Code Transpilation for Hardware Accelerators

Sahil Bhatia
Yuto Nishida, Shadaj Laddad, Hasan Genc, Sophia Shao, Alvin Cheung

Specialised Languages and Instruction Sets

Legacy/Unoptimized Code Needs Lifting

```python
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] + mat[i+1][j+1]
            )
        result.append(row)

    return result
```

Unoptimized Source Code

1 -1	2 1	3	4
5 -1	6 1	7	8
9	10	11	12

1	2	3	4
5	6	7	8
9	10	11	12

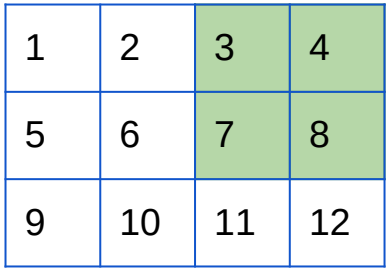
2

```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)

    return result
```

1	2	3	4
5	6	7	8
9	10	11	12

2	2

```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)

    return result
```




```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)

    return result
```

1	2	3	4
5	6	7	8
9	10	11	12

2	2	2
2		

```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)

    return result
```

1	2	3	4
5	6	7	8
9	10	11	12

2	2	2
2	2	

```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)
    return result
```

1	2	3	4
5	6	7	8
9	10	11	12

2	2	2
2	2	2

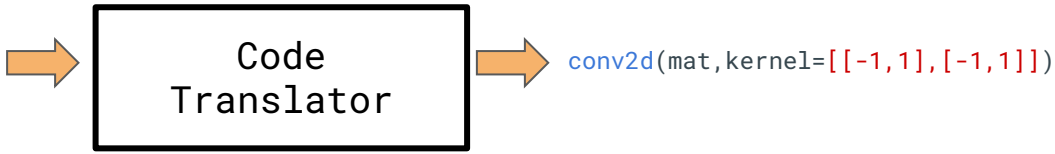
```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] +
mat[i+1][j+1]
            )
        result.append(row)
    return result
```

Convolution operation with 2x2 Filter!

Legacy/Unoptimized Code Needs Lifting

```python
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] + mat[i+1][j+1]
            )
        result.append(row)

    return result
```

Unoptimized Source Code

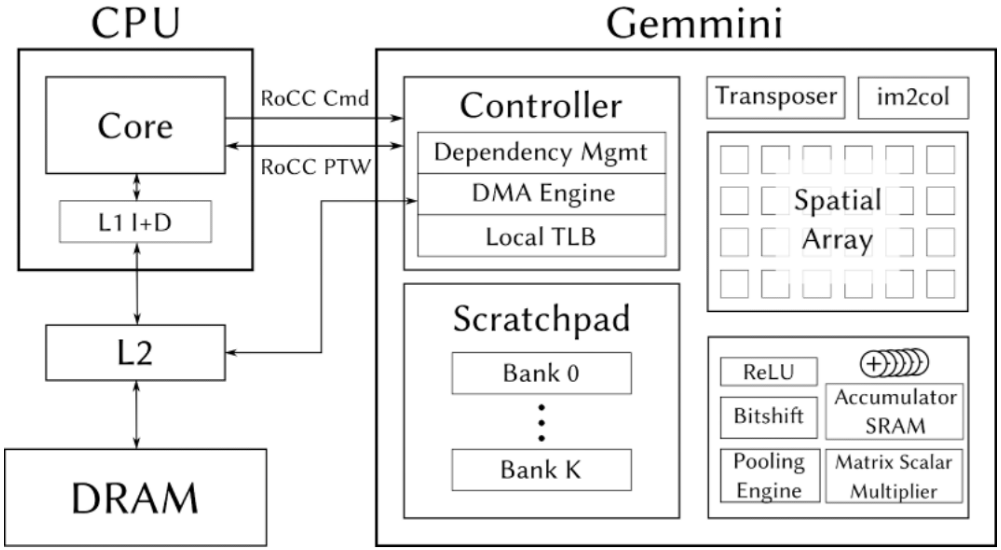


Hardware Optimized Code

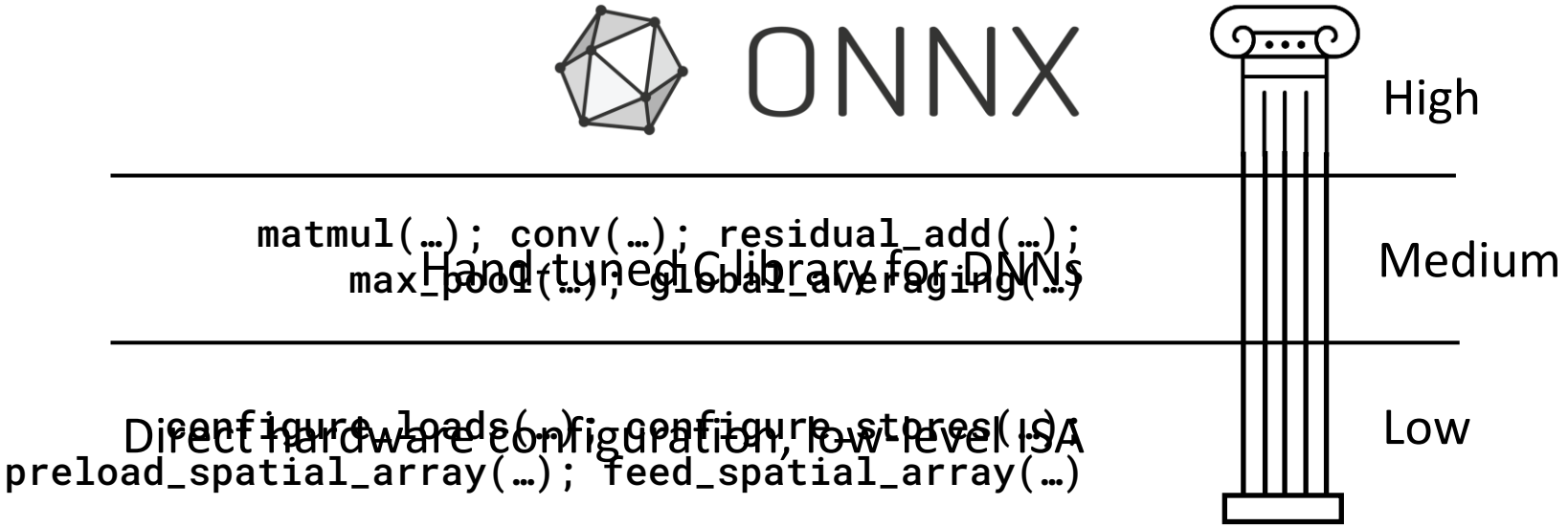
GEMMINI

GEMMINI

- DNN Accelerator Generator
- Flexible Hardware Template
- Full Stack
- Full System



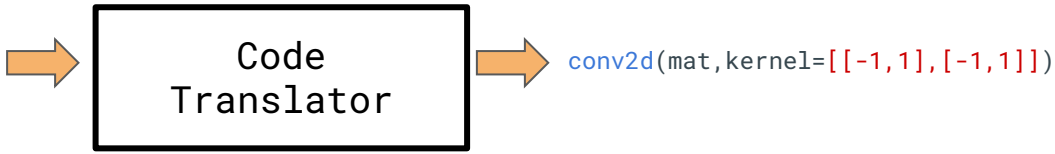
GEMMINI Programming Model



Legacy/Unoptimized Code Needs Lifting

```python
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] + mat[i+1][j+1]
            )
        result.append(row)

    return result
```

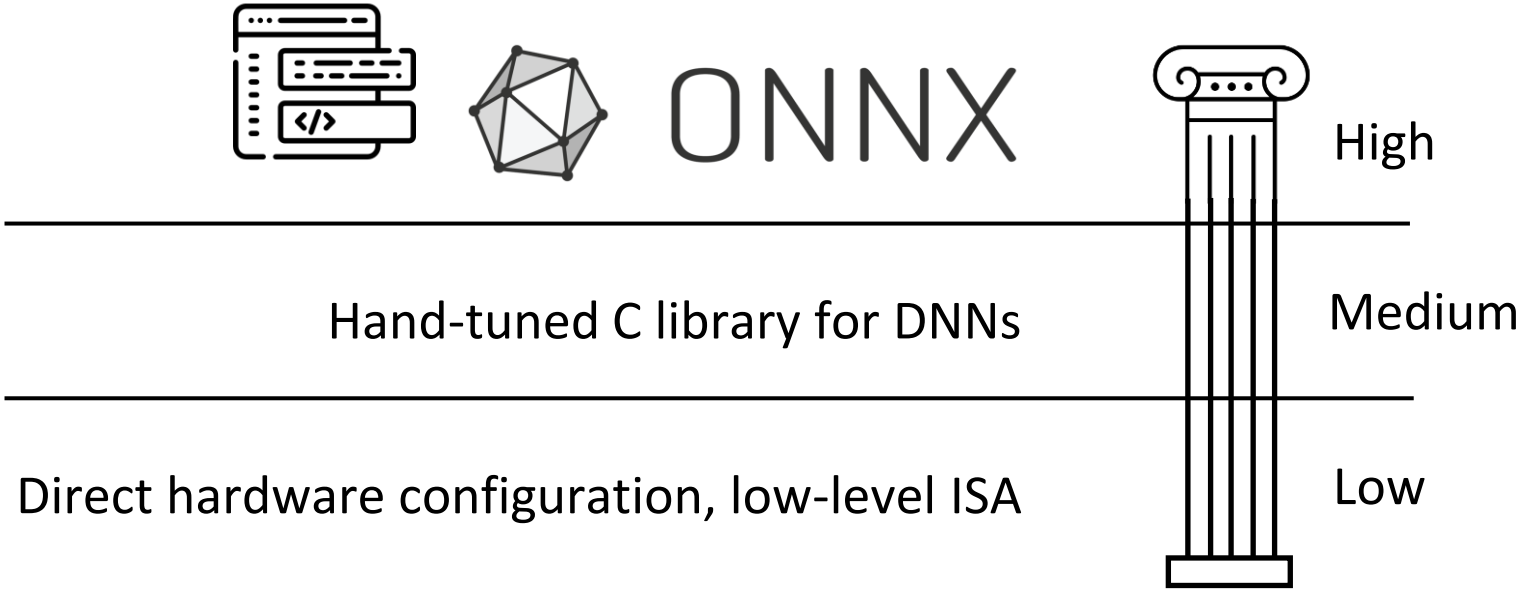
Unoptimized Source Code



Hardware Optimized Code

GEMMINI

GEMMINI Programming Model



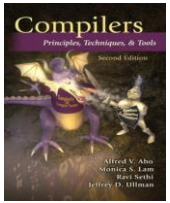
Code Translators

1. Manually Rewrite
    a. Tedious
    b. Error-Prone
    c. Infeasible for large code base

Code Translators

1. Manually Rewrite
    a. Tedious
    b. Error-Prone
    c. Infeasible for large code base

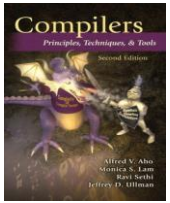
2. Pattern Matching Compiler
    a. Brittle
    b. Difficult to get right
    c. Hard to maintain



Code Translators

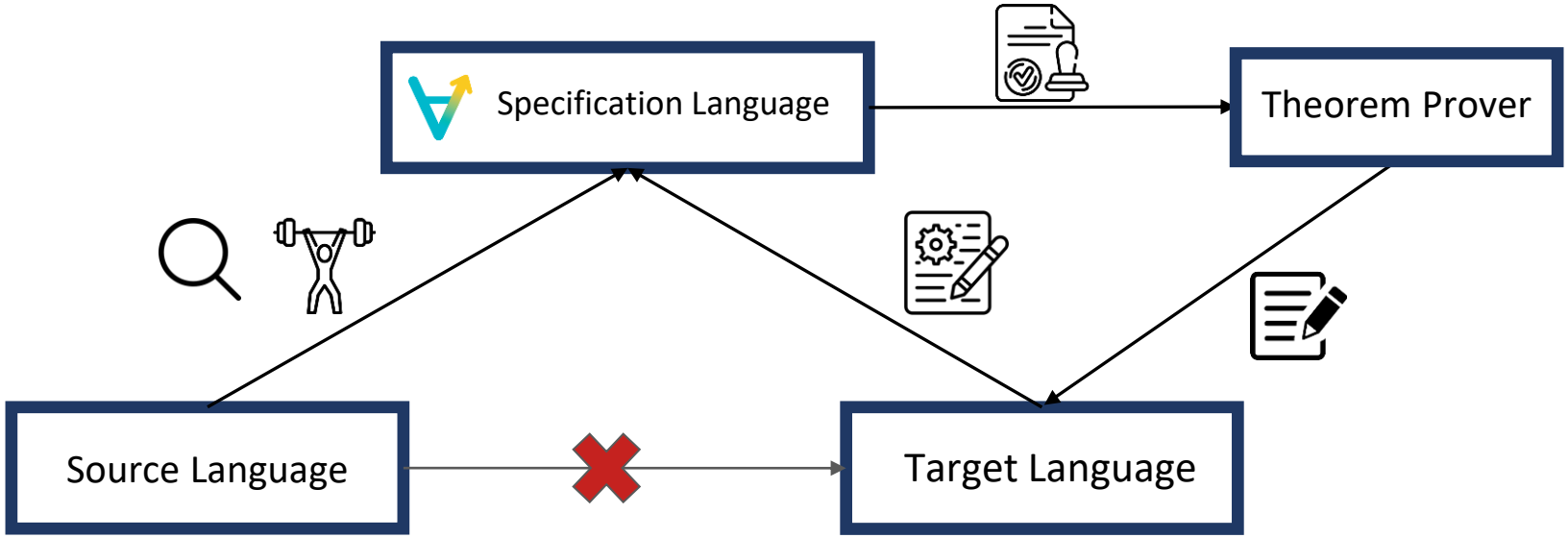
1. Manually Rewrite
    a. Tedious
    b. Error-Prone
    c. Infeasible for large code base

2. Pattern Matching Compiler
    a. Brittle
    b. Difficult to get right
    c. Hard to maintain



3. Neural Approaches
    a. No guarantees on correctness
    b. Struggles with new DSL/ISA

Metalift



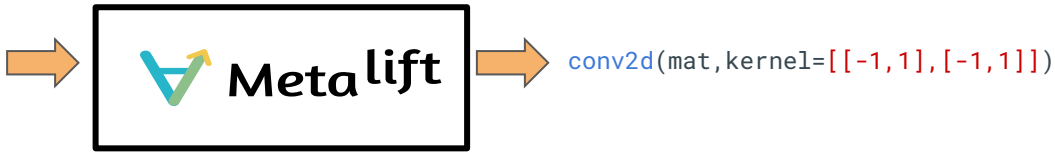
C, C++, Python

GEMMINI

Legacy/Unoptimized Code Needs Lifting

```python
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] + mat[i+1][j+1]
            )
        result.append(row)

    return result
```

Unoptimized Source Code

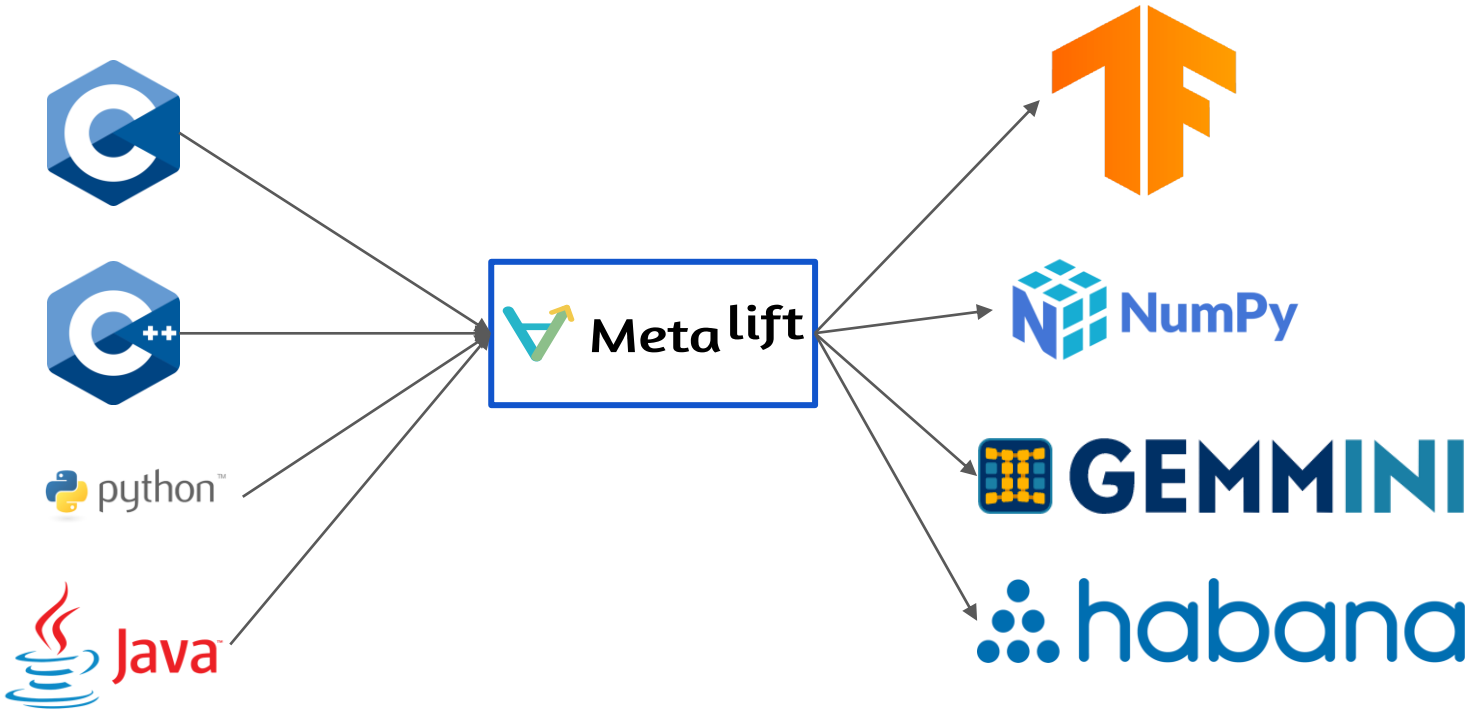


- No tedious pattern matching rules
- Code correctness guarantees
- Automated

Hardware Optimized Code

GEMMINI

Why MetaLift?



Step 1 - Defining the Semantics

```
def conv2d(inp, kernel):
    def helper(inp, kernel, i, j):
        if i >= len(inp) - 1:
            return []
        else:
            return [conv_inner(inp, kernel, i, j)] + helper(inp, kernel, i + 1, 0)
      return helper(inp, kernel, 0, 0)

def conv_inner(inp, kernel, i, j):
    if j >= len(inp[i]) - 1 :
        return []
    else:
        return
           [inp[i][j]          * kernel[0][0] +
             inp[i][j + 1]     * kernel[0][1] +
             inp[i + 1][j]     * kernel[1][0] +
             inp[i + 1][j + 1] * kernel[1][1]]
           + conv_inner(inp, kernel, i, j + 1)
```

Step 2 - Define the search space

1. Describe the output vars using the operators in the target language

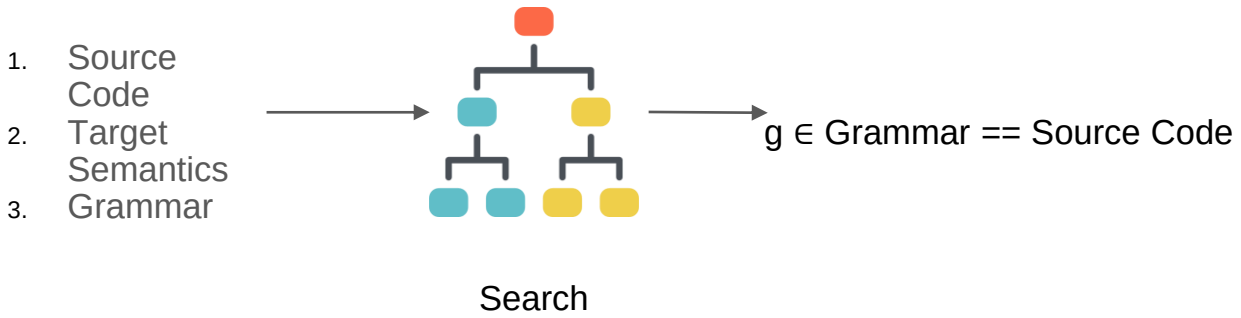
```
def grammar(mat, result):
      val    = Choose(-2,-1,0,1,2)
      result = Choose(conv2d(mat, kernel = [[val,val],[val,val]]))
```

```
result  := conv2d(mat, kernel = [[val,val],[val,val]])
val     := -2 | -1 | 0 | 1 | 2
```

Step 3 - Call our Synthesis API

1. Source Code
2. Target Semantics
3. Grammar

```
Search(grammar,target,source code)
```



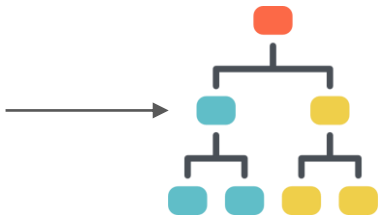
Step 3 - Call our Synthesis API

```
1. conv2d(mat, kernel = [[0,0],[0,0]])
2. conv2d(mat, kernel = [[1,0],[1,0]])
3. conv2d(mat, kernel = [[1,1],[1,1]])
4. conv2d(mat, kernel = ...
5. conv2d(mat, kernel = ...
6. conv2d(mat, kernel = ...
```

```
result := conv2d(mat, kernel = [[val,val],[val,val]])
val    := -2 | -1 | 0 | 1 | 2
```

Step 3 - Call our Synthesis API

```
def test(mat):
    rows = len(mat)
    cols = len(mat[0])
    result = []
    for i in range(rows - 1):
        row = []
        for j in range(cols - 1):
            row.append(
                - mat[i][j] + mat[i][j+1]
                - mat[i+1][j] + mat[i+1][j+1]
            )
        result.append(row)

    return result
```



Software Synthesis

```
result = conv2d(mat,kernel=[[-1,1],[-1,1]])
```

Step 4 - Write CodeGen Rules

1. Synthesized code is in high level language

```
conv2d(mat,kernel=[[-1,1],[-1,1]])
```

Step 4 - Write CodeGen Rules

1. Synthesized code is in high level language

```
conv2d(mat,kernel=[[-1,1],[-1,1]])
```

- conv2d ⇒ tiled_conv_auto (Gemmini)
- conv2d ⇒ torch.nn.conv2d (Pytorch)
- conv2d ⇒ tf.nn.conv2d (Tensorflow)

From the **same representation** we can target **three different back-ends**!

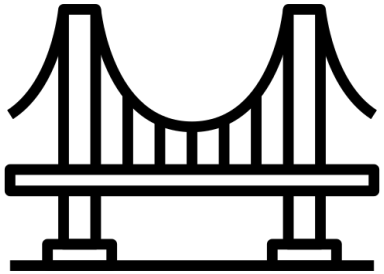
Translated Code Performance

1. Running the unoptimized code on Gemmini takes **2x** more cycles than the translated code
2. The compiler was built in < **200 LOC**

Future Work

1. Adding support for other instructions in Gemmini
2. Building the translator for other open-source accelerators
3. Optimizing the Metalift’s search to take into account the performance of the code on the accelerator

Future Work



Cost Model

- https://metalift.pages.dev/docs/tutorial/
- https://github.com/ucb-bar/gemmini

Contact: sahilbhatia@berkeley.edu