YAMML
Yet Another Matrix Manipulation Language

Bill Chen
Kent Hall
Janet Zhang
Doria Chen
James Xu

Project Manager
System Architect
Language Guru
Tester
Tester
Motivation

- As machine learning is becoming more prevalent, there is an increasing need for easier matrix-based computations.

- YAMML harnesses the familiar syntax of C++ and adds built-in support for matrix creation and common matrix operations.

- Machine learning engineers and architects can use YAMML to more efficiently and accurately perform matrix-based computations.
Compiler Architecture

source.yaml → scanner → parser → ast → semantic checking → sast → code generation → LLVM IR → notavirus.exe

External C++ library
Language Overview

Core Features
- Static scoping
- Mixed variable declarations and code
- Variable initializers (local and global)
- Explicit & implicit type casting
- Strongly & statically typed

Matrix Functions

<table>
<thead>
<tr>
<th>Access</th>
<th>M[0,0];</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splice</td>
<td>M[:,:];</td>
</tr>
<tr>
<td>Element Assignment</td>
<td>M[0,0] = 1.0;</td>
</tr>
<tr>
<td>Matrix Operations</td>
<td>M * M; M .* M; M * 2.0;</td>
</tr>
<tr>
<td>Convolution</td>
<td>trans(M);</td>
</tr>
<tr>
<td></td>
<td>filter2d(M);</td>
</tr>
</tbody>
</table>

Primitives
- int, float, str, char, bool

Matrix
- [ ]; [1.0, 2.0]; [1.0; 2.0];
- [1.0, 2.0; 3.1, 4.1];

Control Flow Keywords
- if, else, while, for, return, continue, break

Arithmetic Operators + Assignment
- -    +    =    *    /    .*    |

Logical Operators
- !    &&    ||

Conditional Operators
- <   >   ==   !=   <=   >=   !=

Comments
- /*... */   //
Language Overview: More C–based Features

Imports
#import <file.yml>

Function Declaration
int main (){  
    return 0;
}
matrix foo (matrix m){  
    return m;
}

Implicit Casting
1+1.1 //2.1

Control Flow
int i = 0;
for (i ; i < 5; i = i + 1) {  
    /* something */
}

int i = 0;
while (i < 5) { /* body */ }

int y = 5;
if (x == y) {  
    /* something */
}
else {  
    /* something */
}

Scoping
{
    int z;
    int a = 5;
    {
        int a = 7;
        a = a + 1;
        print(a); //8
    }
    a = a + 1;
    print(a); //6
}
Implementation: Matrix

```c
int main() {
    matrix M = [1,2,3];
}
```
Implementation: Standard Library and Built-ins

**Printing Functions**
- `print(int);`
- `printf(float);`
- `printb(boolean);`
- `printStr(str);`
- `printmat(matrix);`

**Matrix Functions**
- `int height(matrix m);`
- `int width(matrix m);`
- `float sum(matrix m);`
- `float mean(matrix m);`
- `matrix trans(matrix m);`
- `matrix filter2d(matrix m, matrix k);`
- `matrix empty(int r, int c);`
- `matrix imread(str filename);`
- `matrix imwrite(str filename);`
Demo: Matrix Operations

Matrix Declaration
matrix M = [1.1, 1.2, 1.3; 1.4, 1.5, 1.6; 1.7, 1.8, 1.9]; //3 row, 3 column matrix. index starts at 0
matrix N = [2.1, 2.2, 2.3; 2.4, 2.5, 2.6; 2.7, 2.8, 2.9];

Accessing Elements
M[1, 2]; //1.8

Arithmetic Operations
M*N //matrix multiplication
M.*N //element-wise multiplication
M./N //element-wise division
M * 1.1 //returns a matrix of floats

Slicing
M[0:1, 1:2]; //returns [1.2, 1.3; 1.5; 1.6]
Testing

- Run all unit tests: `./testall.sh`
- Tests:
  - Statements and expressions
  - Scope
  - Matrix operations
  - Functions
  - Standard library functions

- Run individual test:
  1. `./yammlc.sh ./tests/test-feature.yaml`
  2. `./test-feature.exe`
Future Directions

● Implement garbage collection

● Additional Matrix Operations
  ○ Colored Image Manipulation

● Additional Libraries