Python Bytecode and ML Frameworks

Python Execution Flow

Python Code → Bytecode → VM Execution

ML Framework Architecture

Frontend
Python API

+

Backend
C++ / CUDA

Eager Mode

Default

- Flexible execution
- Good for debugging
- Slower performance

Graph Mode

Optimized

- Pre-compiled graph
- Faster execution
- Production ready

JIT Compilation

`torch.jit.script()` speeds up inference via Just-In-Time compilation

TorchScript

Serialize models for production (remove Python dependency)

Real Performance

CUDA kernels drive performance, not Python code

Python Overhead

Negligible for large tensor operations (batching helps)

Interactive Interpreter Execution

1 Python Source Code

```
def add(a, b):
return a + b

result = add(3, 5)
print(result)
```

2 Bytecode

3 VM Execution

Stack:

[ ]

Variables:

{ }

Output:

–

◀ Previous | Step 1 / 9 | Next ▶ | Reset

Step 1: Python Source Code

We define a simple Python function and call it. This code creates an add function that takes two numbers, then passes 3 and 5 as arguments and prints the result.