CSE 291D: Project Progress Report
Serverless Compiler

Prajwal Yelandur Raghu, Pratyush Karmakar, Raghasuma Aishwarya Putchakayala, Sruthi Praveen Kumar Geetha

1. Introduction

This project aims to build a web application that compiles and runs an input source program in typed Python code via serverless functions (using the serverless framework provided by AWS Lambda) and server-based functions. The compilation involves the conversion of input source code from typed Python to Web Assembly (WASM) code which is a binary executable that is later run in the web browser environment. Finally, the project also aims to compare the performance between serverless and server-based implementation by using evaluation metrics such as CPU and memory utilization and speed.

Figure 1 shows the workflow design describing how different components interact in the project. The web application will accept typed Python source code as input and display the compiled WASM code generated by both the serverless and the server-based compilers. Besides compiled WASM code, the outputs from running these WASM codes are displayed, which provide strong **proof of correctness and fidelity** of the serverless implementation of the compiler functions. An AWS Step Function is used to run a serverless workflow coordinating parser, type checker and code generator states which have been implemented as AWS Lambda functions.

2. Design

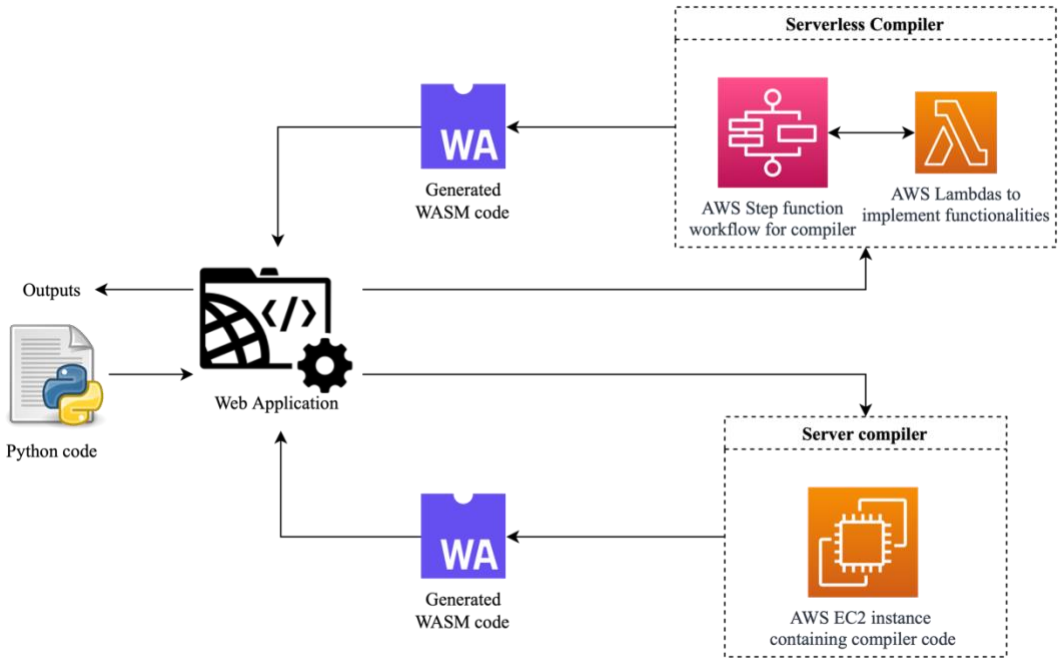

Figure 1 Design diagram

3. Implementation

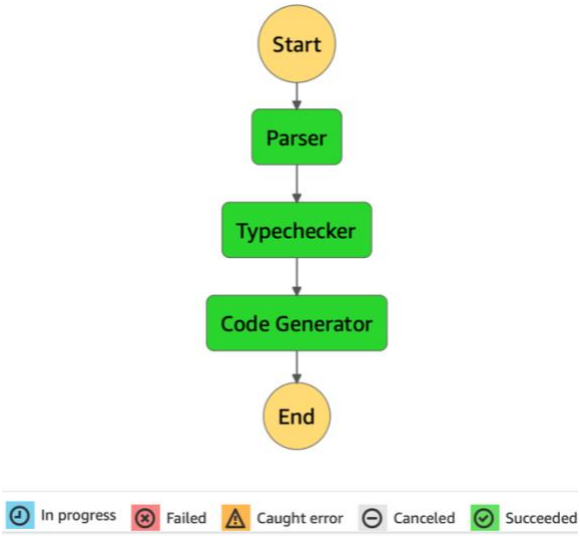

Figure 2 Graph view of AWS Step function for compiler

Figure 2 represents the successful execution of the workflow described by the AWS step function. It has three states - parser, type checker and code generator, each invoking its respective Lambda function. The **parser** takes input source program written in typed Python and translates it using Lezer-based Python grammar, into a parsed Abstract Syntax Tree (AST). This parsed AST becomes the input for the next state in the workflow - **type checker**. Type checker outputs a type checked AST which is then used by the **code generator** to produce its equivalent WASM code as output. A Lambda layer is used to package the shared dependencies which can be used with the Lambda functions of the compiler. Figure 3 shows successful compilation of the input, program – function call that returns the sum of two `int` numbers that the `add` function receives as parameters, and its generated WASM code.

Input

```
{
  "program": "def add(a:int, b:int)->int: return a+b\nadd(2,3)"
}
```

Output

```
{
  "globals": "(global $heap (mut i32) (i32.const 4))\n(func $add (param $a
i32) (param $b i32) (result i32)\n    (local $$last i32)\n    \n    (local.get
$a)\n(local.get $b)\n(i32.add)\nreturn\n    (i32.const 0))",
  "wasmSource": "(local $$last i32)\n(i32.const 2)\n(i32.const 3)\n(call
$add)\n(local.set $$last)"
}
```

Figure 3 Example input and output

4. Future Work and Timeline

Table 1 represents the project timeline. We plan to complete the following aspects of the compiler as the next steps in this project:

- Creating a web application to show the output of the execution of Python programs
- Implementing a server-based compiler
- Comparing the metrics of Serverless and Server based compilers
- Optimizing and improving the performance of the implemented serverless compiler

Table 1 Project timeline

Week	Tasks	Progress
4	Choice of Problem Statement and Project Proposal	✅
5	Exploring AWS Lambda and Step Functions	✅
6	Designing and implementing workflow using AWS Step Functions	✅
7	Progress Report	✅
8	Implementation of web application and server-based compiler	⏳
9	Comparison of server and serverless compiler and optimizations	⏳
10	Project Presentation and Report	⏳