Setup

Benchmark:

- Down-and-out european call option pricing
- Underlying process: GBM
  - Generated via for loop: good proxy for simulating more complex processes (Stochastic or Local Vol, SLV)
- 50k paths, 500 time steps

System Setup:

- CPU: AMD Ryzen 5 7600X
- Cores: 6-Core/12-Thread
- Memory: 32GB DDR5-4800
- Freq: Up to 5.45 GHz
- Vector Extensions: AVX512

Framework Performance Comparison

Framework	First Pass (s)	Second Pass (s)
TensorFlow	55.5302	8.2559
PyTorch	7.3873	7.2214
JAX	6.8160	6.4025
AADC	**0.1662**	**0.0995**

Table: Execution times comparison (in seconds)

First Pass: RNG + evaluate + compilation.
Second Pass: RNG + evaluate on compiled graph

Testbench code is attached.

Conclusions:

- AADC shows orders of magnitude gains in both compilation and execution.
- Why? Existing Python AAD frameworks are geared towards ML applications.
  - ML workloads:
    - Relatively few nodes (e.g. YOLO v8 network: 53 layers)
    - Each node is big (parameter matrices).
  - Quant finance workloads:
    - Many nodes (e.g. typical HW1F SDF + short rate simulation: > 1000 nodes)
    - Each node is small (time steps in a process simulation loop).

- AADC is specifically designed for quant finance workloads.
- Framework can fully exploit AVX512 hardware capabilities.
- It comes with support of well-known and loved NumPy ufuncs and functions.
- If needed we can record through a mixture of pure Python and Python bindings for existing C++ libraries (proprietary or OSS, e.g. QuantLib).