From Compute Load to Operation Count: A Trace-Based View of Agentic Workflows on Heterogeneous SoCs

Siavash Alamouti, Fay Arjomandi, Michel Burger, Jeremy Hsu
mimik Technology Inc.

Abstract

Studies of agentic AI on heterogeneous SoCs usually measure compute demand: how much work each workload gives the CPU, the GPU, and the NPU. Demand tells you how much load each resource carries. It does not count operations, so it cannot tell you how many separate operations of each kind a real workflow runs. We traced a real multi-agent video summarization workflow on the Agentix Operating Engine (mimOE), mimik's hybrid distributed platform for multi-agent systems, running across one coordinating node and two worker nodes. From one run we report two numbers. Counted by operations, 82 percent are CPU operational work and 18 percent are GPU inference. Counted by time, the numbers reverse: 11 percent CPU and 89 percent GPU. Both numbers are correct, and they measure different things. We use the trace to show how a real agentic workflow uses the CPU, the GPU, and memory of a heterogeneous SoC, and why its speed depends on how well these run together rather than on the peak speed of any one of them.

Executive summary

A real agentic workflow is not one large inference job with a small amount of supporting code. It is a long series of short operations that coordinate, move, and prepare data, with a small number of long inference calls in between. We traced one such workflow, a multi-agent video summarization run, and measured it two ways from the same run. Counted by the number of operations, most of the work is on the CPU. Measured by time, most of the work is on the GPU. The GPU handled under a fifth of the operations but used close to nine tenths of the compute time. Both views are correct, and they describe the same run.

The consequence for the choice of an SoC follows from this. End-to-end speed does not depend on the peak speed of the GPU on its own. It depends on how well the CPU and the GPU work at the same time, with the CPU preparing the next batch of data while the GPU runs the current one, and on how quickly data moves between them. An SoC with a fast GPU can still run slowly if the CPU cannot keep it busy, or if moving data between the two is slow. What decides this is how well the CPU, the GPU, and memory work together on one SoC, not the peak speed of any single one of them.

For anyone choosing or building silicon for agentic AI, this is the practical point: a real workflow rewards SoCs whose CPU, GPU, and memory are well matched and run together, and the way to judge that is to measure a real workflow rather than to compare peak GPU figures.

1 The gap between load and count

Our companion study [1] models compute demand on a heterogeneous SoC. For each workload it computes how much work the CPU, the GPU, and the NPU each carry, and from that it derives

utilization, latency, and memory use. That work answers a clear question: how much load does each resource carry under a given workload.

It does not answer a second question. A demand model treats one inference call as some quantity of CPU work plus some GPU work plus some NPU work. It never counts the call as one operation, and it never counts the many smaller operations that surround the call. So it cannot tell you how many separate operations of each kind a real workflow runs, or where those operations occur.

These are two different views of the same workload. The first is compute load. The second is operation count. The first is best produced by a model. The second is best produced by a trace, because only a trace records each operation as it happens. This paper supplies the second view, using a trace of a real workflow, and then connects the two views.

2 The workflow and the trace

2.1 The workflow

The workflow is split video summarization, running on mimOE as a set of cooperating agents. mimOE is mimik's Agentix Operating Engine[1], a hybrid distributed platform: it runs the agents and lets them coordinate across several nodes, where a node is one device taking part in the run. One coordinating node holds the source video and runs a small planner model. It divides the video into time ranges and assigns each range to a worker node. Each worker node pulls its assigned segment, decodes it, selects informative frames, packs those frames for the vision-language model (VLM), and runs the model on the GPU. Each worker returns a short structured description to the coordinating node, which merges the descriptions and produces the final summary.

The run we traced used one coordinating node and two worker nodes. Each worker node holds both the CPU work, that is decode, frame selection, and packing, and the GPU work, that is the vision-language model. Frames are produced and consumed on the same node, so the frame data does not cross the network. Only the video segment and the short descriptions cross the network.

2.2 The trace

The run produced a distributed trace [3, 4]. A distributed trace is a record of the operations in a run, and each operation appears as one span, which notes the service that ran it, its start time, and its duration. This trace has 50 spans, and the whole run took 37.2 seconds of wall-clock time.

We classify each span by the service that produced it. The vision-language model runs as a single service, and that service runs on the GPU, so every span from it is a GPU operation. Every other span, that is coordination, data transfer, and preprocessing, is a CPU operation. This rule does not assign operations to the CPU or the GPU by assumption. It reads the assignment from the service that actually ran.

A distributed trace records both sides of a remote call, the client side and the server side, as separate spans with the same operation name. Counting both would count one operation twice. We therefore report the server-side spans only, which removes the duplicates. The server-side set has 28 operations. For reference, the full set of 50 spans gives 77 percent CPU by count and 10 percent CPU by time, close to the server-side figures below; we report the server-side set because it counts each operation once.

[1]The "x" in Agentix denotes a cross-platform design: the same microservice agents run across CPU-, GPU-, and NPU-centric devices, from compute-capable end devices such as smart cameras and robots up to servers.

3 Results

3.1 The two counts

Table 1 gives the two views from the same run. By operation count, the CPU runs 23 of 28 operations, which is 82 percent. By compute time, the GPU uses 43.1 of 48.6 seconds of measured CPU and GPU time, which is 89 percent. The two views point in opposite directions, and both are correct. One inference call is a single operation that runs for a long time. One coordination or data operation is short, but there are many of them.

Table 1: Operation count and compute time from the same run (server-side spans).

	Operations	Share	Time (s)	Share
CPU	23	82%	5.5	11%
GPU	5	18%	43.1	89%
Total	28	100%	48.6	100%

The ratio is 4.6 CPU operations for every GPU inference. The GPU does the heavy compute. The CPU does most of the operations.

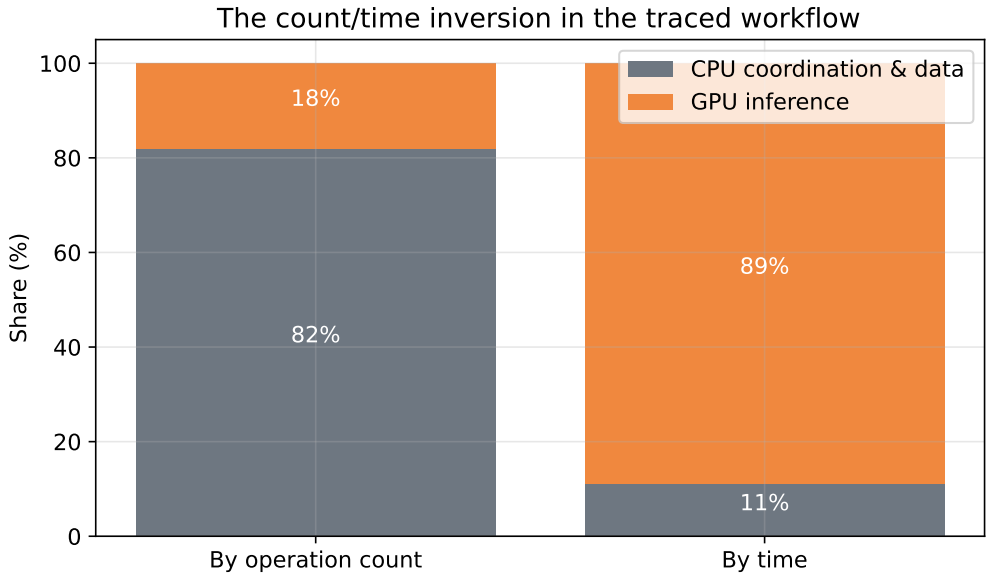

Figure 1: The same run seen two ways. By operation count the CPU is 82 percent; by compute time the GPU is 89 percent. The two views reverse.

3.2 What the CPU operations are

Table 2 breaks the CPU operations into groups. They fall into three kinds of work: coordination, which sets up and directs the run; data movement, which brings the video and the model to the worker nodes; and preprocessing, which turns video into frames the model can read. A small tracing operation is also present. Each group repeats once per worker node, so the count of these operations grows as a workflow adds nodes and agents, while the kind of work stays the same.

Table 2: CPU operations by group, and the GPU total (server-side spans).

Operation group	Resource	Operations	Time (ms)
Coordination (node discovery)	CPU	3	161.1
Coordination (deployment)	CPU	3	1967.3
Coordination (load balancing)	CPU	3	242.6
Data transfer (video chunk)	CPU	4	975.6
Data transfer (model image pull)	CPU	3	1459.9
Preprocessing (frame extraction)	CPU	6	550.7
Tracing	CPU	1	93.1
CPU total	CPU	23	5450.4
VLM inference	GPU	5	43114.3

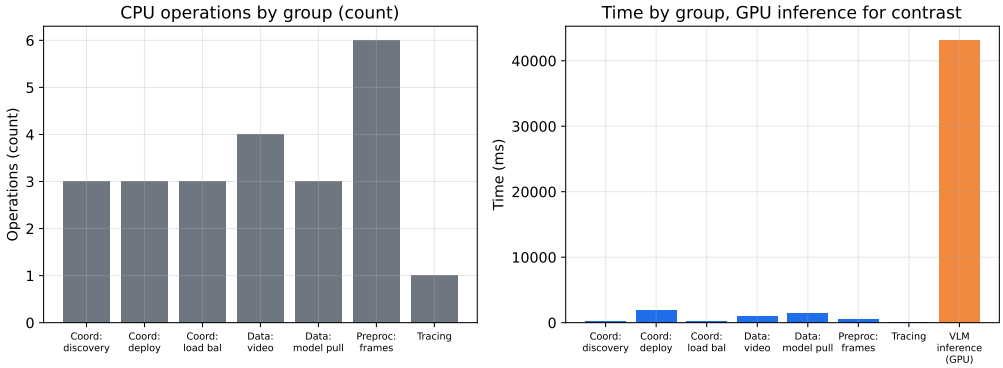

Figure 2: The CPU operations by group, with the GPU inference for contrast. The CPU work is coordination, data movement, and preprocessing, and each group repeats once per worker node.

3.3 Overlap

The GPU used 43.1 seconds of compute time, but the whole run finished in 37.2 seconds. The GPU time is larger than the wall-clock time because the two worker nodes ran their inference at the same time. This is the behavior the workflow is built to produce: while the GPU on a node runs the model on one batch of frames, the CPU on the same node prepares the next batch. When this overlap holds, the CPU work is hidden behind the GPU work and adds little to the total time. When it does not hold, the CPU work and the GPU work run one after the other, and the total time grows.

4 How a real agentic workflow uses a heterogeneous SoC

The trace gives a clear picture of how this workload uses an SoC with a CPU, a GPU, and shared memory.

The CPU is busy throughout the run, in many short operations. It is not idle, and it is not the slowest resource by time, but it is on the path of almost every step. Coordination, data movement, and preprocessing all run on it, and each step must finish before the next inference can begin.

The GPU sets the total time. It runs few operations, but each one is long, and together they are 89 percent of the compute time. An SoC with a weak GPU would be slow on this workload,

because the model is the longest single piece of work.

The handoff between the two is what decides the total time. On every batch, the CPU prepares frames and the GPU reads them. If this handoff is fast, meaning the GPU can read the frames the CPU prepared without the data being copied from one memory to another, and the two resources overlap, the run is fast. If the handoff requires copying data between separate memories, or if the CPU cannot keep up, the GPU waits and the run slows. An SoC that shares memory between the CPU and the GPU lets the GPU read the prepared frames in place, with no separate copy.

The NPU is not used in this workflow, because the model runs on the GPU. This is the GPU-heavy case. In other agentic workflows, such as continuous sensing with small models, the same operation-count view would put many short operations on the NPU instead. This trace is one specific case, but the pattern holds across agentic workflows: the workload runs a large number of short operations that prepare and direct work, and a small number of long operations that do the heavy compute, and the two must overlap.

5 Relation to the companion study

Our companion study [1], *Architectural Fit for Production-Scale Agentic AI on Heterogeneous SoCs*, models compute demand across the modeled scenarios for two architecture classes: a CPU-centric class with a strong CPU, a capable GPU, an NPU, and shared memory; and a GPU-centric class with a dominant GPU and a moderate CPU. It reports that the CPU-centric class holds the most CPU headroom in 100 percent of scenarios, and the most total compute headroom in 74 percent under static single-engine placement, rising to 82 percent once mimOE balances each workload across the available engines, ceding only the large-model reasoning-heavy workloads to the GPU-centric class; energy is comparable between the two classes. In its multi-tenant healthcare use case, once mimOE removes memory as the constraint the workload is bound by operational work on the CPU, and the CPU-centric class sustains more than twice as many patients per device as the GPU-centric class, on the order of 234 against 104. A companion deployment study [2] reports that a microservice deployment on mimOE uses far less memory than a monolithic one, roughly 10 times less for a 15-patient multi-tenant case, enough that the monolithic footprint exceeds a 128 GB device while the microservice footprint fits, and that removing the per-device memory cap together with pooling work across devices multiplies effective cluster capacity.

This paper explains the mechanism behind those results. The demand model shows that the CPU-centric class keeps utilization low and supports more work. The trace shows why. A real agentic workflow runs many short CPU operations that must overlap with a few long GPU operations. A CPU-centric SoC with a strong CPU and shared memory runs the short operations and the handoffs without slowing the GPU, so utilization stays low and more work fits on the SoC. An SoC that is strong in only one resource struggles with this workload, because it cannot run the many short CPU operations and keep the GPU busy at the same time.

The two views are complementary, and neither sizes an SoC on its own. Sizing an SoC by compute share alone would treat the workload as 89 percent GPU and would under-build the CPU and the data path, which carry 82 percent of the operations. Sizing an SoC by operation count alone would treat the workload as 82 percent CPU and would under-build the GPU, which carries 89 percent of the time. An SoC must serve both: the few long operations and the many short ones, with overlap between them.

6 Scope and limits

This is one workflow and one trace. It is a strong workflow for the question, because it is a real multi-agent run with a clear split between coordination, data, preprocessing, and inference, but it is one case.

The run is distributed across three nodes, not a single SoC. It measures the structure of an agentic workflow, that is the count, the time, and the overlap of operations of each kind. It does not measure a single SoC. The structure it shows is what informs the choice of an SoC, and it is the structure that the companion demand model uses, but the trace itself is a measurement of the workflow, not of the silicon.

The other workflow profiles in this series are built by enumerating the operations of each workflow and assigning each to a resource, then calibrating to the operation mix this trace shows. Those profiles are models, and we label them as models. This trace is the one measured anchor.

7 Conclusion

A demand model and an operation-count trace measure two real and different properties of an agentic workload. The trace of a real video summarization workflow shows that the GPU carries most of the compute time but a small share of the operations, that the CPU carries most of the operations in a small share of the time, and that the total time depends on how well the two overlap over shared memory. This explains, from the workflow side, why a CPU-centric SoC with a strong CPU, a capable GPU, and shared memory carries an agentic workload with low utilization and high capacity. The compute-load model and the operation-count trace point to the same factor: how well the many short CPU operations overlap with the few long GPU operations, and how quickly data moves between them.

References

[1] S. Alamouti et al., *Architectural Fit for Production-Scale Agentic AI on Heterogeneous SoCs*, companion study.

[2] S. Alamouti et al., *Why the Agentix Operating Engine Matters: A Monolithic vs Microservice Comparison of Agentic AI on Heterogeneous SoCs*, companion study.

[3] OpenTelemetry, distributed tracing specification and tooling. https://opentelemetry.io

[4] Jaeger, distributed tracing platform. https://www.jaegertracing.io

A Classification rules

Each span is assigned to a resource by the service that produced it.

- A span from the vision-language model service is a GPU operation.
- A span from any coordination, data transfer, or preprocessing service is a CPU operation.
- Root and container spans, which hold child operations rather than doing work themselves, are excluded from the counts.
- Where a remote call appears as both a client span and a server span with the same operation name, only the server span is counted, so each operation is counted once.

B Reproduction

The counts and times in this paper are produced from the trace file by the analysis script, which reads each span, applies the classification rules above, removes duplicate client and server spans, and prints the operation counts, the time totals, and the breakdown by group. The trace file and the script are provided with this paper.