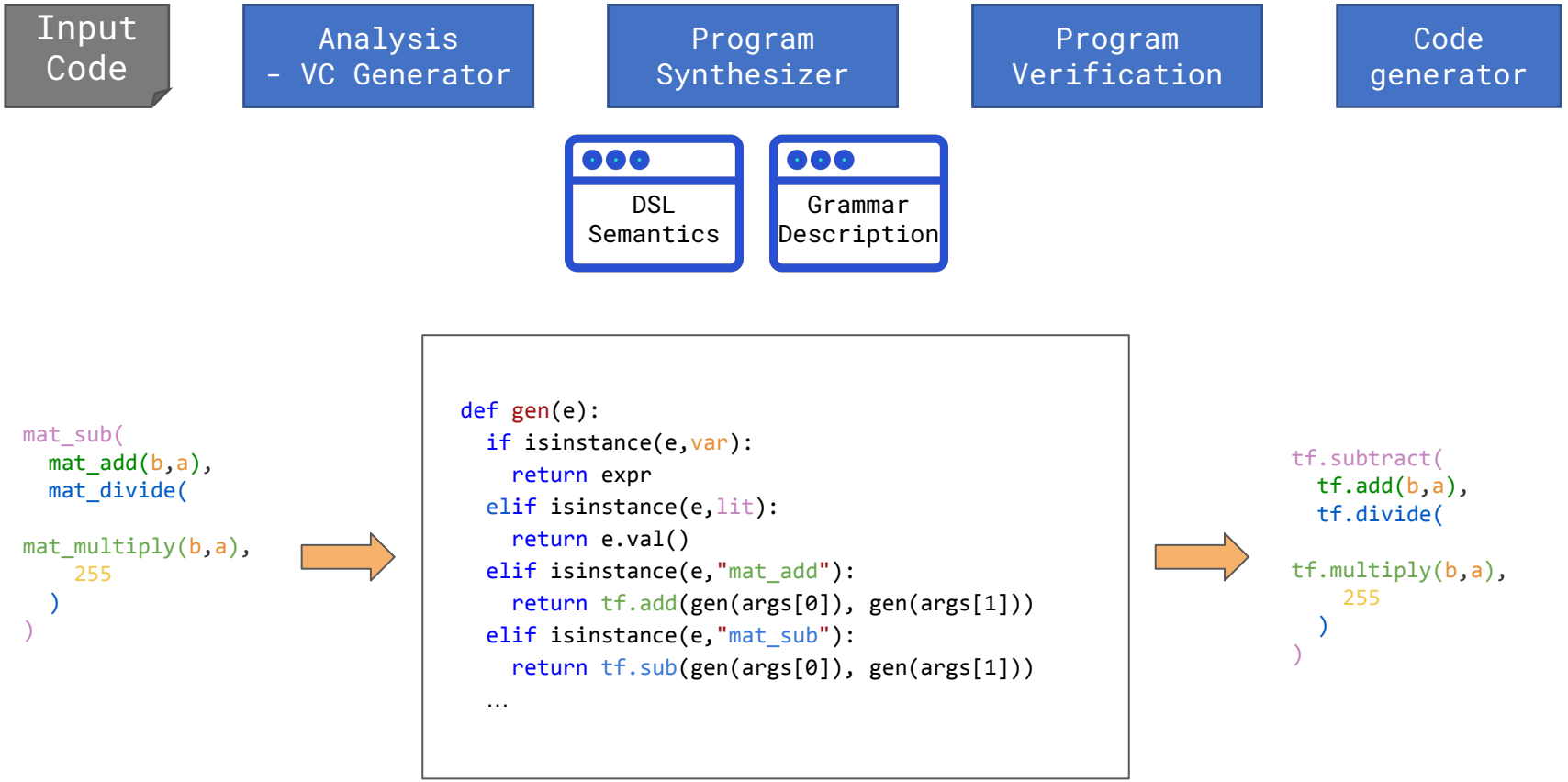
DSL Semantics

Grammar Description

```
PS  := mat_sub(
         mat_add(b,a),
         mat_divide(mat_multiply(b,a),255)
       )
Inv := i >= 0 and i <= b.size() and
       out == mat_sub(
         mat_add(b[:i],a[:i]),
         mat_divide(mat_multiply(b[:i],a[:i]),255)
       )
```

Theorem Prover

$$\exists PS \in G$$
$$\exists Inv \in GI$$


Input Code
Analysis - VC Generator
Program Synthesizer
Program Verification
Code generator
DSL Semantics
Grammar Description
mat_sub(
mat_add(b,a),
mat_divide(
mat_multiply(b,a),
255
)
)
def gen(e):
if isinstance(e,var):
return expr
elif isinstance(e,lit):
return e.val()
elif isinstance(e,"mat_add"):
return tf.add(gen(args[0]), gen(args[1]))
elif isinstance(e,"mat_sub"):
return tf.sub(gen(args[0]), gen(args[1]))
...
tf.subtract(
tf.add(b,a),
tf.divide(
tf.multiply(b,a),
255
)
)

Results

	Accuracy	LOC vs Dedicated Compiler
Java / Apache Spark	40/45	< ~90%
C	5/5	< 500 LOC
NumPy		
TensorFlow		
PyTorch		

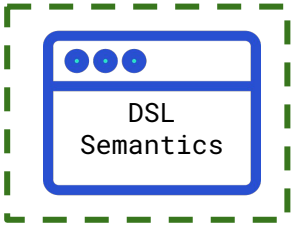
- Demonstrates Generality of MetaLift
- Built Transpilers for 3 Different Application Domains

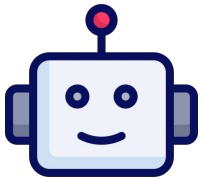
LLMLift : Verified Code Translation with LLMs

Bottlenecks in MetaLift

Manual Effort

Domain Specific Knowledge





LLM

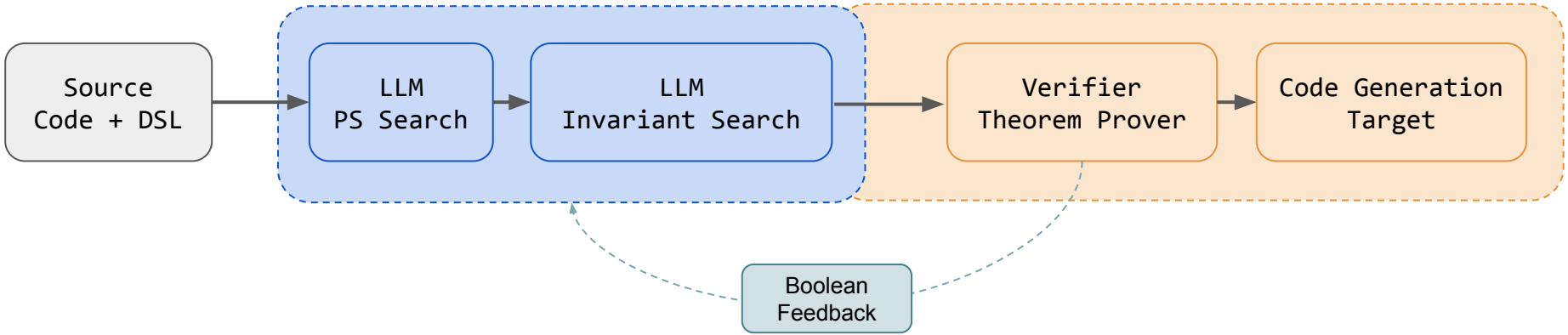
Enumerates Transpiled Code + Proof

```
result = mat_add(a,b)
result = mat_sub(a, mat_add(a,b))
result = mat_sub(
  mat_add(b,a),
  mat_divide(mat_multiply(b,a),255)
)
```

Enumerating Expressions

Fast!

LLMLift Framework



Code Search with LLMs

```
Your task is to rewrite the given `test` Python Function. You need to use
only the set of provided functions and constants to achieve this. The
rewritten program should be semantically equivalent to the `test`
function.
```

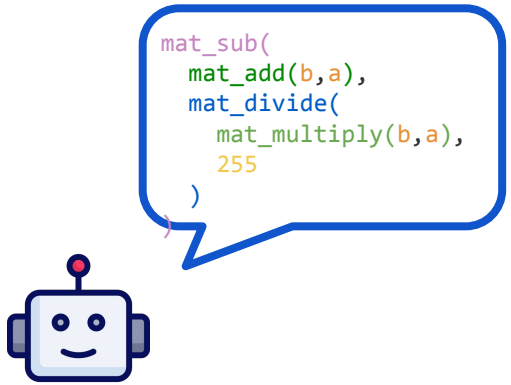
Prompt Preamble

```
#defined functions
def mat_add(matrix_x: List[List[int]], matrix_y: List[List[int]]) ->
List[List[int]]:
  return ( []
    if len(matrix_x) < 1 or not len(matrix_x) == len(matrix_y)
    else [
      vec_add(matrix_x[0], matrix_y[0]),
      *mat_add(matrix_x[1:], matrix_y[1:]),
    ]
  )
...
```

Target language definition

```
matrix<int> screen_blend(matrix<int> b, matrix<int> a) {
  ...
}
```

Source Program



Proof Search with LLMs

```
Your task is to rewrite the given `test` Python Function. You need to use
only the set of provided functions and constants to achieve this. The
rewritten program should be semantically equivalent to the `test`
function.
```

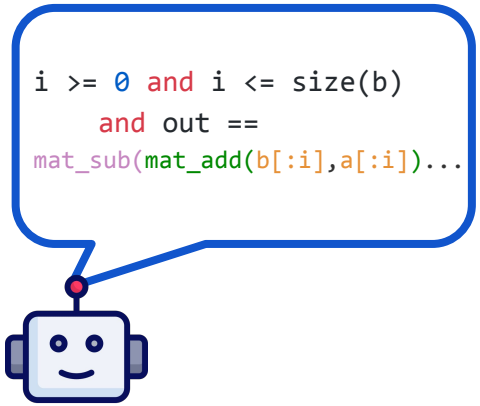
Prompt Preamble

```
#defined functions
def mat_add(matrix_x: List[List[int]], matrix_y: List[List[int]]) ->
List[List[int]]:
  return ( []
    if len(matrix_x) < 1 or not len(matrix_x) == len(matrix_y)
    else [
      vec_add(matrix_x[0], matrix_y[0]),
      *mat_add(matrix_x[1:], matrix_y[1:]),
    ]
  )
...
```

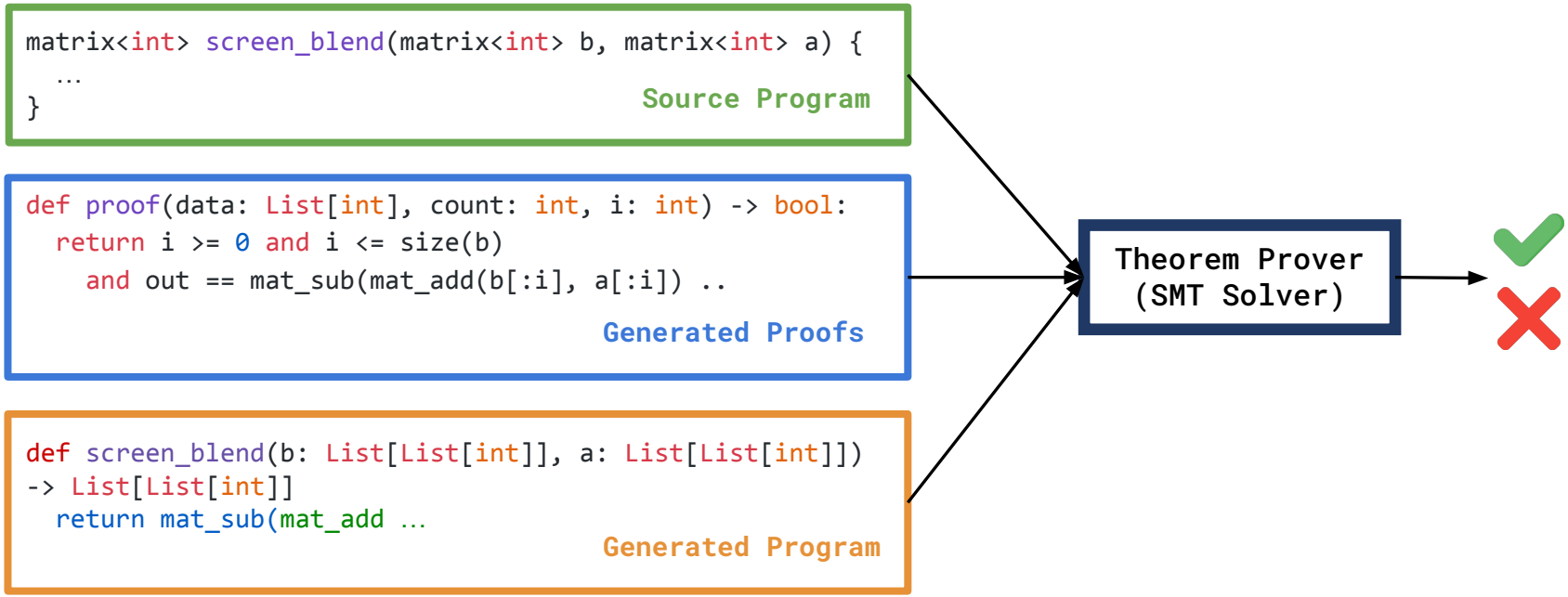
Target language definition

```
matrix<int> screen_blend_8(matrix<int> b, matrix<int> a) {
  ...
assert == mat_sub(mat_add(b,a),mat_divide
(mat_multiply(b,a),255))
}
```

Source Program



Verification



Results

Source Lang	Target Lang	Symbolic Synthesizer	LLMLift
Java	Apache Spark	40/45	44/45
C	Barefoot Networks	10/10	10/10
C	TACO	57/60 (1000 LoC Heuristics)	60/60
C	PyTorch	23/23 (1200 LoC Heuristics)	23/23

- Easily Built Transpilers for 4 Different Domains
- Transpiles More Programs in Less Time
- Requires No Domain-Specific Heuristics

TensIR: An IR for Tensor Computation

Basic tensor operators

```
p in op       := s_comp | t_comp | c_control
scalar_comp   := reduce_max | reduce_sum
tensor_comp   := tensor_scalar | tensor_tensor | transpose |
                 tensor_vec_prod | access
control_flow  := ite
access        := t[:l] | t[l:] | t[l1:l2]
```

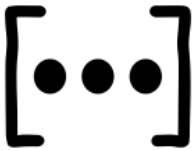
Benchmarks

Open-source implementations of blending modes in Photoshop

12 Benchmarks

C++ based ML inference code of Llama2

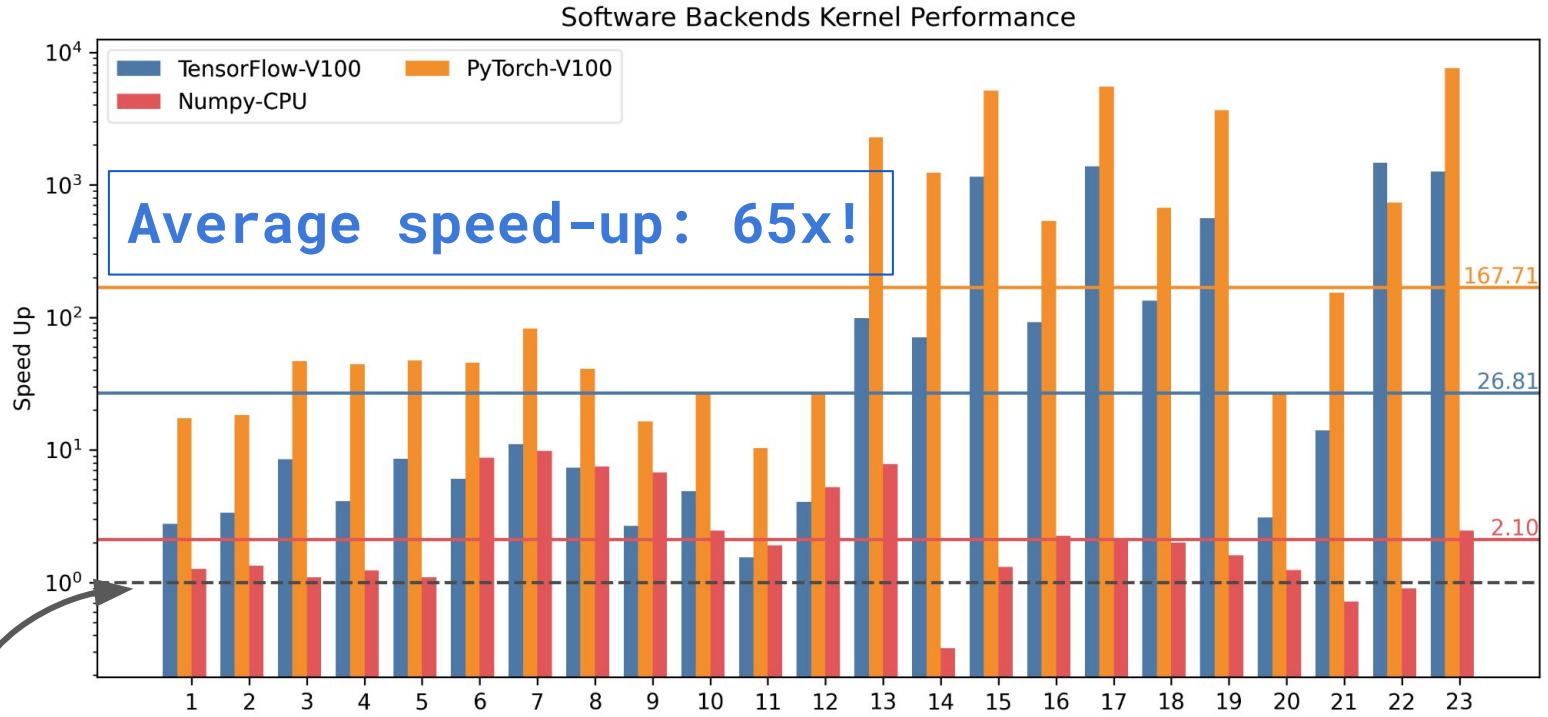
11 Benchmarks



Open-source array-processing benchmarks

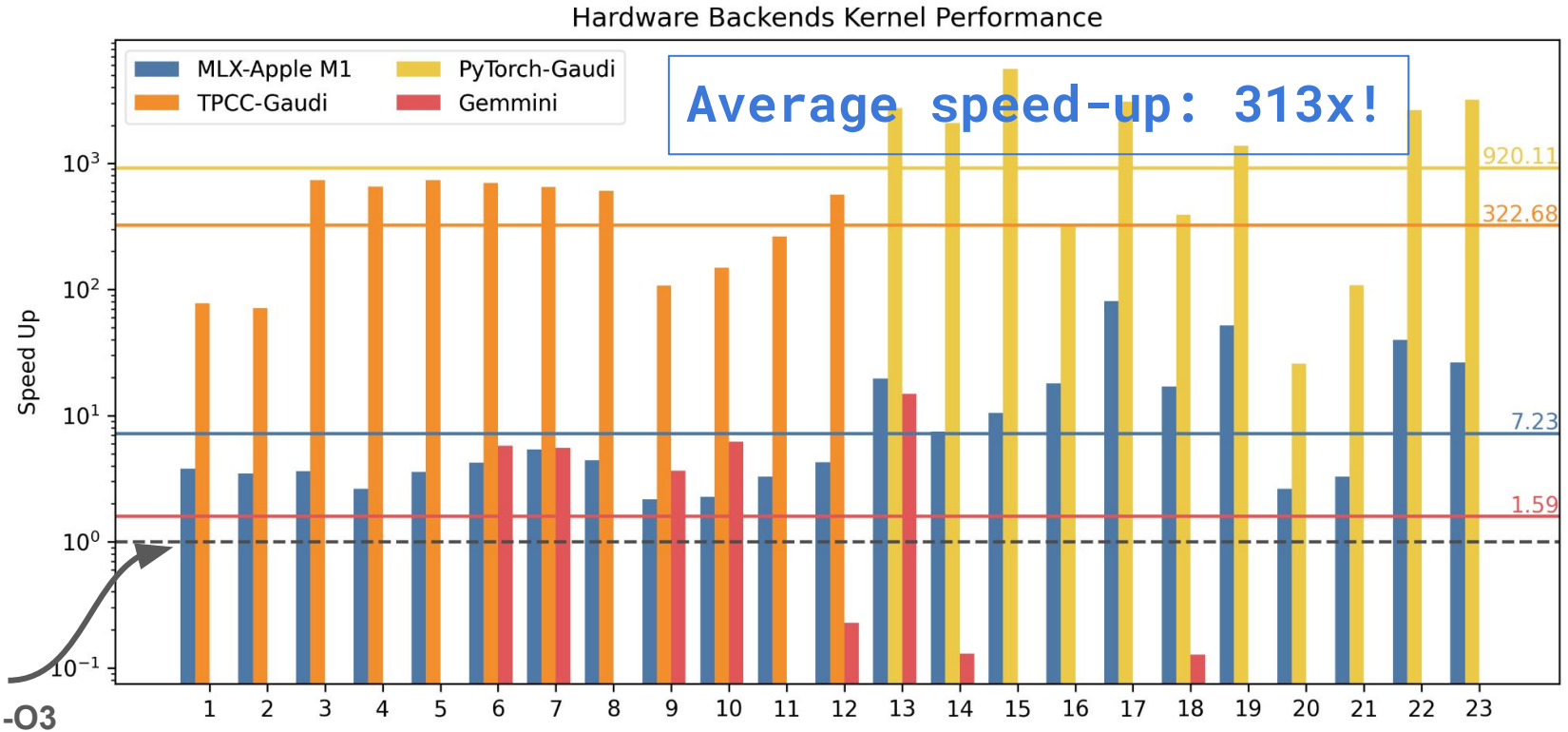
46 Benchmarks

Performance Timings – Software Frameworks



GCC -O3 Baseline

Performance Timings – Hardware Accelerators



Tutorial

Special thanks to…

AWS for providing support and the instances you will be using today!

Setup

1. Sign-in to the AWS machines
   a. Go to https://catalog.us-east-1.prod.workshops.aws
      Join with access code: d30f-023176-6d

2. Clone the MetaLift repository
   a. `git clone https://github.com/metalift/metalift.git`
   b. `cd metalift`
   c. `git checkout asplos`
   d. `docker build -f Dockerfile.tutorial -t llmlift-tutorial .`
   e. `docker run --rm -it llmlift-tutorial bash`

Example 1 - Sequential Program

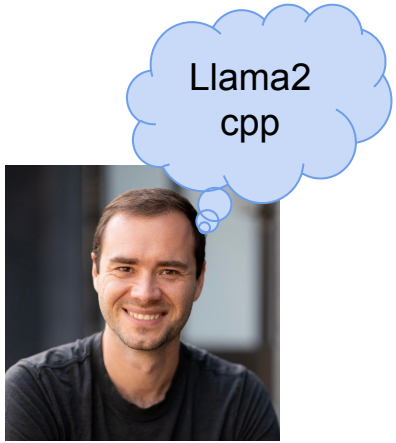
```
int test(int base, int arg1, int base2, int arg2)
{
    int a = 0;

    a = (base + base2) + arg1 * arg2;

    a = a + a;

    return a;
}
```

Source Program

```
def fma(x, y, z):
    return x + y * z
```

Target Language

Example 2 - Loopy Program (RMSNorm)




```
void rmsnorm(float* o, float* x, float* weight, int size) {
    // calculate sum of squares
    float ss = 0.0f;
    for (int j = 0; j < size; j++) {
        ss += x[j] * x[j];
    }
    ss /= size;
    ss += 1e-5f;
    ss = 1.0f / sqrtf(ss);
    // normalize and scale
    for (int j = 0; j < size; j++) {
        o[j] = weight[j] * (ss * x[j]);
    }
}
```

https://github.com/karpathy/llama2.c/blob/master/run.c

Equivalence Proof

```cpp
#include <vector>
using namespace std;

int rmsnorm_part1(vector<int> input,
vector<int> weight) {
    int ss = 0;
    for (int i = 0; i < input.size(); i++)
        ss += input[i] * input[i];
    return ss;
}
```

ss == reduce_sum(vec_elem_mul(input * input))

`i >= 0 ∧ i <= input.size() ∧ ss = reduce_sum(vec_elem_mul(input[:i], input[:i]))`

Step 1 — Base case I(0)

Initialization

```
int ss = 0;
int i = 0;
```

Invariant to verify

```
i >= 0  ∧  i <= input.size()  ∧  ss = reduce_sum(vec_elem_mul(input[:i], input[:i]))
```

`i >= 0` → `0 >= 0`

trivially true

✓

`i <= input.size()` → `0 <= input.size()`

size() is always non-negative

✓

`ss == reduce_sum(vec_elem_mul(input[:0],input[:0])` → `0 == reduce_sum([])`

sum of empty list is 0 wi

✓

I(0) holds ✓

Step 2 — Inductive step (I(i) → I(i+1))

What we assume

Inductive hypothesis

`ss == reduce_sum(input[:i]²)`

```
int ss += input[i] * input[i] ;
int i += 1;
```

Bound clause

`i+1 <= input.size()`

holds from loop guard i < input.size()

What we derive

`new_ss = ss + input[i] * input[i]`

loop body

`= reduce_sum(input[:i]²) + input[i]²`

I.H.

`= reduce_sum(input[:i+1]²)` ✓

I(i+1) holds ✓ — induction complete

Step 3 — Termination & conclusion

```
i >= 0  ∧  i <= input.size()  ∧ ss = reduce_sum(vec_elem_mul(input[:i], input[:i]))
```

At loop exit (i == input.size()), substitute into Invariant:

```
ss  ==  reduce_sum(vec_elemwise_mul(input[:input.size()], input[:input.size()]))
```

which simplifies to:

```
ss  ==  reduce_sum(vec_elemwise_mul(input, input))
```

which is exactly what the DSL one-liner returns:

```
ss == reduce_sum(vec_elem_mul(input * input))
```

∴ C source ≡ DSL target

The C++ loop and the DSL one-liner are formally proved functionally equivalent ✓

Example 3 - Extending the DSL

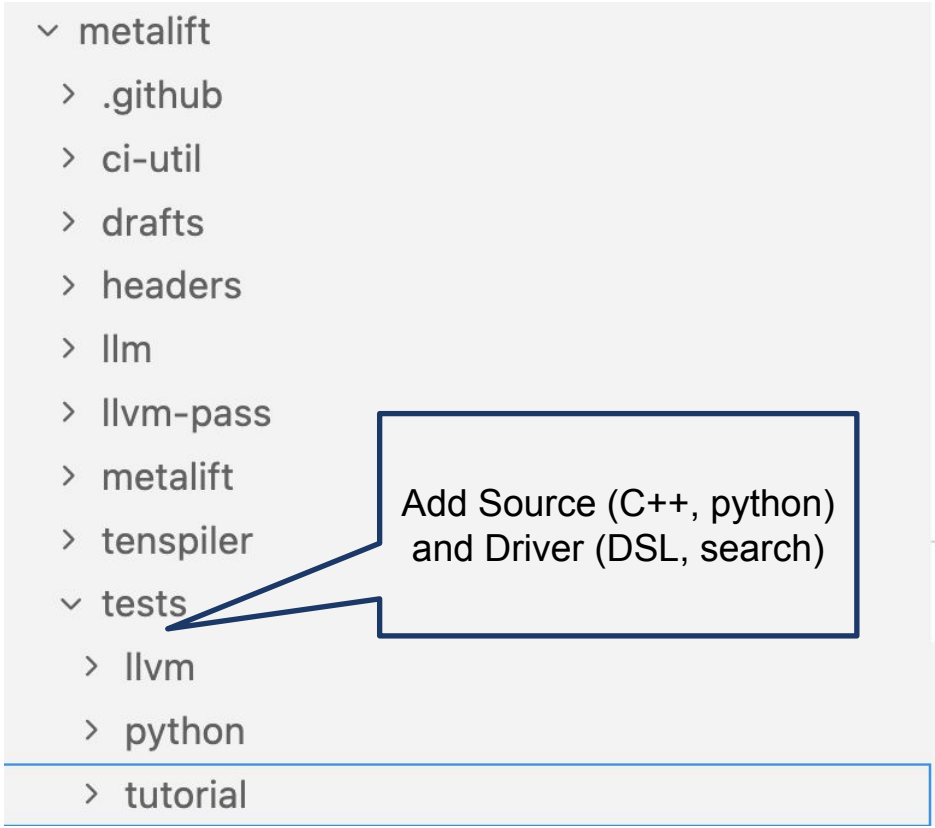
```
int conv_activation(int bias, int weight, int input)
{
    int result = bias + weight * input;
    if (result < 0) result = 0;
    return result;
}
```

Source Program

```
def fma(x, y, z):
    return x + y * z

def relu(x):
    if x > 0:
        return x
    else:
        return 0

def fused_fma_relu(x, y, z):
    val = x + y * z
    if val > 0:
        return val
    else:
        return 0
```

Target Language

Extending LLMLift



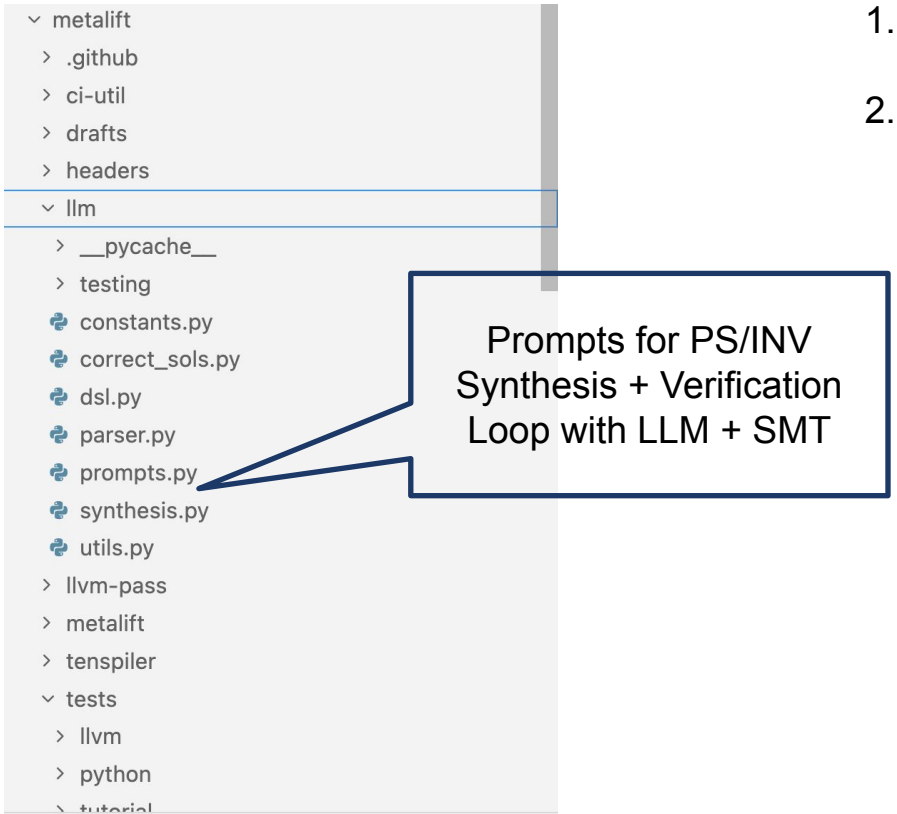
Source

1. Standalone C/C++/Python

Driver File

1. Semantics of the target language DSL/ISA operators
2. Call the search function (**run_synthesis_for_cc**) → compile to LLVM, generate specification, invoke LLMs (PS, INV), invoke SMT solver (CVC5)

Extending LLMLift



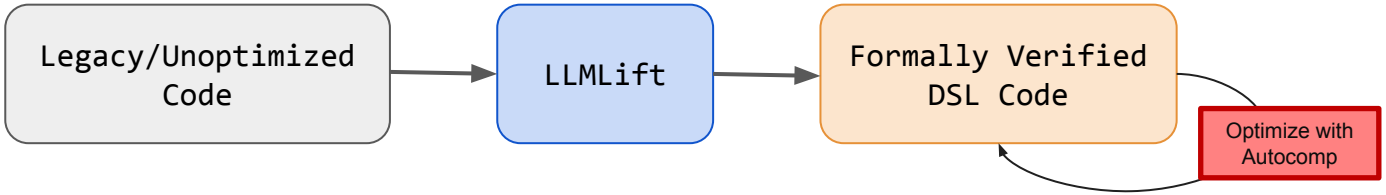
LLM + Verification

1. We have support for OpenAI, Claude, Gemini (Bring your own key!)
2. For Verification: SMT (CVC5), Bounded Verification (Rosette)

https://docs.racket-lang.org/rosette-guide/index.html

Summary

1. Verified Lifting-based Compilers - ***No pattern-matching rules!***
2. LLMs + Formal Verification 🔥
   a. LLMs - ***Fast Search***
   b. Formal Verification - ***Prevents Hallucinations***
3. MetaLift/LLMLift - Easily ***extensible*** and quickly ***iterate*** on your DSL/ISA



Contact : sahilbhatia@berkeley.edu

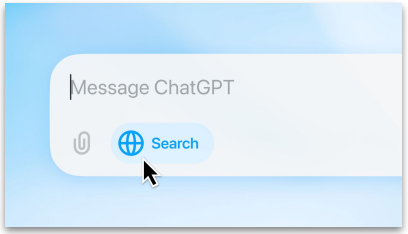
Autocomp + Trainium Tutorial

Charles Hong, Sahil Bhatia, Alvin Cheung, Yakun Sophia Shao

Berkeley
UNIVERSITY OF CALIFORNIA

SLICE

AI is everywhere



Chatbots

Robots

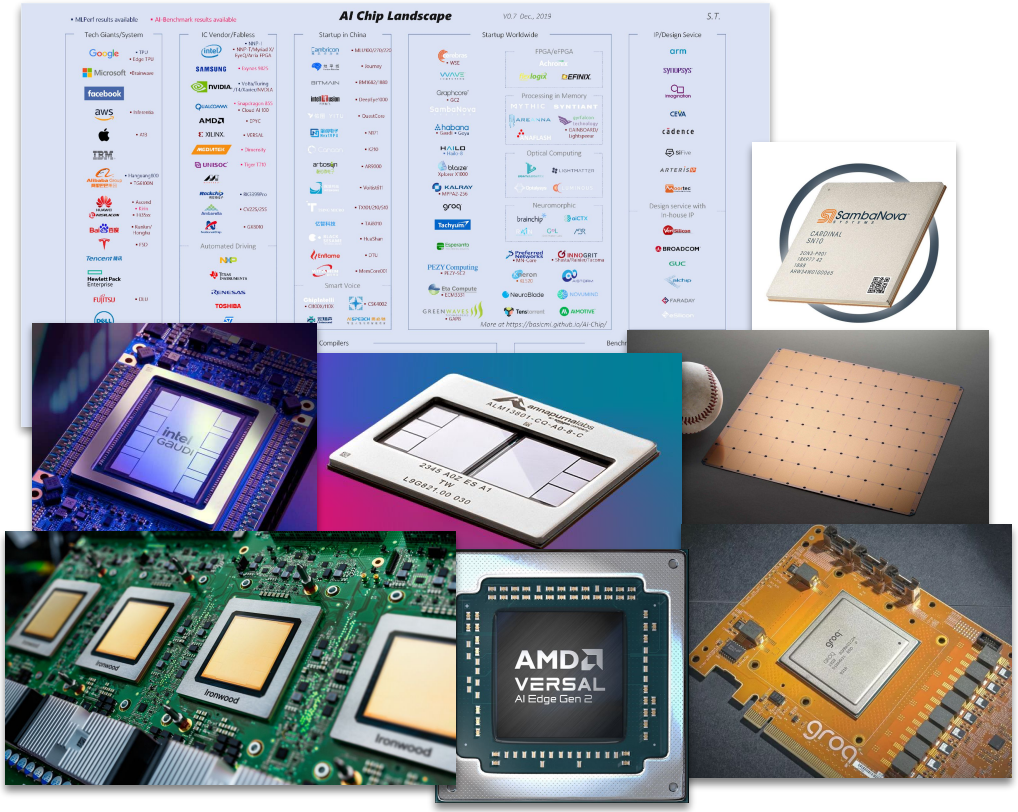
Mobile phones

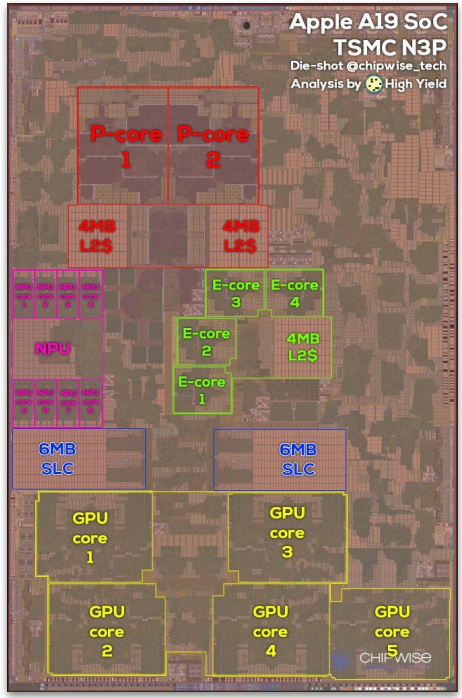
Augmented Reality

Autonomous Vehicles

Genomics

AI chips are everywhere





The real AI chip landscape

The Big Players

🔒 The Moat: Software (CUDA)

NVIDIA wins thanks to **decades of hand-tuning** and **customer lock-in** to tooling and infrastructure. Only deep vertical integration (Google) can compete.

The Challengers

+ many more

⚠️ The Pain Point

Can be technically superior (faster/cheaper per watt), but fail to match software user experience in practice.
“Hardware rich, software poor.”

Why is AI software so difficult?

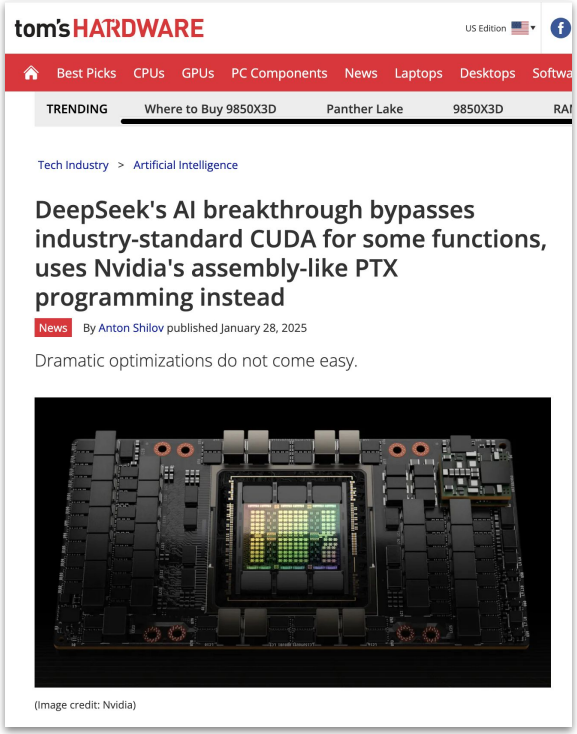
Traditional Compilers

- Decades of development
- CPU ISA and micro-architecture rarely change
- Well-established boundary between HW/SW

AI Software

- Software is a moving target (MLPs→CNNs→LLMs→?)
- Hardware is a moving target (Ampere→Hopper→Blackwell)
- No clear ISA boundary
- HW/SW co-design is common (quantization, sparsity)

The end result: hand-written, low-level code



Disclaimer: No PTX in this presentation.

AI Software

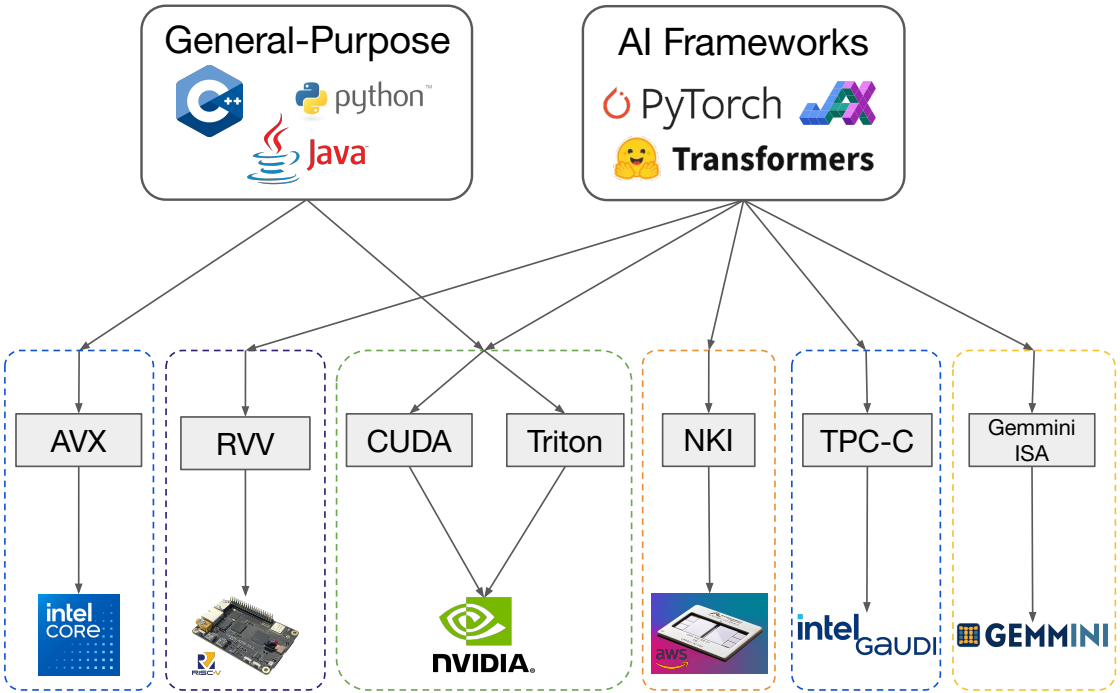
- Hardware is a moving target (Ampere→Hopper→Blackwell)
- No clear ISA boundary
- HW/SW co-design is common (quantization, sparsity)
- Relatively new, with requirements rapidly changing (LLMs!)

Lots of sources, lots of targets…

High-Level Languages

Low-Level Performance Programming

Heterogeneous Accelerators

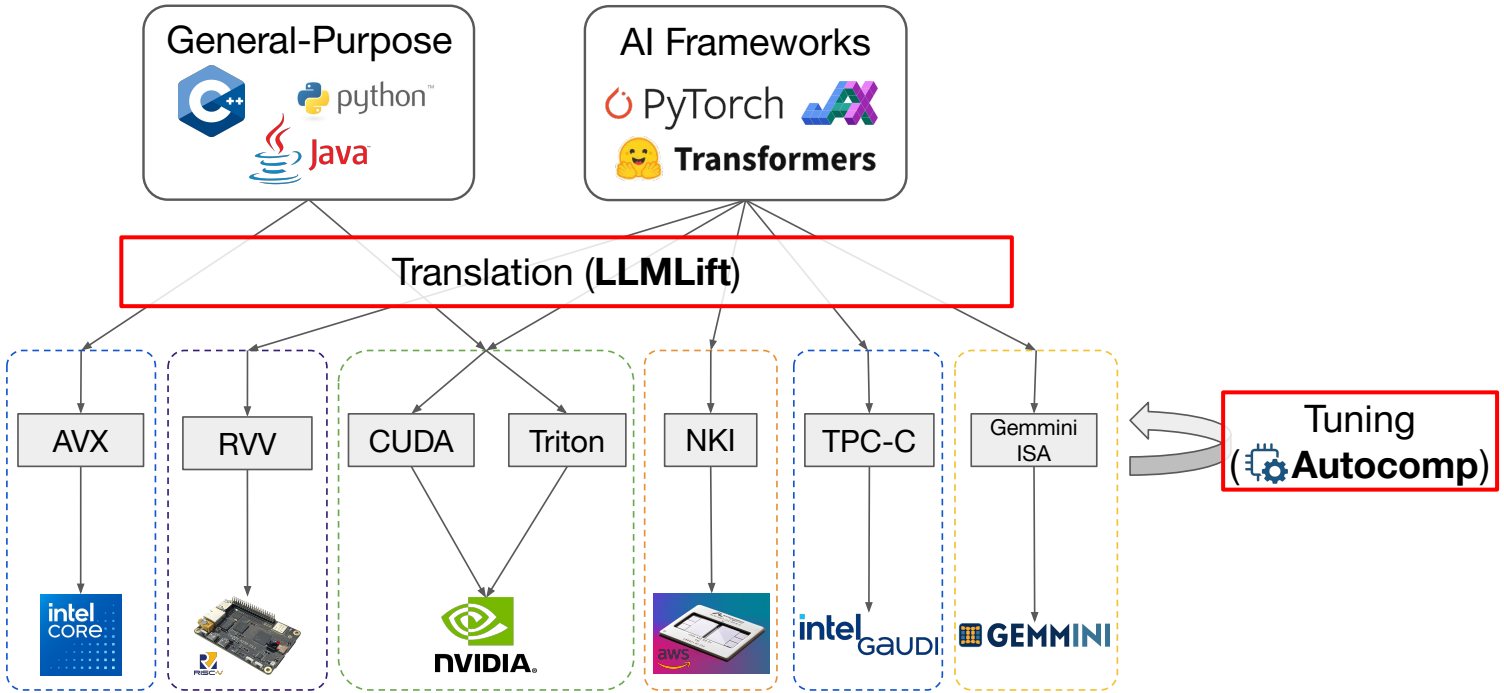


Our solutions: LLMLift + Autocomp

High-Level Languages

Low-Level Performance Programming

Heterogeneous Accelerators



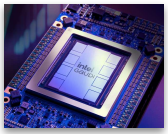
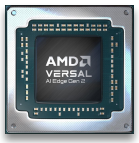
Hundreds of models × dozens of hardware backends

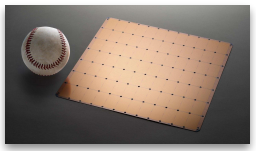
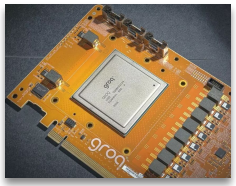
= a combinatorial problem!

OpenAI LLaMA deepseek Qwen Gemma Mistral AI KIMI MINIMAX

×



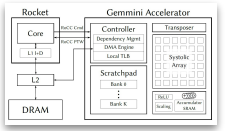




Autocomp: The kernel optimizer for any AI hardware

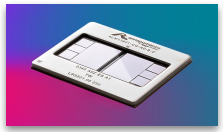
The old way: Adapting traditional compilers to new hardware requires massive engineering effort

The Autocomp way: Target a new hardware platform just by modifying a structured prompt

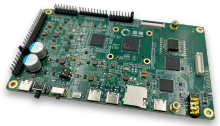
Just change:
- Accelerator ISA
- Optimization Menu
- Generation Rules



Gemmini
Academic Accelerator



Trainium 1
Industry Accelerator



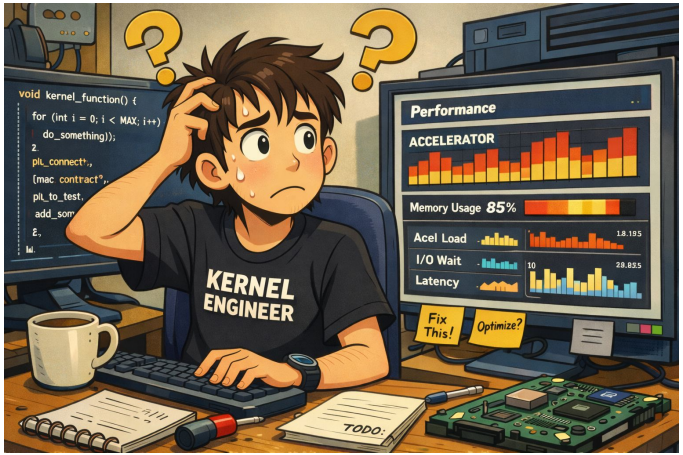
Canaan K230
RVV Dev Board

NVIDIA L40S
GPU

Your Hardware
?

Why not LLMs?

1. LLM doesn’t know how to program **specialized accelerator ISA**
2. Code optimization is a highly challenging reasoning problem
3. Need **runtime feedback/profiling** data to know which optimizations to apply



The **Autocomp** Approach

Problem 1: LLM doesn't know how to program accelerator ISA.

Observation: DSLs can be taught in context (our ISLAD'24 paper).

Problem 2: Code optimization is a challenging reasoning problem.

Observation: Each individual optimization (tiling, double-buffering, etc.) is well-understood. Applying the right ones in the right order is the problem!

Problem 3: Need runtime feedback/profiling data to know which optimizations to apply.

Observation: Run code and provide profiling data in context. Search widely and throw out bad search paths.

In-Context Learning

- Early LLM days: we found that providing **both ISA+code examples** is critical to increasing % of functionally correct generated code

	pass@k		
	k=1	k=10	k=50
Zero-shot	0.33%	3.33%	16.7%
One-shot ICL	44.67%	84.42%	99.79%
One-shot ICL (NL-annotated)	46.0%	88.81%	99.98%
No ISA, One-shot ICL (NL-annotated)	1%	9.12%	29.29%

TABLE I
`gpt-4-turbo` TRANSLATED CODE CORRECTNESS FOR MATRIX-VECTOR MULTIPLICATIONS, WITH NATURAL LANGUAGE DESCRIPTIONS FOR FUNCTIONS IN THE INPUT CODE.

- Condensing ISA, hardware architecture info, code examples into reasonably sized prompt is a bit of an art. Currently working on turning this into a repeatable, automated process

Problem 1: LLM doesn't know how to program accelerator ISA.

Observation: DSLs can be taught in context (our ISLAD'24 paper).

Problem 2: Code optimization is a challenging reasoning problem.

Observation: Each individual optimization (tiling, double-buffering, etc.) is well-understood. Applying the right ones in the right order is the problem!

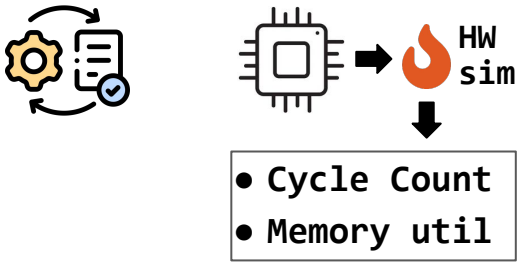
Problem 3: Need runtime feedback/profiling data to know which optimizations to apply.

Observation: Run code and provide profiling data in context. Search widely and throw out bad search paths.

Plan-then-Implement: 2-Phase Optimization Pass

Hardware Feedback:

Correctness + Performance



1. Plan Prompt

Accelerator <architecture> and <ISA>.
Original code: <code>
<opt menu> (a list of optimization strategies)
Select **one** and describe how to apply it (what variables/instructions/loops to apply to).

Sample!

2. Implement Prompt

Accelerator <architecture> and <ISA>.
Original code: <code>
Plan from phase 1: <plan>
Take the selected optimization, and implement it.

Sample!

Problem 1: LLM doesn't know how to program accelerator ISA.

Observation: DSLs can be taught in context (our ISLAD'24 paper).

Problem 2: Code optimization is a challenging reasoning problem.

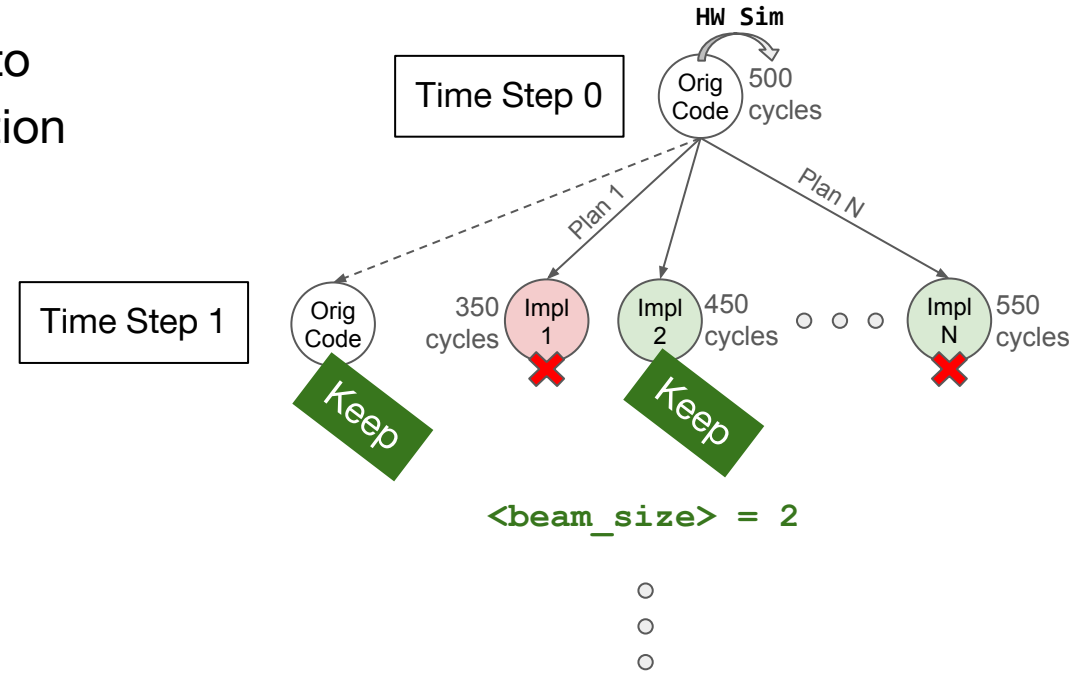
Observation: Each individual optimization (tiling, double-buffering, etc.) is well-understood. Applying the right ones in the right order is the problem!

Problem 3: Need runtime feedback/profiling data to know which optimizations to apply.

Observation: Run code and provide profiling data in context. Search widely and throw out bad search paths.

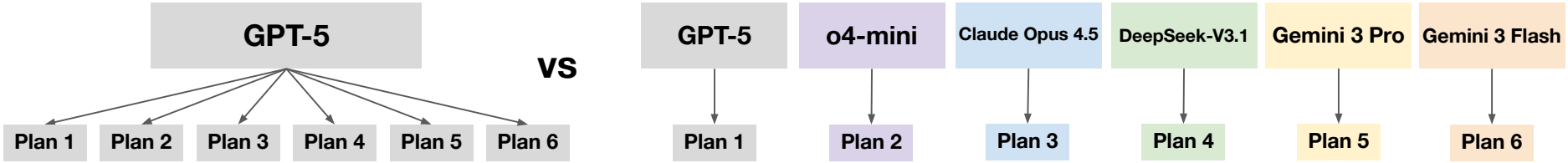
LLM-Integrated Beam Search

- Iteratively optimize
  - Go from correct solution to **faster, also correct** solution
- Use test cases to eliminate functionally incorrect code
- Keep `<beam_size>` best implementations at each time step



Increasing Plan/Code Diversity

- Sampling gives us different choices, but we noticed similar plans were often being regenerated
- **Optimization Menu Dropout:** Each time a plan is generated, randomly "drop out" some of the suggested optimizations
  - We used 70% probability of being dropped
- **LLM Ensembling**: When sampling plans/code, divide samples between multiple different models



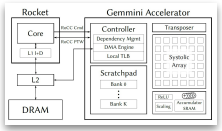
Experimental Results

Back to the motivation…

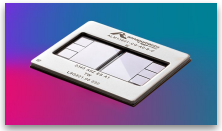
The old way: Adapting traditional compilers to new hardware requires massive engineering effort

The Autocomp way: Target a new hardware platform just by modifying a structured prompt

Just change:
- Accelerator ISA
- Optimization Menu
- Generation Rules



Gemmini
Academic Accelerator



Trainium 1
Industry Accelerator

Canaan K230
RVV Dev Board

NVIDIA L40S
GPU

Your Hardware
?

Evaluation: Gemmini

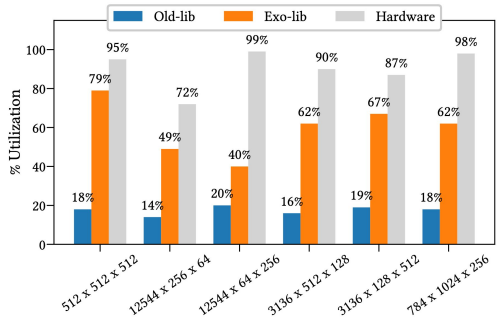
- Prior work used Exo scheduling DSL to **hand-optimize** Gemmini code
  - Gemmini: open-source tensor accelerator generator
- Let's see if we can beat **human experts** on **key workloads (GEMM/Conv)**!

Autocomp **vs.**

exo

in app.py

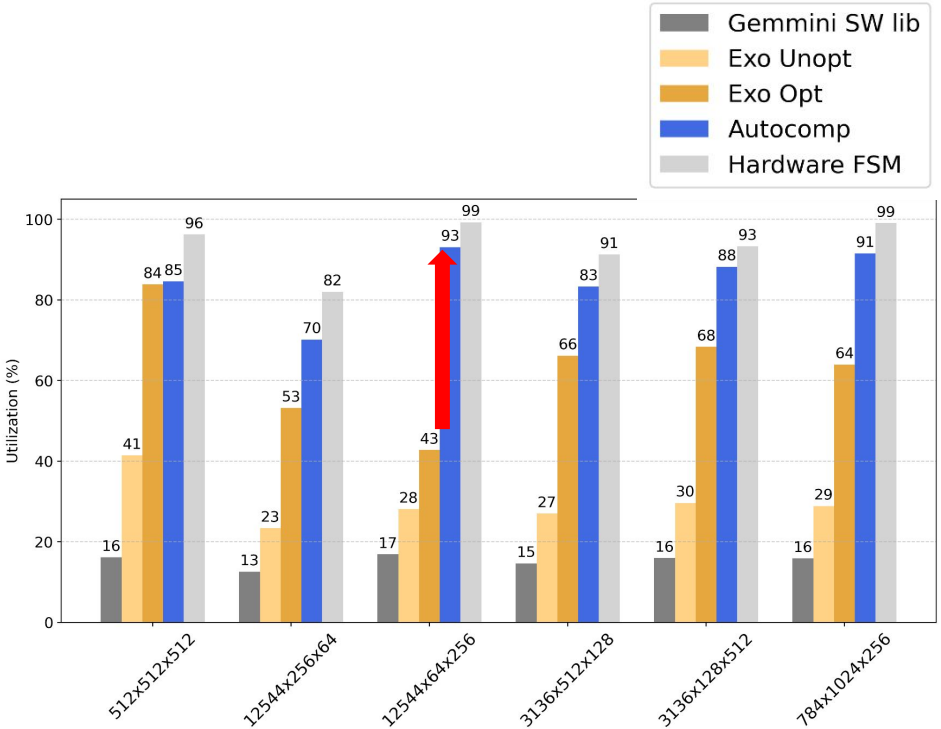
```
def gemm(...):
    res: R[...] @ ACCUMULATOR
    a  : R[...] @ SCRATCHPAD
    b  : R[...] @ SCRATCHPAD
    for io in seq(0, 8):
        for jo in seq(0, 8):
            ... # Load C to res
            for ko in seq(0, 8):
                # Load A to a
                for ii in seq(0, 16):
                    for ki in seq(0, 16):
                        a[...] = A[...]
                ... # Load B to b
                # Matmul of a and b
                for ii in seq(0, 16):
                    for ji in seq(0, 16):
                        for ki in seq(0, 16):
                            res[..]+=a[..]*b[..]
            ... # Store res to C
```



(a) MATMUL utilization (as a percentage of peak FLOPS). X axis labels are the size of matrices in N x M x K.

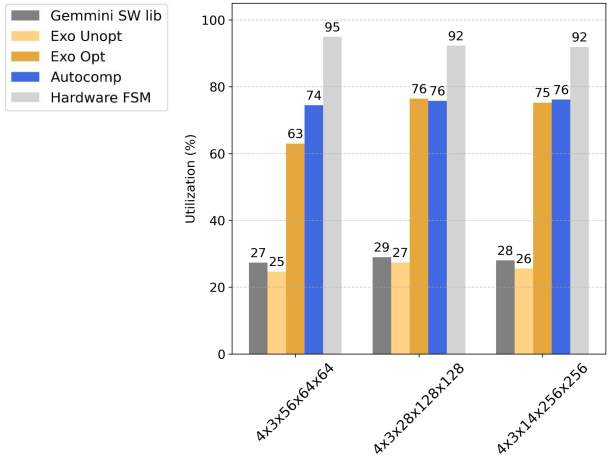
GEMM Optimization Results

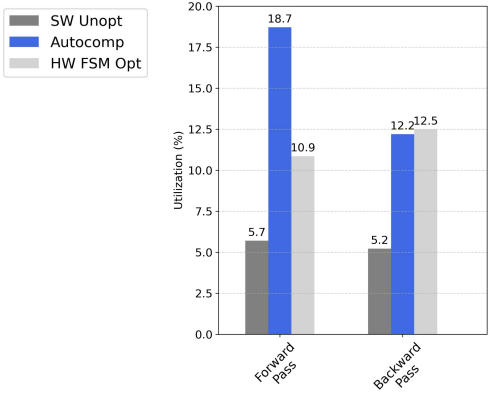
- **Highlights:**
  - **5.6x** speedup over Gemmini software library
  - **1.4x** speedup over previous best hand-optimized code
- We can explore much more of the search space than a human can



Evaluation: Convolution/TinyMPC

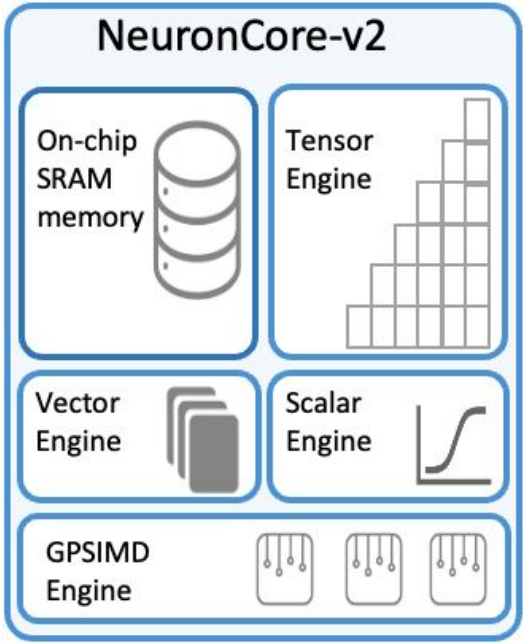
- Also optimized other workloads
  - Convolution
  - Robotics model-predictive control (sequence of small linear algebra kernels)
- See paper for details!



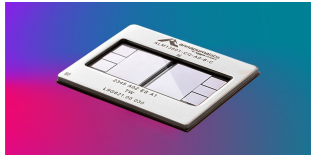


AWS Trainium

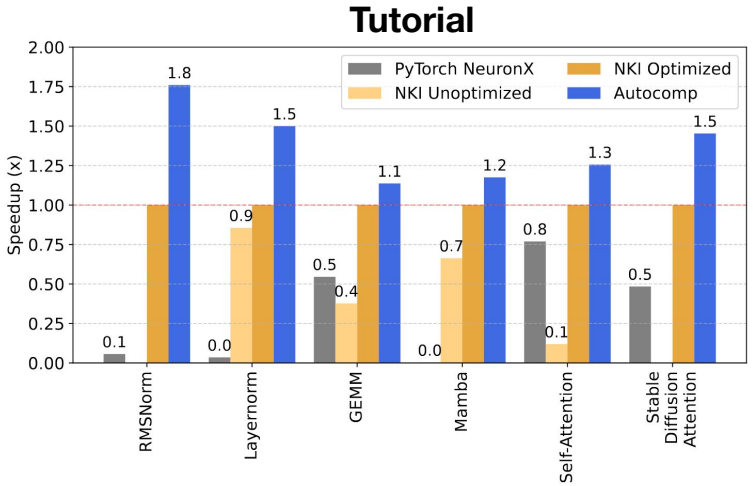
- Systolic array-based architecture with tensor engine, vector engine, scalar engine
- Kernels written in hardware specific DSL called Neuron Kernel Interface (NKI)
- The tutorial environment gives you access to a `trn1.2xlarge` instance

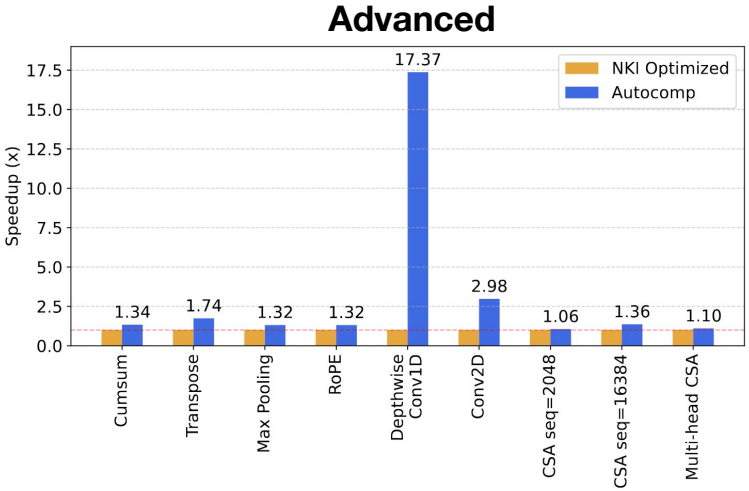


Evaluation: Trainium



- Benchmark: AWS's own Trainium NKI kernel examples
- Tutorial:
  - **1.36x** faster than NKI Optimized (starting from NKI Unoptimized)
  - **13.52x** faster than PyTorch
- Advanced (examples for production users):
  - **1.9x** faster than baseline

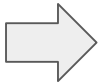




Trainium Conv1D

- Key optimization (~10x speedup): Add a constant-size inner loop to the main convolution loop
- Small/static loop bound seems to enable coalescing of tensor_reduce() operations (profiler summary shows reduced # of VectorEngine insts)
  - Coalescing can also be done manually

```
# convolution loop
for o in nl.affine_range(out_image_size):
    img_tile  = img_local[i_p, i_f_a + o]    # (C_T
    filt_tile = filt_psum[i_p, i_f_win]      # (C_T
    prod = nisa.tensor_tensor(img_tile,
                              filt_tile,
                              op=np.multiply)
    out_sb[i_p, o] = nisa.tensor_reduce(np.add,
                                        prod,
                                        axis=[1])
```

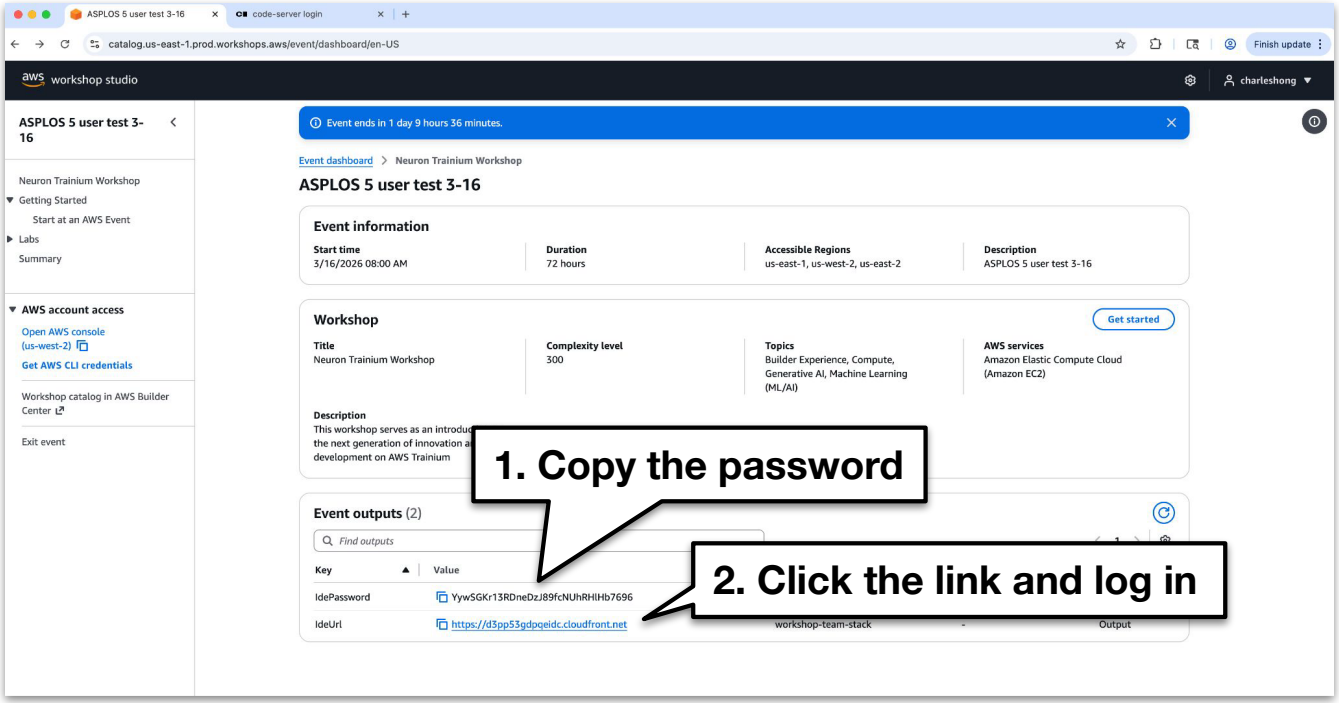


```
# --- Phase B + Phase C (optimized): block the big 1D output dim ------
# We tile the out_image_size dimension into fixed-size blocks.
OUT_TILE = 64
NUM_FULL_BLOCKS = out_image_size // OUT_TILE

# Process full blocks of OUT_TILE outputs
for b in nl.affine_range(NUM_FULL_BLOCKS):
    base_out = b * OUT_TILE
    for o in nl.affine_range(OUT_TILE):
        # identical to original per-output compute, now grouped by blocks
        img_tile  = img_local[i_p, i_f_a + (base_out + o)]   # (C_TILE_SIZE
        filt_tile = filt_psum[i_p, i_f_win]                  # (C_TILE_SIZE
        prod = nisa.tensor_tensor(img_tile,
                                  filt_tile,
                                  op=np.multiply)
        out_sb[i_p, base_out + o] = nisa.tensor_reduce(np.add,
                                                       prod,
                                                       axis=[1])
```

See our blog post!

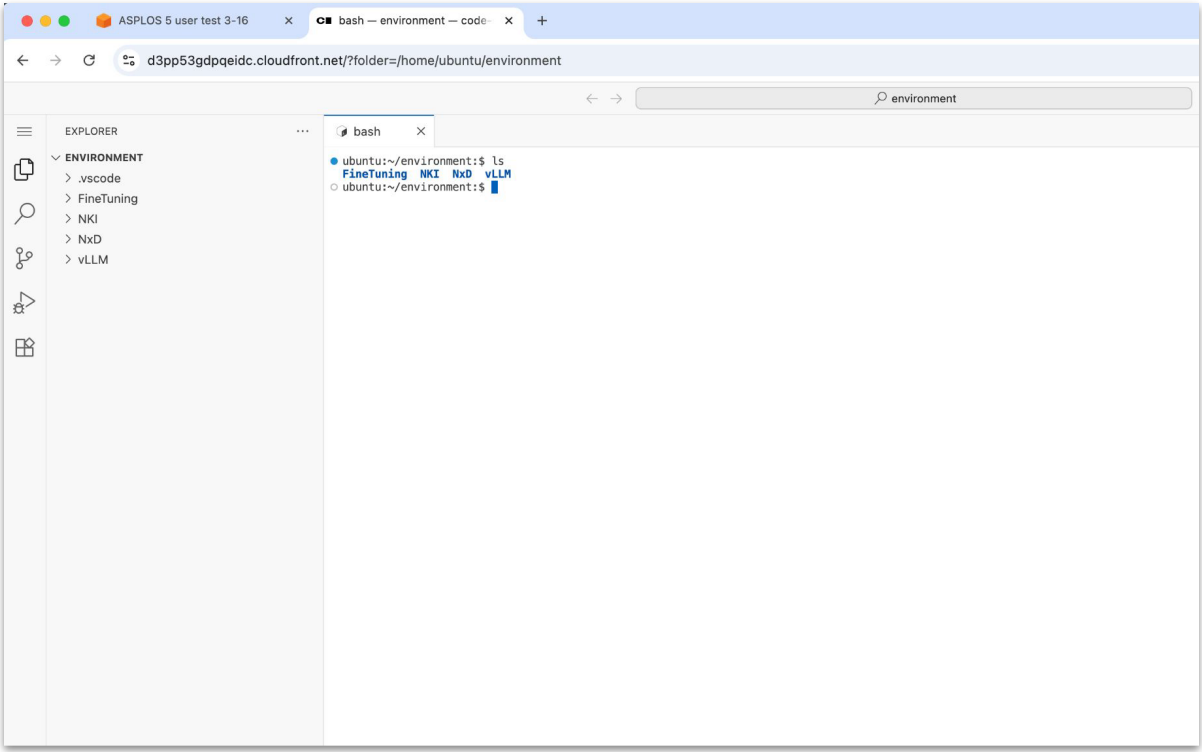
Hands-On: Getting Started

AWS Tutorial Environment (if you weren't here earlier)



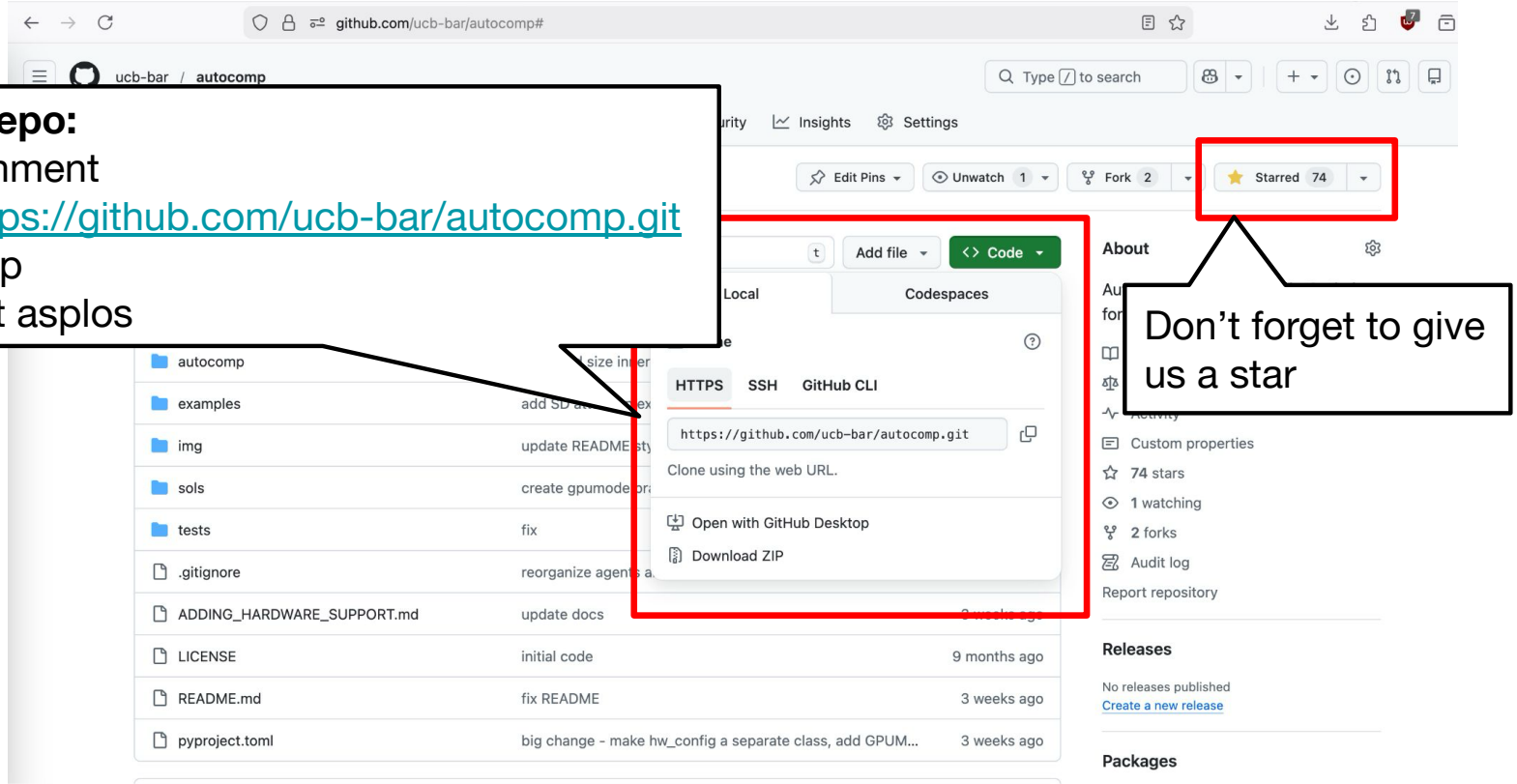
AWS Tutorial Environment

Looks something like this:



Clone Autocomp into the instance

Clone the repo:
cd ~/environment
git clone https://github.com/ucb-bar/autocomp.git
cd autocomp
git checkout asplos



Autocomp repo organization

autocomp/

- **autocomp/**
  - **agents/** - Hardware-specific agents (Saturn, Gemmini, Trainium, CUDA)
  - **backend/** - Hardware-specific evaluation scripts
  - **hw_config/** - Defines hardware configuration (e.g. Trainium instance type)
  - **search/** - Logic for Autocomp beam search
  - **common/** - LLM utils, logging, etc.
- sols/
- tests/

Autocomp repo organization

autocomp/

- autocomp/
  - agents/
  - backend/
  - hw_config/
  - search/
  - common/
- **sols/** - The actual code to be optimized (used by search/ and backend/)
- **tests/** - Test harnesses specific to individual sols (used by backend/)

Search

Running Search

autocomp/autocomp/search/search.py

bash autocomp | search.py | 0_conv1d_ref.py U | 0_conv1d_test.py

autocomp > autocomp > search > search.py > main

```python
def main():
    # Select evaluation backend, LLM agent, and hardware config
    backend_name = "trn"  # Options: "gemmini", "trn", "tpu", "kernelbench", "gpumode"
    agent_name = "built:trn-nki1"  # Options: "gemmini", "trn", "cuda", "built:<name>", or a path to a built agent (for TPU v6e, use built:tpu-v6e)
    simulator = None # "firesim" or "spike" if backend_name == "gemmini"; "gpumode-local" or "gpumode-cli" if backend_name == "gpumode"
    # Hardware configuration
    hw_config = TrnHardwareConfig("trn1.2xlarge")
    # Examples:
    # hw_config = TrnHardwareConfig("trn1.2xlarge")
    # hw_config = GemminiHardwareConfig(pe_dim=16, spad_size_kb=256, acc_size_kb=64)
    # hw_config = CudaHardwareConfig("NVIDIA L40S", "2.5.0", "12.4")
    # hw_config = TpuHardwareConfig("v6e-1")

    # Models are specified as "provider::model"
    # Valid providers are "openai", "anthropic", "together", "aws", "gcp", "vllm"
    # If no provider is specified, the provider is inferred from the model name
    models = ["aws::us.anthropic.claude-opus-4-5-20251101-v1:0", "aws::zai.glm-4.7", "aws::deepseek.v3.2", "aws::moonshotai.kimi-k2.5"] # Models for planning
    code_models = None # Models for code implementation (None means use same as planning models)
    metric = "latency"
    search_strategy = "beam"
    iterations = 3
    prob_type = "trn-tutorial" # see README.md or sols directory for available problems
    prob_id = 1
```

Install Autocomp package and dependencies

```
> source /opt/aws_neuronx_venv_pytorch_latest/bin/activate
> cd ~/environment/autocomp
> pip install -e .
```

If you weren't here earlier:
cd ~/environment
git clone https://github.com/ucb-bar/autocomp.git
cd autocomp
git checkout asplos

Env: https://tinyurl.com/asplos2026

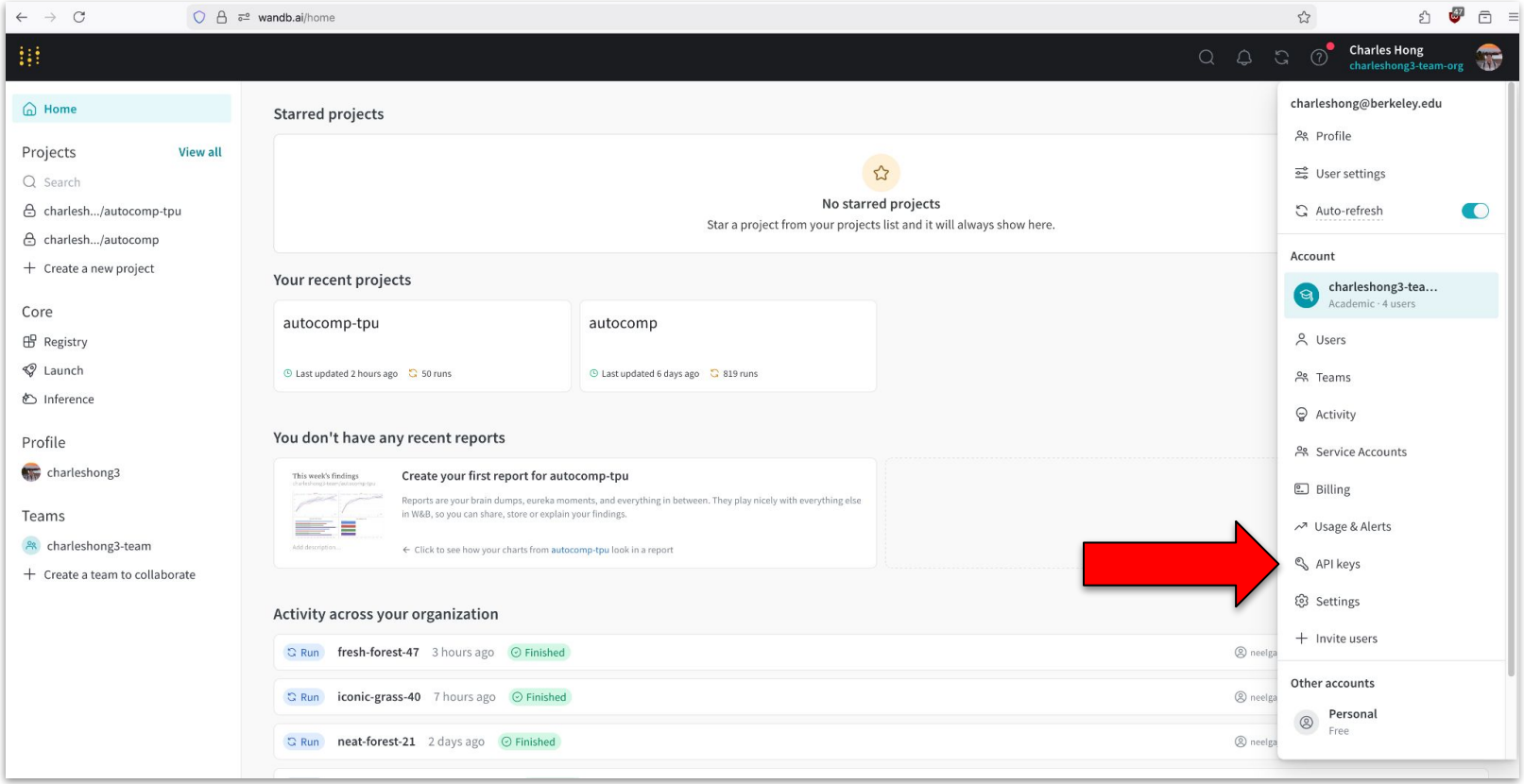
Run search

```
> python autocomp/search/search.py
```

Enter your W&B API key if you have an account (or make an account, it's quick)

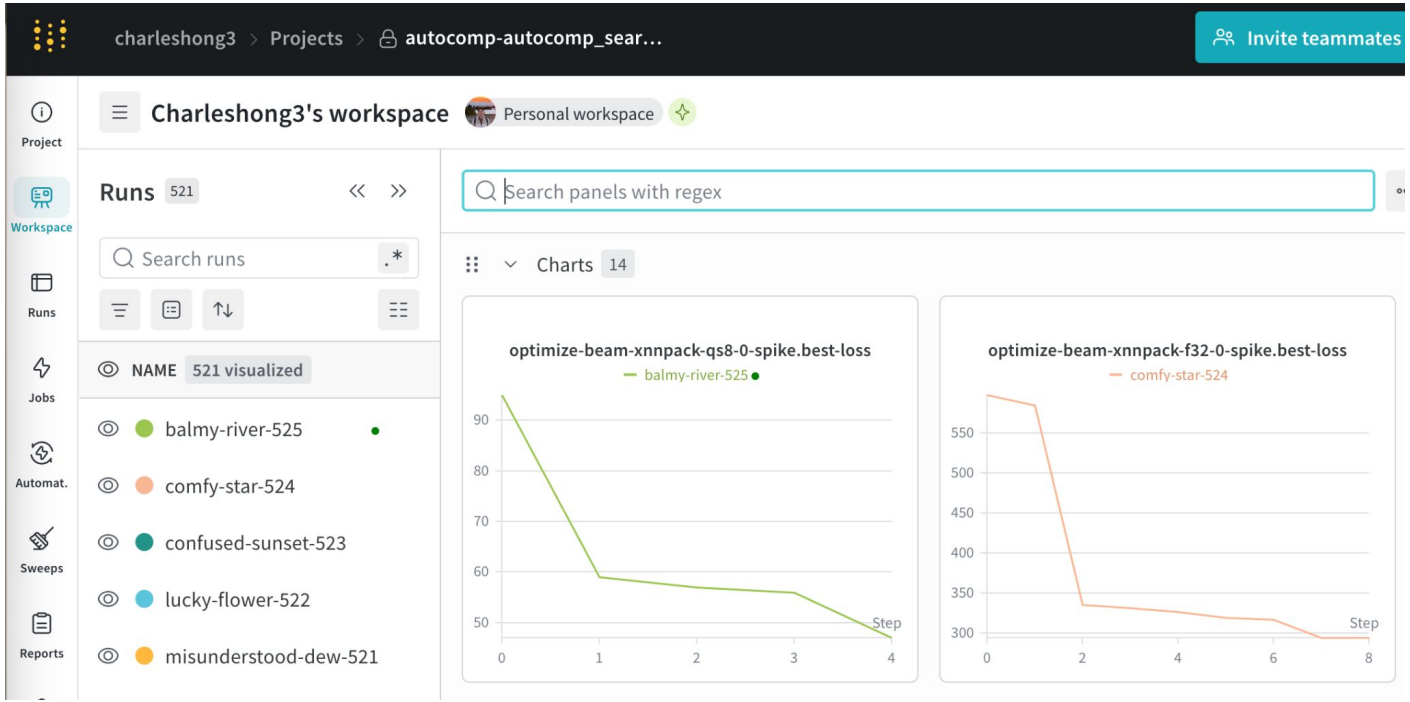
```
ubuntu:~/environment/autocomp:$ python autocomp/search/search.py
[2026-03-18 06:06:04,913 INFO llm_utils.py:62 <module>] Keys unavailable: OPENAI_API_KEY, ANTHROPIC_API_KEY, TOGETHER_API_KEY, AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, GOOGL
[2026-03-18 06:06:04,975 INFO search.py:936 main] Output directory: output/trn_trn-tutorial_1_beam_iters8_Trainium_trn1.2xlarge_us-4-5-20251101-v1:0_aws::zai.glm-4.7_aws::dee
b4_score1_ms1_fgisa1_ex0.25
[2026-03-18 06:06:05,485 INFO trn_eval.py:467 _try_combined_evaluation] Phase 1: compiling ref + 1 implementations in parallel
[2026-03-18 06:06:21,324 INFO trn_eval.py:266 _compile_impls_parallel] Compile ref: OK
[2026-03-18 06:07:21,992 INFO trn_eval.py:266 _compile_impls_parallel] Compile 0: OK
[2026-03-18 06:07:21,992 INFO trn_eval.py:474 _try_combined_evaluation] Phase 1 complete: ref=OK, 1/1 implementation NEFFs saved
[2026-03-18 06:07:21,992 INFO trn_eval.py:482 _try_combined_evaluation] Phase 2: correctness + benchmark via libnrt
[2026-03-18 06:07:34,106 INFO trn_eval.py:513 _try_combined_evaluation] Code 0 latency: 2.425
[2026-03-18 06:07:34,107 INFO search.py:236 __init__] Initial code scores:
[2026-03-18 06:07:34,107 INFO search.py:238 __init__] 2.425
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
wandb: Enter your choice: 2
wandb: You chose 'Use an existing W&B account'
wandb: Logging into https://api.wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: Create a new API key at: https://wandb.ai/authorize?ref=models
wandb: Store your API key securely and do not share it.
wandb: Paste your API key and hit enter:
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /home/ubuntu/.netrc
wandb: Currently logged in as: charleshong3 to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
wandb: Tracking run with wandb version 0.25.1
wandb: Run data is saved locally in /home/ubuntu/environment/autocomp/wandb/run-20260318_060804-5xhqy8hr
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run super-pond-51
wandb: ⭐ View project at https://wandb.ai/charleshong3-team/autocomp-tpu
```

W&B API Key

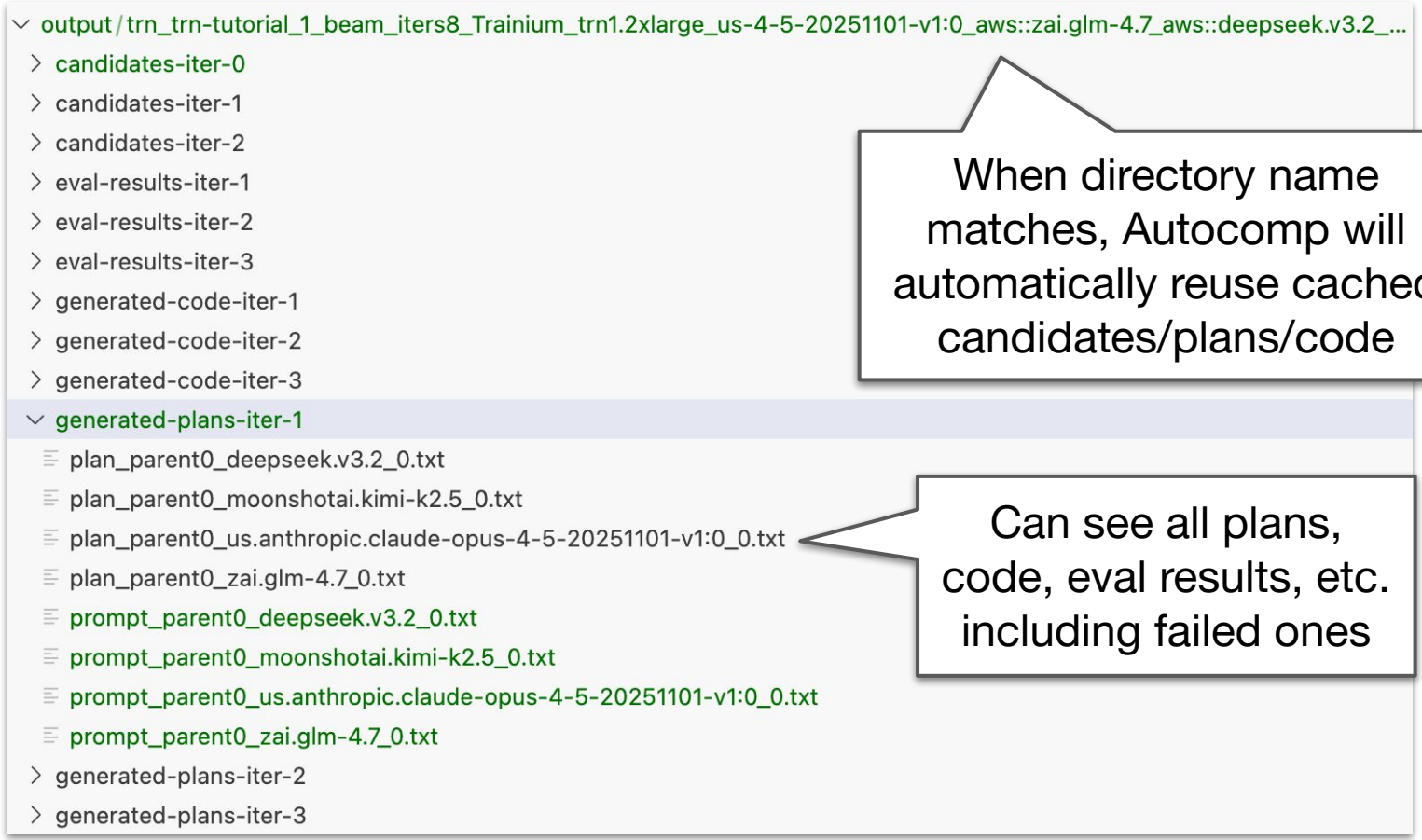

wandb.ai/home
Charles Hong
charleshong3-team-org
Home
Projects
View all
Search
charlesh.../autocomp-tpu
charlesh.../autocomp
Create a new project
Core
Registry
Launch
Inference
Profile
charleshong3
Teams
charleshong3-team
Create a team to collaborate
Starred projects
No starred projects
Star a project from your projects list and it will always show here.
Your recent projects
autocomp-tpu
Last updated 2 hours ago
50 runs
autocomp
Last updated 6 days ago
819 runs
You don't have any recent reports
Create your first report for autocomp-tpu
Reports are your brain dumps, eureka moments, and everything in between. They play nicely with everything else in W&B, so you can share, store or explain your findings.
Click to see how your charts from autocomp-tpu look in a report
Activity across your organization
Run fresh-forest-47 3 hours ago Finished
Run iconic-grass-40 7 hours ago Finished
Run neat-forest-21 2 days ago Finished
charleshong@berkeley.edu
Profile
User settings
Auto-refresh
Account
charleshong3-tea...
Academic · 4 users
Users
Teams
Activity
Service Accounts
Billing
Usage & Alerts
API keys
Settings
Invite users
Other accounts
Personal
Free

wandb

Lets you monitor status of ongoing runs and keep track of past runs



Autocomp artifacts (located at `<cwd>/output`)



CodeCandidate object (`output/candidates-iter-<i>`)

Stores full history of each candidate in beam, including:

- Ancestor candidates
- Latency
- Plan
- Code
- Models used to generate plan/code
- stdout/stderr (if returned by evaluation backend)

```
            # Load gamma and beta per-chunk
            i_p_param = nl.arange(1)[:, None]
            gamma_chunk = nl.load(gamma_reshaped[i_p_param, col_start + i_f_chunk])
            beta_chunk = nl.load(beta_reshaped[i_p_param, col_start + i_f_chunk])

            # Normalize: (x - mean) * inv_std
            centered = nl.subtract(input_chunk, mean_tile, mask=combined_mask)
            normalized = nl.multiply(centered, inv_std, mask=combined_mask)

            # Apply affine transformation: gamma * normalized + beta
            scaled = nl.multiply(normalized, gamma_chunk, mask=combined_mask)
            output_chunk = nl.add(scaled, beta_chunk, mask=combined_mask)

            # Store result
            nl.store(output_tensor[row_offset + i_p, col_start + i_f_chunk],
                     value=output_chunk, mask=combined_mask)

    return output_tensor
''',
score=1.446,
hw_feedback=[],
plan_gen_model='moonshotai.kimi-k2.5',
code_gen_model='us.anthropic.claude-opus-4-5-20251101-v1:0',
stdout='Latency: 1.446 ms (P99)\n',
stderr=''),
plan='''## Analysis of the Code

Looking at the current implementation, I can identify several inefficiencies:

1. **Redundant gamma/beta loads**: Inside the normalization loop, `gamma_chunk` and `beta_chunk` are loaded fr

2. **Multiple passes over data**: The code loads input data, stores it to a buffer, then reads it back for nor

3. **Unrolling structure is awkward**: The manual unrolling with `for unroll_idx in range(2)` creates a Python

4. **Gamma and beta are loaded per-chunk inside the row loop**: These parameters are the same for all rows, so

## Selected Optimization: #6 - Increase reuse by keeping data in SBUF across outer loop iterations

**Plan:**

The key insight is that `gamma_vector` and `beta_vector` are 1D parameters of shape `[num_cols]` that are **co
```

trn-tutorial problem 1

- Layernorm
- Ran as a quick test for 3 iterations

bash autocomp | 1_layernorm_ref.py | search.py M

autocomp > sols > trn-tutorial > 1_layernorm_ref.py > ...

```python
@nki.jit
def test(input_tensor, epsilon, gamma_vector, beta_vector):
  """Computes LayerNorm.
  """
  output_tensor = nl.ndarray(input_tensor.shape, dtype=input_tensor.dtype,
                             buffer=nl.shared_hbm)

  # Ensure that the shapes of tensors match
  assert input_tensor.shape[1] == gamma_vector.shape[0] == beta_vector.shape[0]

  # Generate tile indices for loading/storing data
  i_p_io = nl.arange(nl.tile_size.pmax)[:, None]
  i_f_io = nl.arange(input_tensor.shape[1])[None, :]
  i_p_param = nl.arange(1)[:, None]

  # Number of rows in the input tensor
  num_rows = input_tensor.shape[0]

  # Load gamma and beta, which will be reused across rows/tiles of input_tensor
  gamma_sb = nl.load(gamma_vector.reshape((1, gamma_vector.shape[0]))[i_p_param, i_f_io])
  beta_sb = nl.load(beta_vector.reshape((1, beta_vector.shape[0]))[i_p_param, i_f_io])

  # Broadcast the gamma and beta to match the dimensions of the tiles
  gamma_sb_bcast = gamma_sb.broadcast_to((nl.tile_size.pmax, gamma_vector.shape[0]))
  beta_sb_bcast = beta_sb.broadcast_to((nl.tile_size.pmax, beta_vector.shape[0]))

  # Tile partition dimension of the input tensor by nl.tile_size.pmax
  for i in nl.affine_range(math.ceil(input_tensor.shape[0]/nl.tile_size.pmax)):
    # Load input tile
    input_sb = nl.load(input_tensor[i * nl.tile_size.pmax + i_p_io, i_f_io],
                       mask=(i * nl.tile_size.pmax + i_p_io < num_rows))

    # Compute mean and variance
    mean = nl.mean(input_sb, axis=1)
    # Trick to calculate var with mean: mean(x^2) - mean(x)^2
    var = nl.mean(nl.square(input_sb), axis=1) - mean * mean

    # Normalize the input by shifting with the mean
    # and scaling with rsqrt of variance and epsilon
    shift_scale_tensor = (input_sb - mean) * nl.rsqrt(var + epsilon)

    # Scale the normalized tile using gamma and add beta
    output_sb = shift_scale_tensor * gamma_sb_bcast + beta_sb_bcast

    nl.store(output_tensor[i * nl.tile_size.pmax + i_p_io, i_f_io], value=output_sb,
             mask=(i * nl.tile_size.pmax + i_p_io < num_rows))

```

Conclusion

We covered:

- How to build an agent
- How to build an evaluation backend
- How to set search parameters and run search
- How to look at search results using local artifacts and W&B
- How to add and run new workloads

Thank you for attending!