Verified Code Translation Tutorial

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

Agenda

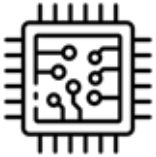
1. Overview of Verified Code Translation
2. Tools: **MetaLift** + **LLMLift**
3. Hands-on Session:
    a. Translate sequential program to DSL
    b. Translate loopy program to NKI (Trainium)
    c. Lifting Your Own Kernel (extending the DSL)

Domain Specific Languages (DSL)

PyTorch

MLX

- Domain-Specific Optimizations
- Improved Readability and Maintainability



Legacy/Unoptimized Code

```
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

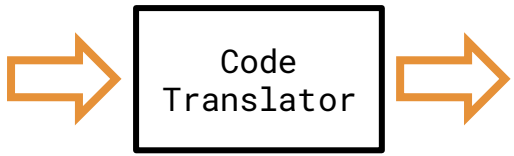

```
out = o* a + (32 - b) * b
```

Big Data


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

Lifting Unoptimized Code

```
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);
  }
}
```




```
torch.sub(
  torch.add(b,a),
  torch.div(
    torch.mul(b,a),
    255
  )
)
```

PyTorch

```
tf.subtract(
  tf.add(b,a),
  tf.divide(
    tf.multiply(b,a),
    255
  )
)
```

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].s
      int pixel = b[i][j] + a[
        - (b[i][j] * a[i][j])
      row_vec.push_back(pixel)
      }
    out.push_back(row_vec);
  }
}
```

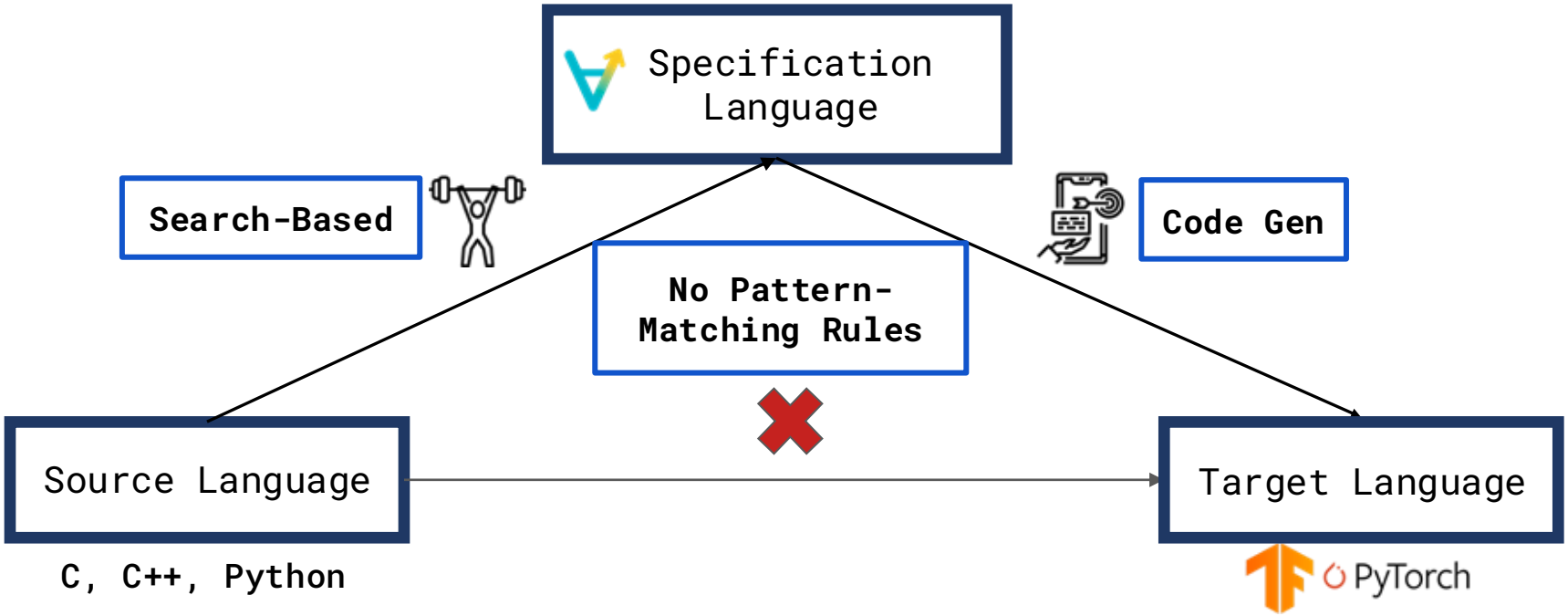
- Brittle
- Difficult to Get Right
- Hard to Maintain

```
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)
```

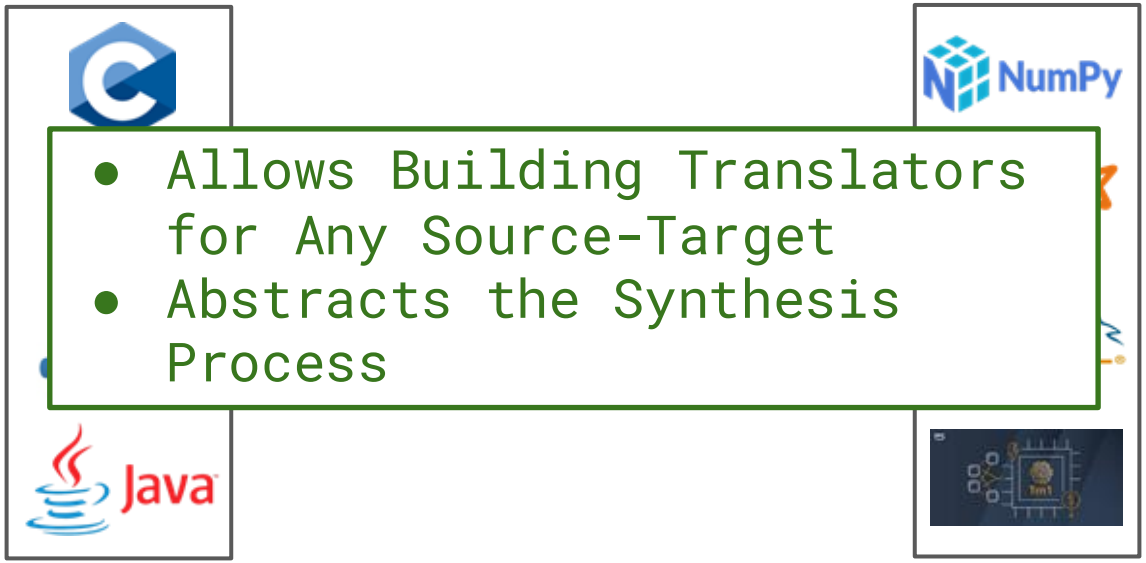
Prior Approaches: Verified Lifting-Based Tools



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?

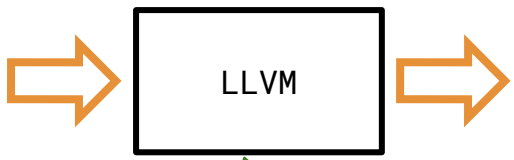


- Allows Building Translators for Any Source-Target
- Abstracts the Synthesis Process

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);
  }
}
```

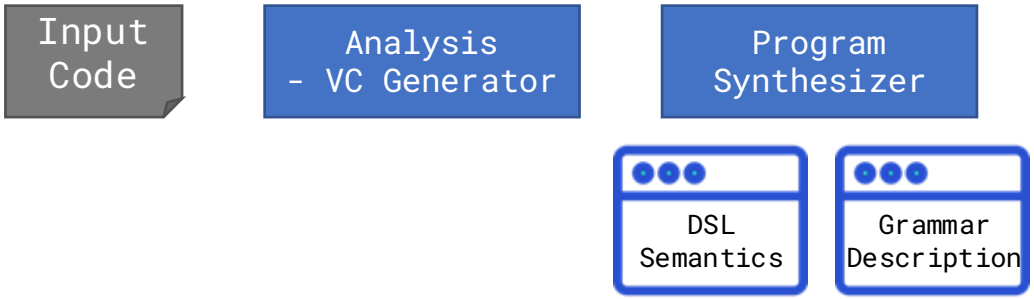


Supports Many
Source Languages

Specification

source == target (**PS**)

Loop:
source == target (**PS + Inv**)



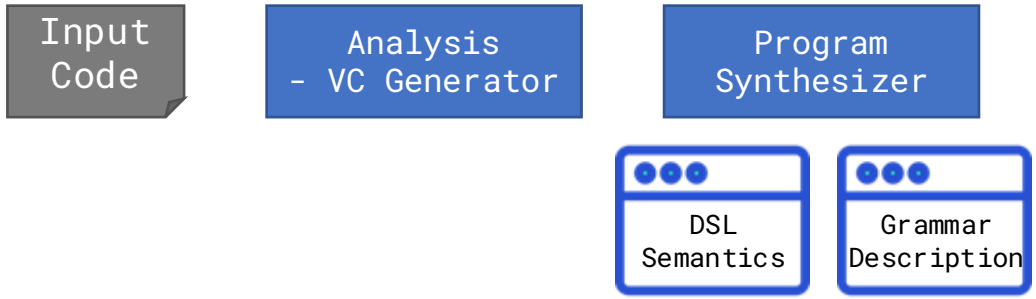

```python
def mat_add(
  matrix_x: List[List[int]],
  matrix_y: List[List[int]]
) -> List[List[int]]:
  return (
    []
    if len(matrix_x) < 1 or not len(matrix_x) == len(matrix_y)
    else [
      vec_add(matrix_x[0], matrix_y[0]),
      *mat_add(matrix_x[1:], matrix_y[1:]),
    ]
  )
```

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

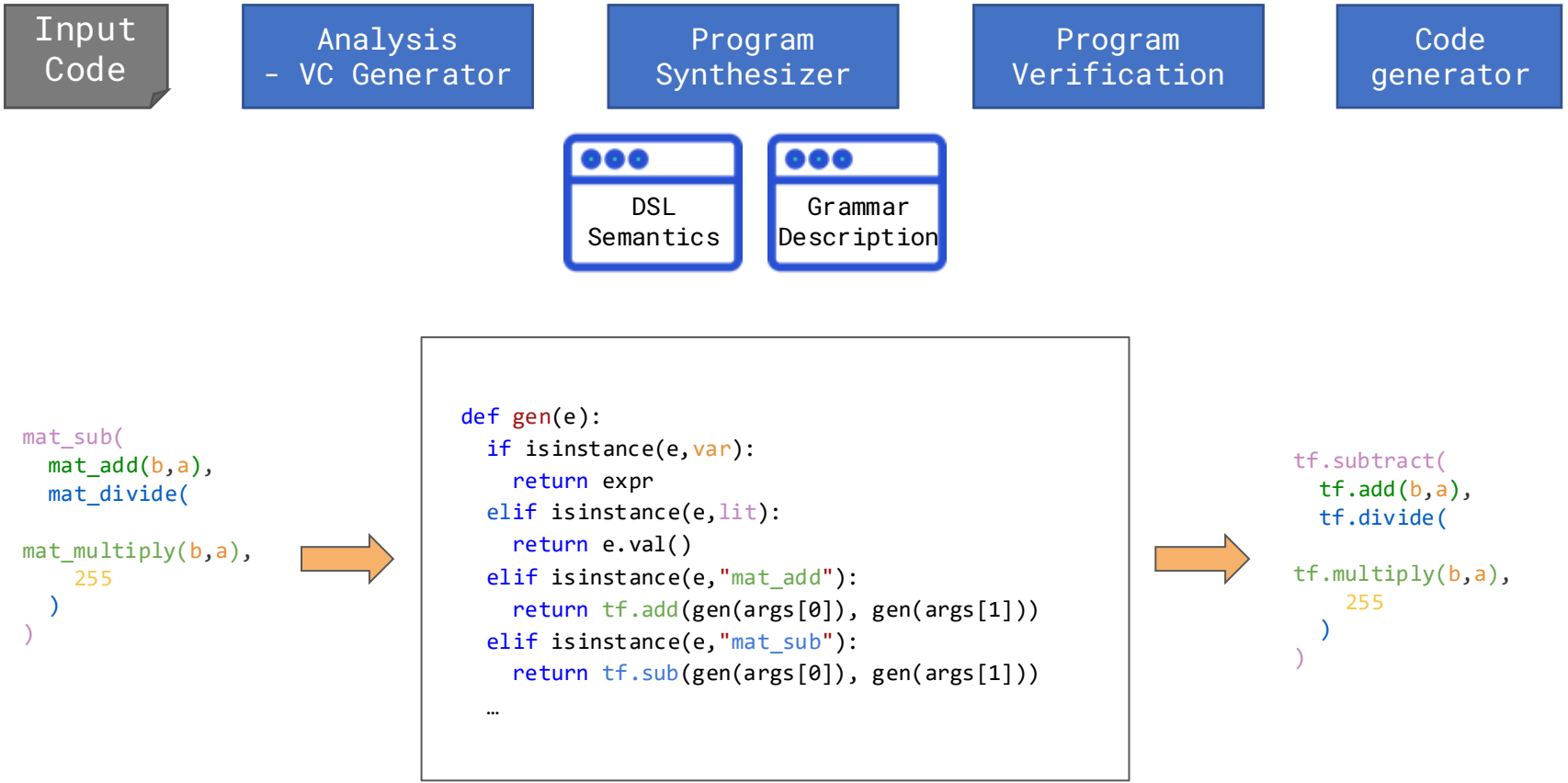
Grammar Description

```
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 / Spark	40/45	< ~90%
C	5/5	< 500 LOC
NumPy		

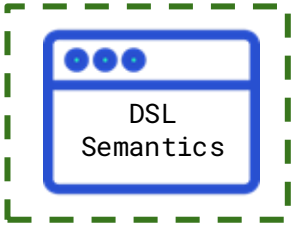
- Demonstrates Generality of MetaLift
- Built Transpilers for 3 Different Application Domains

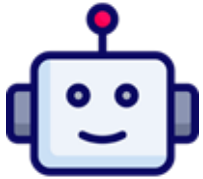
LLMLift: Verified Code Translation with LLMs

Bottlenecks in MetaLift

Manual Effort

Domain Specific Knowledge





LLM

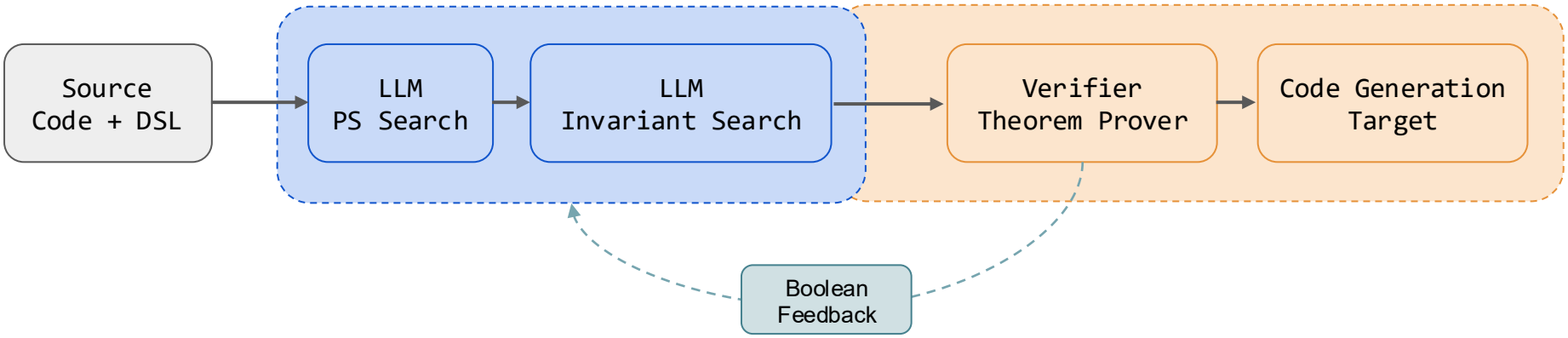
Enumerates Transpiled Code + Proof

```
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.
```

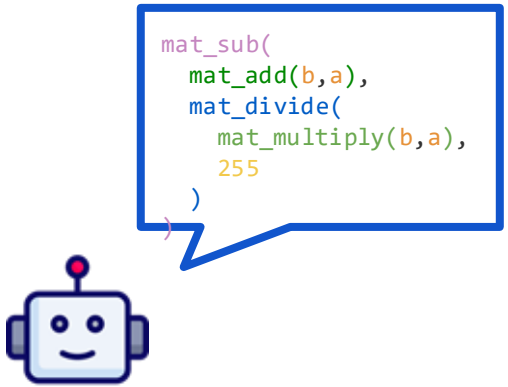
Prompt Preamble

```
#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

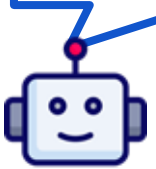
```
#defined functions
def mat_add(matrix_x: List[List[int]], matrix_y: List[List[int]]) ->
List[List[int]]:
  return ( []
    if len(matrix_x) < 1 or not len(matrix_x) == len(matrix_y)
    else [
      vec_add(matrix_x[0], matrix_y[0]),
      *mat_add(matrix_x[1:], matrix_y[1:]),
    ]
  )
```

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

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



Verification

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

Source Program

```
def proof(data: List[int], count: int, i: int) -> bool:
  return i >= 0 and i <= size(b)
    and out == mat_sub(mat_add(b[:i], a[:i]) ..
```

Generated Proofs

```
def screen_blend(b: List[List[int]], a: List[List[int]]) -
> List[List[int]]
  return mat_sub(mat_add …
```

Generated Program

Results

Source Lang	Target Lang	Symbolic Tools	LLMLift
			
			
			
		23/23	**23/23**

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

1200 LoC Heuristics

Tutorial

Setup

1. Sign-in to the aws machines
2. Clone 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

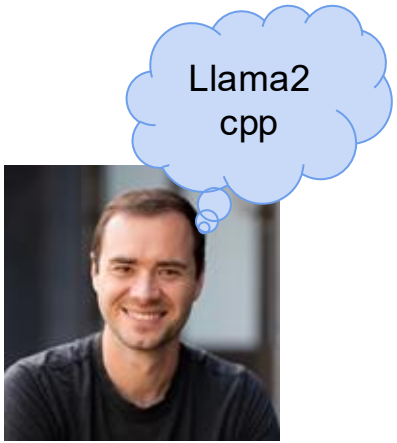
```
int test(int base, int arg1, int base2, int arg2)
{
    int a = 0;

    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

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]
```

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

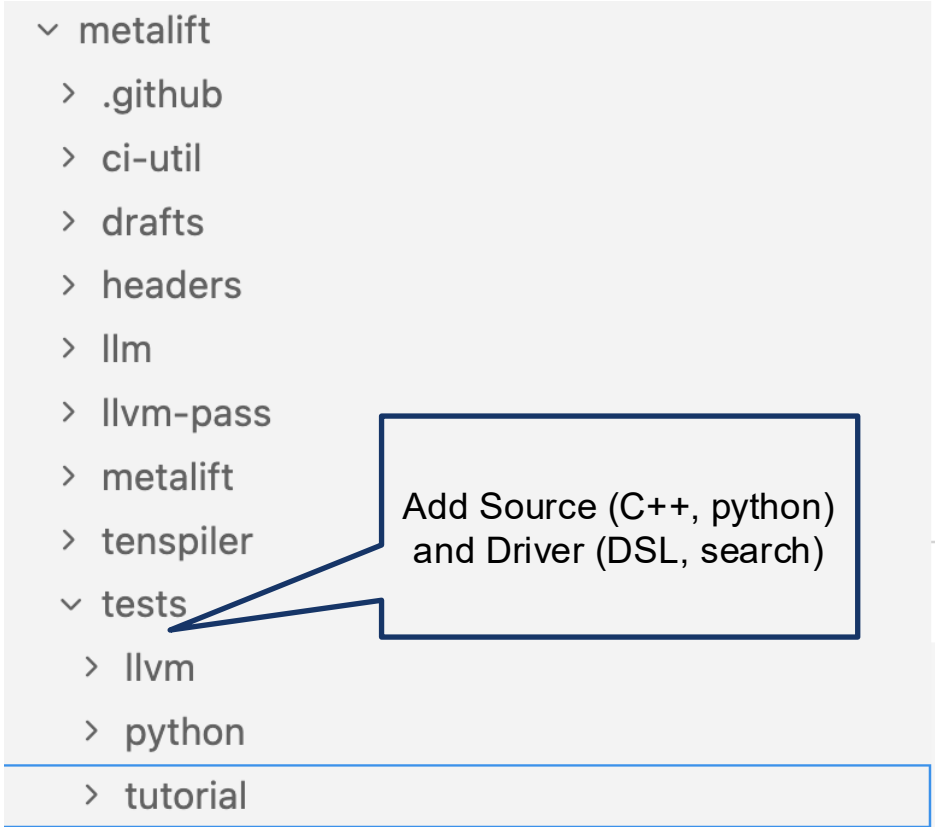
```
int conv_activation(int bias, int weight, int input)
{
    int result = bias + weight * input;
    if (result < 0) result = 0;
    return result;
}
```

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



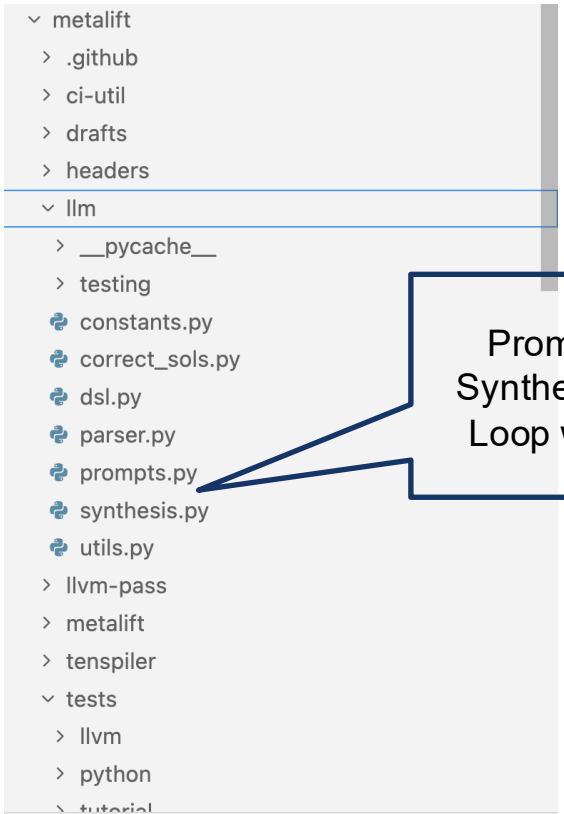
Source

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



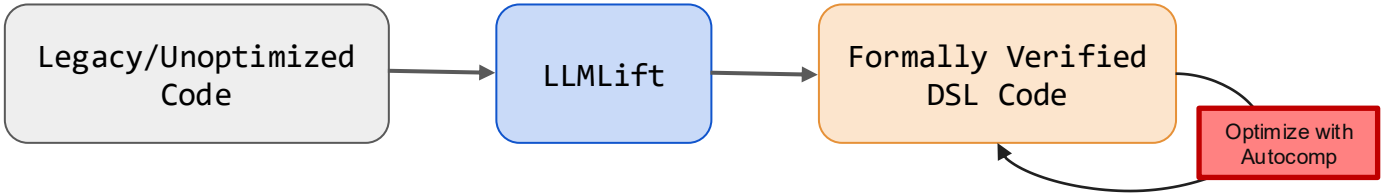
LLM + Verification

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

Acknowledgements

1. Amazing Co Vibe Coder - Jie! 
2. Amazing Berkeley Students - Huijae, Boru, Jack! 
3. aws - Jonhn, Jim! 
4. Charles, Alvin and Sophia!