A. On-Device Profiling: Reproducibility Guide

All profiling scripts target Intel Lunar Lake via OpenVINO. The pipeline runs in three stages.

Step 1 | Convert and Quantize

```
python convert.py              # base models
python quantize_uniform.py     # INT8 / INT4
python quantize_mixed.py       # KL M-P (CPU)
python quantize_mixed_gpu.py   # KL M-P (GPU)
```

`convert.py` exports each Hugging Face Mamba and Mamba-2 checkpoint to an OpenVINO FP16 IR (`.xml`/`.bin`) using the OpenVINO Model Conversion API. The resulting FP16 baselines are written to `ov_models/` and serve as the starting point for all quantization scripts.

`quantize_uniform.py` produces two fixed-precision endpoints for each model: a fully INT8-symmetric model and a fully INT4-symmetric model, both quantized per-channel with no sensitivity guidance. These bracket the Pareto curve — uniform INT8 gives the highest quality at larger model size, and uniform INT4 gives the smallest size at lowest quality — and serve as baselines for comparing the mixed-precision configurations in between. SSM `conv1d` layers (OpenVINO type `Convolution`) are excluded from quantization in all scripts; for Mamba-2 the XAMBA CumBA MatMul node is additionally excluded because it has no INT8/INT4 GPU kernel on Intel Lunar Lake iGPU.

`quantize_mixed.py` implements our KL-guided mixed-precision method for the CPU pipeline, targeting Mamba-130M and Mamba-1.4B. It merges the per-layer 4-bit and 8-bit KL sensitivity data into a single list, sorting entries from least to most sensitive. Each layer therefore appears twice — once for its 4-bit cost and once for its 8-bit cost — with last-wins semantics resolving ties. Ten evenly-spaced cutoff points are drawn through this merged list. At each cutoff, a two-pass NNCF compression is applied to the FP16 baseline: Pass 1 compresses the designated layers to INT4_SYM; Pass 2 compresses a separate set to INT8_SYM; remaining layers stay FP16.

`quantize_mixed_gpu.py` implements the GPU pipeline, which is restricted to INT8/FP16 mixed precision because Intel Lunar Lake iGPU does not expose INT4 weight kernels. It loads only the 8-bit sensitivity data, sorts layers from least to most sensitive, and applies a single-pass INT8_SYM compression that ignores the sensitive tail. Mamba-2 is capped at eight points (`point01`–`point08`) because the surrounding INT8 context at higher compression levels causes oneDNN to fail its primitive descriptor for the XAMBA CumBA MatMul node, even when that node is explicitly excluded from quantization.

The sensitivity metric is controlled by a constant inside each script. All output filenames automatically inherit the selected tag (e.g., `mamba-130m-hf_kl_point05.xml`):

```
SENSITIVITY_METRIC = "kl_student_to_teacher"
METRIC_TAG = "kl"
```

The ten configurations **p01–p10** are produced by splitting the sensitivity-ranked layer list into equal segments. Configuration **p01** quantizes only the least sensitive layers, while **p10** approaches near-uniform quantization.

Step 2 | Benchmark

```
python benchmark.py      # CPU latency +
    throughput
python benchmark_gpu.py  # GPU latency +
    throughput
```

Each model is evaluated using the OpenVINO `benchmark_app` tool:

```
benchmark_app -m <model>.xml \
  -d CPU -hint latency \
  -t 60 -niter 50 \
  --inference_only TRUE
```

The flags `-t 60` and `-niter 50` ensure that each run executes for at least 60 seconds and at least 50 iterations, whichever takes longer.

Results are written to `log/benchmark_log/` as per-model text logs and a summary CSV file: `latency_throughput_{device}_{tag}_report.csv`.

```
python eval_perplexity.py      # CPU, WikiText-2
    PPL
python eval_perplexity_gpu.py  # GPU, WikiText-2
    PPL
```

Perplexity is computed on the WikiText-2 test set. Fake quantization is applied to replicate each configuration's weight precision during evaluation.

```
ds = load_dataset(
    "wikitext", "wikitext-2-raw-v1",
    split="test")
text = "\n\n".join(
    t for t in ds["text"] if t.strip())
```

Results are stored in `perplexity_results_{tag}.json` with the structure

```
{model: {point: perplexity}}
```

This format allows direct comparison across sensitivity metrics and model scales.