BENCHMARK REPORT

# MiniLM-L6-v2 on the JVM: *How far can you push CPU inference?*

Model: all-MiniLM-L6-v2 Runtime: ONNX Runtime 1.20.0 / Java 21 Hardware: 12-core CPU, Windows 11, High Performance Date: April 14, 2026

## Why this benchmark exists

StreamKernel is a JVM-native real-time AI enrichment engine — a single process that ingests streaming records, runs ONNX inference in-process, and writes enriched output to a sink. No Python. No sidecar. No network hop to a model server.

That architectural bet — **inference lives inside the JVM, not beside it** — is the core value proposition for regulated and air-gapped environments. But it raises an obvious question: how fast is *fast enough* on CPU, and which configuration choices actually move the needle?

To answer that precisely, we built `djl-onnx-bench`: a standalone Java benchmark harness that calls ORT directly, with controlled warmup, cooldown between runs, and a sweep runner that randomizes execution order to prevent thermal accumulation bias. This post reports the results of an 8-configuration sweep.

> *"The goal was not to find the fastest number we could publish. It was to understand the shape of the performance surface — where INT8 wins, where thread count matters, and where the tail latency tells a different story than throughput."*
>
> STREAMKERNEL ENGINEERING

## What we tested

Two model variants — **fp32** (standard ONNX export) and **INT8** (dynamically quantized) — across two sequence lengths ( `s16` , `s32` ) and two intra-op thread counts ( `3` , `4` ), all at batch size 32. Eight configurations total, randomized execution order, 10 warmup iterations discarded, 40 measured iterations per run, 10-second cooldown between configurations.

**METHODOLOGY NOTE**

Sequence lengths of 16 and 32 tokens are shorter than StreamKernel's production `tokenizer.max.length=128` . These values were chosen to map the performance surface efficiently. Inference scales roughly O(n²) with sequence length in transformer attention, so production numbers at seq=128 will be proportionally slower — but the *relative* rankings between configurations hold.

All runs on High Performance Windows power plan, plugged in, CPU core parking disabled, PCI-E ASPM off.

| 347 | 8 | 2.9× |
|---|---|---|
| PEAK EPS | CONFIGURATIONS | INT8 / FP32 RANGE |

RESULTS MATRIX

## Sorted by EPS — what actually won

INT8 quantized  FP32 standard

| | MODEL | SEQ | THREADS | AVG MS | P50 MS | P95 MS | P99 MS | EPS | P95/P50 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | int8 | 16 | 3 | 91.97 | 82.05 | 133.51 | 192.31 | 347.82 | 1.63× |
| 2 | int8 | 16 | 4 | 97.15 | 87.23 | 147.15 | 196.72 | 329.28 | 1.69× |
| 3 | fp32 | 16 | 4 | 100.05 | 94.57 | 137.51 | 158.31 | 319.76 | 1.45× |
| 4 | fp32 | 16 | 3 | 120.31 | 111.82 | 172.38 | 188.28 | 265.92 | 1.54× |
| 5 | int8 | 32 | 4 | 149.64 | 142.90 | 238.77 | 290.61 | 213.81 | 1.67× |
| 6 | int8 | 32 | 3 | 187.55 | 164.83 | 258.13 | 306.04 | 170.60 | 1.57× |
| 7 | fp32 | 32 | 4 | 245.14 | 234.65 | 329.53 | 426.50 | 130.52 | 1.40× |
| 8 | fp32 | 32 | 3 | 265.84 | 253.88 | 428.02 | 582.37 | 120.36 | 2.29× |

## What the data says

Finding 01

### INT8 wins on throughput — but not on tail consistency

The top EPS belongs to `int8 s16 intra3` at 347.82. But `fp32 s16 intra4` has the tightest tail in the entire sweep: P95/P50 of 1.45×. For streaming pipelines where tail latency causes queue buildup, that

Finding 02

### INT8's advantage grows with sequence length

At s16, int8 intra4 is only 3% faster than fp32 intra4 (97ms vs 100ms). At s32, that gap widens to 39% (150ms vs 245ms). INT8 quantization reduces memory bandwidth pressure — and longer

distribution shape may matter more than raw throughput.

sequences have larger attention matrices where that pressure compounds.

Finding 03

### Thread count has a crossover point

`intra3` beats `intra4` for INT8 at short sequences (347 vs 329 EPS), but loses at longer ones (171 vs 214 EPS at s32). At s16 the work is small enough that 3 threads saturate it without coordination overhead. At s32, the fourth thread earns its keep.

Finding 04

### Run 6 of 8 shows thermal accumulation

The `fp32 s32 intra3` run — sixth of eight, after four consecutive CPU-intensive benchmarks — has a P99/P50 ratio of 2.29×. The 10-second cooldown between runs was insufficient. This is why randomized execution order and per-run cooldowns are non-negotiable in CPU benchmarking.

## The CPU budget constraint

These results were collected on a 12-logical-core machine. The winning configurations aren't random — they respect a hard constraint: **total ONNX threads ≤ physical cores**. Violating this boundary, as we confirmed in separate StreamKernel pipeline runs, produces severe contention: a `pool.size=3, intra=6` configuration (18 threads on 12 cores) regressed from 410ms to 1,460ms average batch time.

The sweet spot for this hardware is **2–3 predictors × 4–6 intra-op threads = 8–12 total threads**. Beyond that, you're not buying parallelism — you're buying scheduling contention.

FOR STREAMKERNEL USERS

The INT8 model is a drop-in replacement. Set `ai.embedding.model.uri` to your `model.int8.onnx` path. No other changes required. Based on this sweep, expect 35–50% lower batch latency at production sequence lengths compared to fp32, with proportionally higher end-to-end pipeline throughput.

## Benchmark methodology

| | |
|---|---|
| HARNESS<br>djl-onnx-bench (Java 21) | RUNTIME<br>ORT 1.20.0 / DJL 0.32.0 |
| MODEL<br>all-MiniLM-L6-v2 (fp32 + INT8) | BATCH SIZE<br>32 (fixed) |
| WARMUP<br>10 iterations (discarded) | MEASURED<br>40 iterations per config |
| EXECUTION ORDER<br>Randomized | COOLDOWN<br>10s between configs |
| POWER PLAN<br>High Performance (AC) | INPUT TYPE<br>Synthetic pre-tokenized tensors |

Input preparation averaged 0.001–0.002ms per batch — effectively zero. All reported latency is pure ORT inference time. Tokenization cost, DJL pooling, and sink write latency are measured separately in full StreamKernel pipeline benchmarks.

---

## What's next

The immediate next step is running the INT8 model through the full StreamKernel pipeline at `seq=128` — the production tokenizer length — to measure end-to-end throughput uplift. Based on the scaling observed between s16 and s32 in this sweep, INT8 at seq=128 should cut batch latency by 35–50% relative to the fp32 baseline, pushing pipeline throughput well above the current 18.8 records/sec ceiling established in Run 3.

Longer term, the same benchmark harness will be used to evaluate alternative embedding models and quantization strategies as StreamKernel's model zoo expands. The sweep infrastructure supports arbitrary config matrices — add a row to `sweep.csv` and the runner handles the rest.

---