1. Workloads Selected

- Differential-equation solver (forward/backward HMM routines)
- Matrix multiplication (matrixmul.py)
- Instruction-counting script (count_instructions.py)

2. Bytecode Compilation & Disassembly

- Compiled each .py to .pyc via

```
python3 -m py_compile matrixmul.py
```

- Disassembled with the built-in 'dis' module to inspect per-instruction patterns.

3. Instruction Counting

- Used count_instructions.py to tally bytecode ops:

```
python3 count_instructions.py matrixmul.cpython-3*.pyc
```

- Matrix multiply was dominated by CALL_FUNCTION, LOAD_GLOBAL and BINARY_MULTIPLY.
- HMM routines showed heavy LOAD_FAST, BINARY_ADD and COMPARE_OP inside nested loops.

4. Profiling Execution

- Leveraged cProfile (and optionally line_profiler) in hmm_demo_profiling_progress.py to measure total and per-call times.
- Top-10 hotspots for the HMM code were:
  - forward_log – cumtime ≈ 65% of total
  - viterbi – cumtime ≈ 30% of total
  - Loop overhead, array indexing, and NumPy calls shared the remaining 5%.
- <u>Key takeaway</u>: the core of the workload is computing repeated additions, comparisons, and a max-find over state-dimension I at each time step.

5. Parallelism & Data Dependencies

- Both workloads feature tight, regular loops over states or matrix indices with minimal cross-iteration dependencies (aside from backtracking in Viterbi).
- Which suggested mapping to a systolic array of small Processing Elements (PEs): each PE computes the below in a pipelined fashion.

$$\delta_{n,j} = \max_i \{\delta_{n-1,i} + \log A_{i,j}\} + \log B_{j,o_n}$$

6. Candidate Instruction Architectures

- Matrix multiply: a fused multiply-accumulate (MAC) unit is most beneficial
- HMM forward/Viterbi: a specialized max-adder instruction (compare & add) and fast on-chip backpointer memory

7. Hardware Accelerator: Viterbi PE Chain
    - Designed a chain of PEs (systolic array) where each PE handles one hidden state and passes partial results to its neighbor.
    - Achieved fully pipelined operation: one new observation per cycle after initial latency.