Scaling Intelligence
Agentic Workflows with vLLM

TechXchange – 28 April 2026

Jun Kang Chow
Senior Principal AI Engineer,
Embedded LLM

embedded

VISION

vLLM everywhere: the fastest engine on any hardware, powering an interoperable, federated AI utility.

MISSION

A federated AI utility powered by vLLM: fastest on any hardware, interoperable by design, and aligned with the ecosystem so contributors and partners all benefit.

AGENDA

- Why run LLMs yourself?
- What is vLLM?
- What are agents and tools?
- Agentic application examples

Why run LLMs yourself?

Security

Your data never leaves your VPC. Sensitive prompts stay inside your trust boundary.

Privacy

No vendor telemetry on your prompts. No mystery retention policy. No black box.

Control

Your model. Your routing. Your SLAs. Your own pace on upgrades and policy.

The real question

It is no longer whether to self-host. It is what engine you choose once the workload gets serious.

Loading a model is easy. Serving it is hard.

1 user, 1 GPU

works

Any runtime can do this. A single request on a dedicated GPU is not the hard part.

1000 users, 1 GPU

- **KV cache explodes**
- memory fragments
- latency spikes
- GPU sits idle

This is where local-LLM tooling falls over. Serving means scheduling, packing memory, and keeping the accelerator busy under constant churn.

What is vLLM?

Hosted under PyTorch Foundation and THE LINUX FOUNDATION

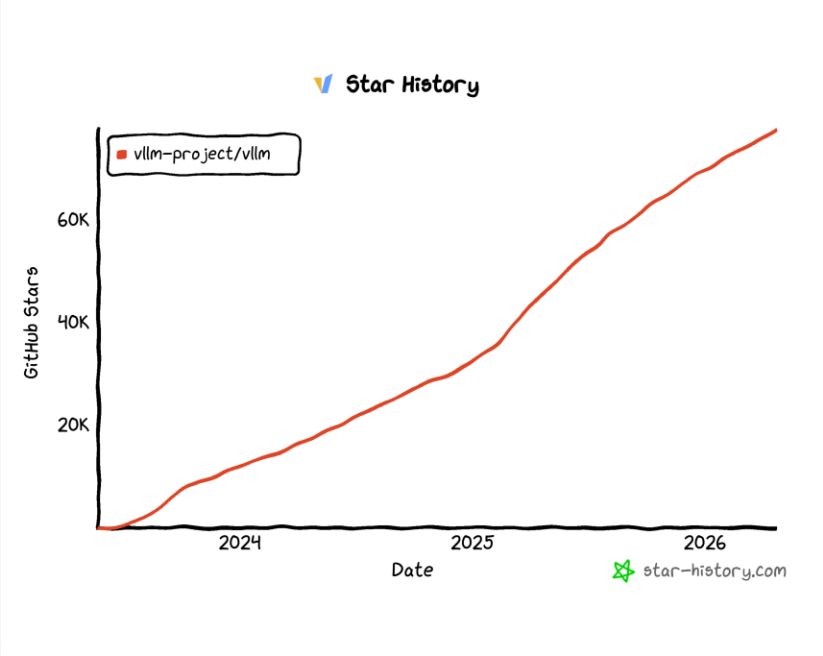
Backed across the ecosystem: NVIDIA, AMD, Meta, Red Hat, Google, Mistral, Embedded LLM and open source community

vLLM's Goal

Build the fastest and easiest-to-use

open-source LLM inference & serving engine

AI Inference’s current center of gravity



Most Popular LLM Serving Engine

- **77K+** GitHub stars, **800+** PRs/month
- **500K++** GPUs deployed 24/7
- **12.5K+** members in slack.vllm.ai

>2000 Contributors from >50 major companies

Broad Model Support (>100 arches)

Llama · DeepSeek · Qwen · GPT-OSS · Gemma · Mistral · Phi · Granite · Cohere

Flexible Device Parallelism

Tensor, Pipeline, Expert, Data, Context Parallel, Disagg Prefill/Decode, Disagg Encoder

Wide Hardware Support

CUDA · ROCm · Gaudi/XPU · TPU · Neuron · Ascend

Metal · MACA · RBLN · Spyre · Monaka

MLU · Kunlun · CPU

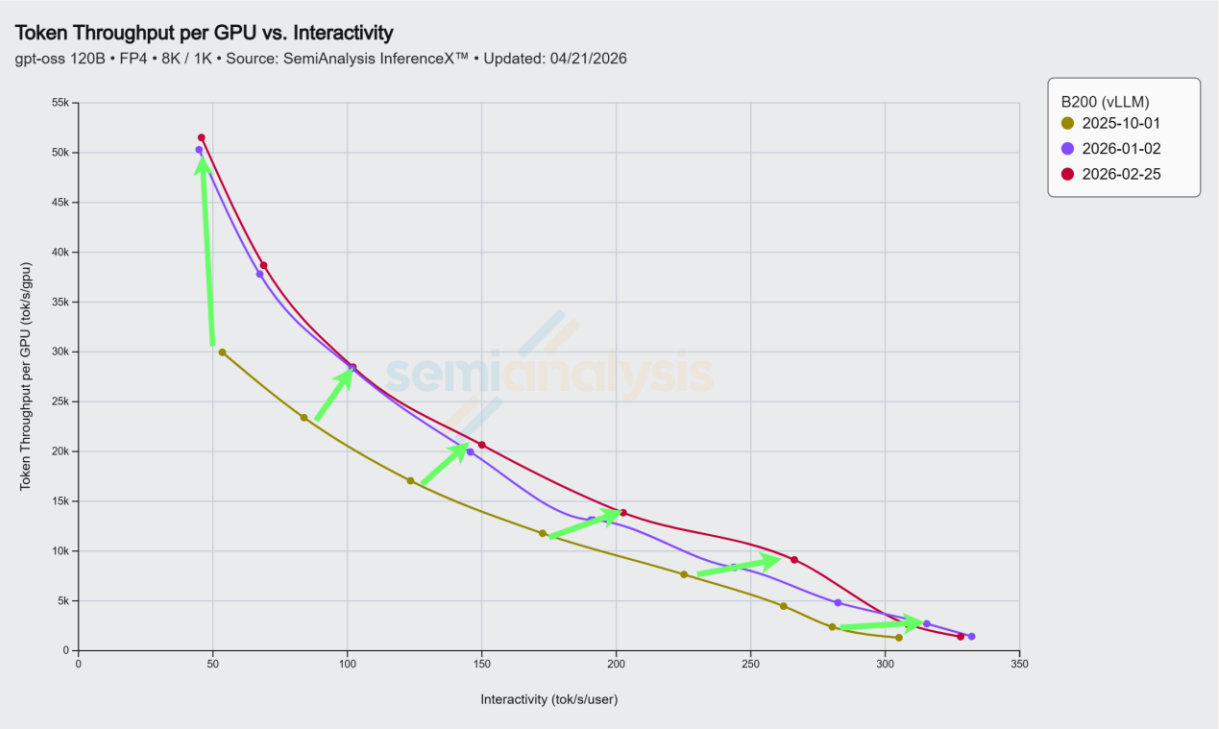
Diverse Project Ecosystem

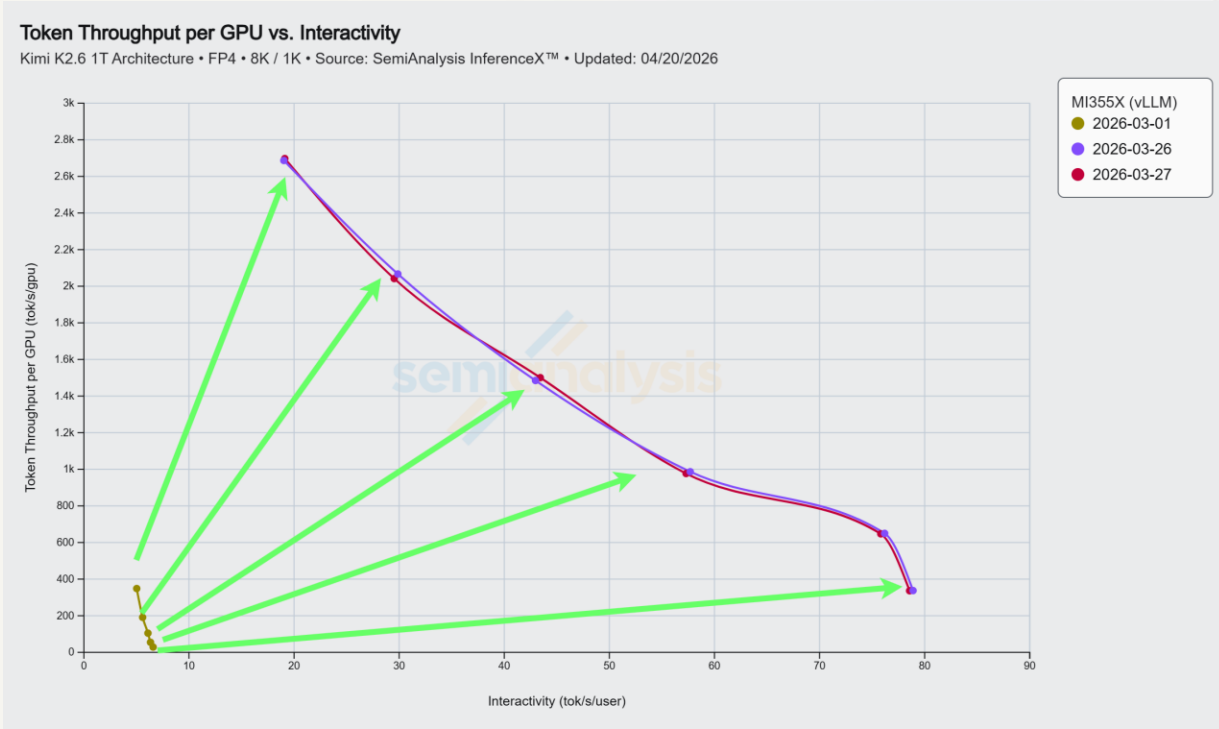
llm-d · vLLM-Omni · GuideLLM · vLLM Semantic Router

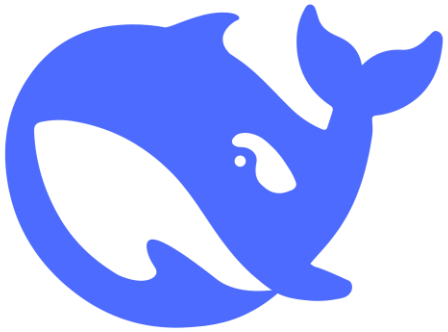
LLM Compressor · PECULATORS · Production Stack

vLLM Continuously Improves Performance

- GPTOSS on CUDA
- Kimi K2.6 on ROCm







DeepSeek-v4 Day 0

Pro

Flash

1M Context

vLLM

Pro + Flash in vLLM

Shared K/V

c4a / c128a

Sparse Attn

Local SWA

FP8 KV

FP4 Indexer

MTP

P/D Recipes

Secret Sauce

- Continuous Batching
- Torch compile
- Fusion Passes / Fusion Op
- PD Disaggregated Inference
- Quantization
- Speculative Decoding

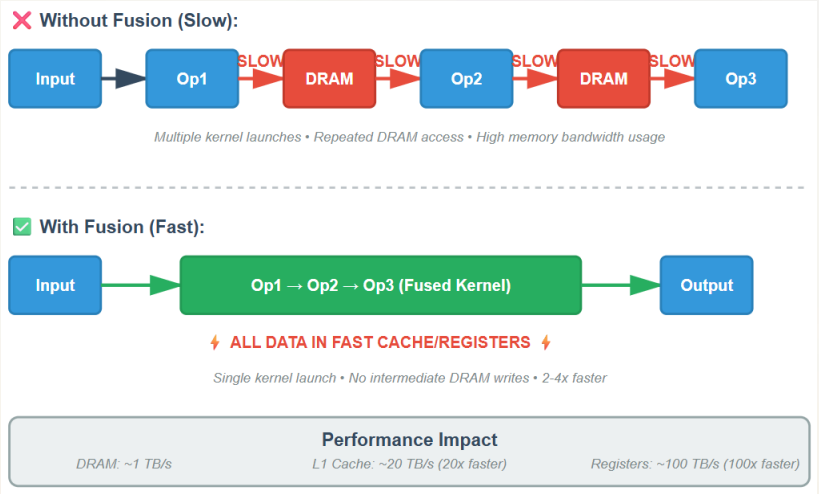
Fusion Passes/Fusion Ops

```
# File: /app/reviewpr3/fusionpasscionly/vllm/_aiter_ops.py:1236 in rms_norm2d_with_add, code: return torch.ops.vllm.rocm_a
rocm_aiter_rmsnorm2d_fwd_with_add_1 = torch.ops.vllm.rocm_aiter_rmsnorm2d_fwd_with_add(view_3, getitem_3, _get_data_attr_1,
getitem_8: "bf16[s72, 2048]" = rocm_aiter_rmsnorm2d_fwd_with_add_1[0]
getitem_9: "bf16[s72, 2048]" = rocm_aiter_rmsnorm2d_fwd_with_add_1[1];  rocm_aiter_rmsnorm2d_fwd_with_add_1 = None

# File: /app/reviewpr3/fusionpasscionly/vllm/model_executor/layers/quantization/kernels/scaled_mm/ScaledMMLinearKernel.py:
view_4: "bf16[s72, 2048]" = getitem_8.view(-1, 2048)

# File: /app/reviewpr3/fusionpasscionly/vllm/_aiter_ops.py:1451 in per_tensor_quant, code: return torch.ops.vllm.rocm_aite
rocm_aiter_per_tensor_quant_1 = torch.ops.vllm.rocm_aiter_per_tensor_quant(view_4, torch.float8_e4m3fnuz, None);  view_4 =
getitem_10: "f8e4m3fnuz[s72, 2048]" = rocm_aiter_per_tensor_quant_1[0]
getitem_11: "f32[1]" = rocm_aiter_per_tensor_quant_1[1];  rocm_aiter_per_tensor_quant_1 = None

# File: /app/reviewpr3/fusionpasscionly/vllm/model_executor/layers/quantization/kernels/scaled_mm/rocm.py:115 in apply_sca
rocm_per_tensor_float_w8a8_scaled_mm_impl_1: "bf16[s72, 576]" = torch.ops.vllm.rocm_per_tensor_float_w8a8_scaled_mm_impl(ge

# File: /app/reviewpr3/fusionpasscionly/vllm/model_executor/layers/quantization/kernels/scaled_mm/rocm.py:118 in apply_sca
narrow_1: "bf16[s72, 576]" = torch.narrow(rocm_per_tensor_float_w8a8_scaled_mm_impl_1, 0, 0, s72);  rocm_per_tensor_float_w
view_5: "bf16[s72, 576]" = narrow_1.view(s72, 576);  narrow_1 = None

# File: /app/reviewpr3/fusionpasscionly/vllm/model_executor/layers/quantization/kernels/scaled_mm/ScaledMMLinearKernel.py:
view_6: "bf16[s72, 2048]" = getitem_8.view(-1, 2048);  getitem_8 = None

# File: /app/reviewpr3/fusionpasscionly/vllm/_aiter_ops.py:1451 in per_tensor_quant, code: return torch.ops.vllm.rocm_aite
rocm_aiter_per_tensor_quant_2 = torch.ops.vllm.rocm_aiter_per_tensor_quant(view_6, torch.float8_e4m3fnuz, None);  view_6 =
getitem_12: "f8e4m3fnuz[s72, 2048]" = rocm_aiter_per_tensor_quant_2[0]
getitem_13: "f32[1]" = rocm_aiter_per_tensor_quant_2[1];  rocm_aiter_per_tensor_quant_2 = None
```

Fx.Graph

torch.ops.vllm.rocm_aiter_rmsnorm2d_fwd_with_add_quant

- Reduced memory traffic by quantizing the output during epilogue



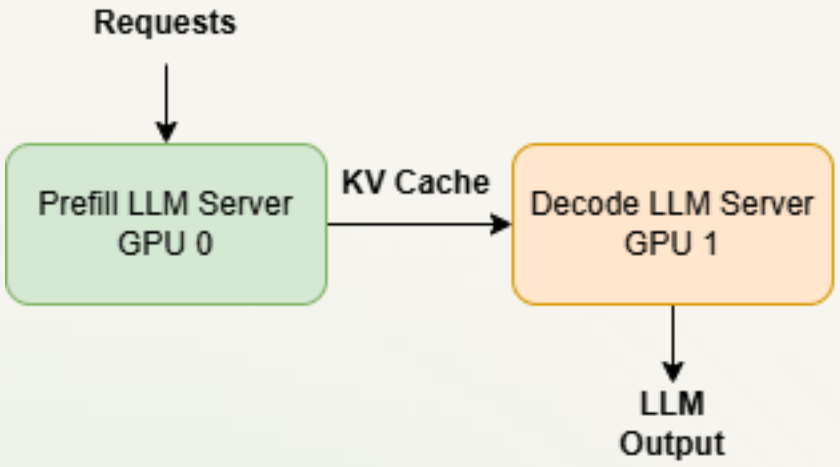
- Reduce kernel launch overhead
- Better cache utilization

PD Disaggregated Inference

- **Why Goodput, Not Throughput?**
- Raw throughput alone is misleading — a system can sustain high request rates while silently violating latency targets for most users.
- **Goodput** = maximum request rate (req/s) such that requests satisfy both **TTFT < *T_ttft*** and **ITL < *T_itl*** .
- Example: Our SLO targets: **TTFT < 1 second** and **ITL < 50 ms per token**.
- There are two phases for each requests
  - Prefill
  - Decode
- Both phases on same GPU compete for compute resources

PD Disaggregated Inference

- Avoid compute resource contention between prefill and decode phase



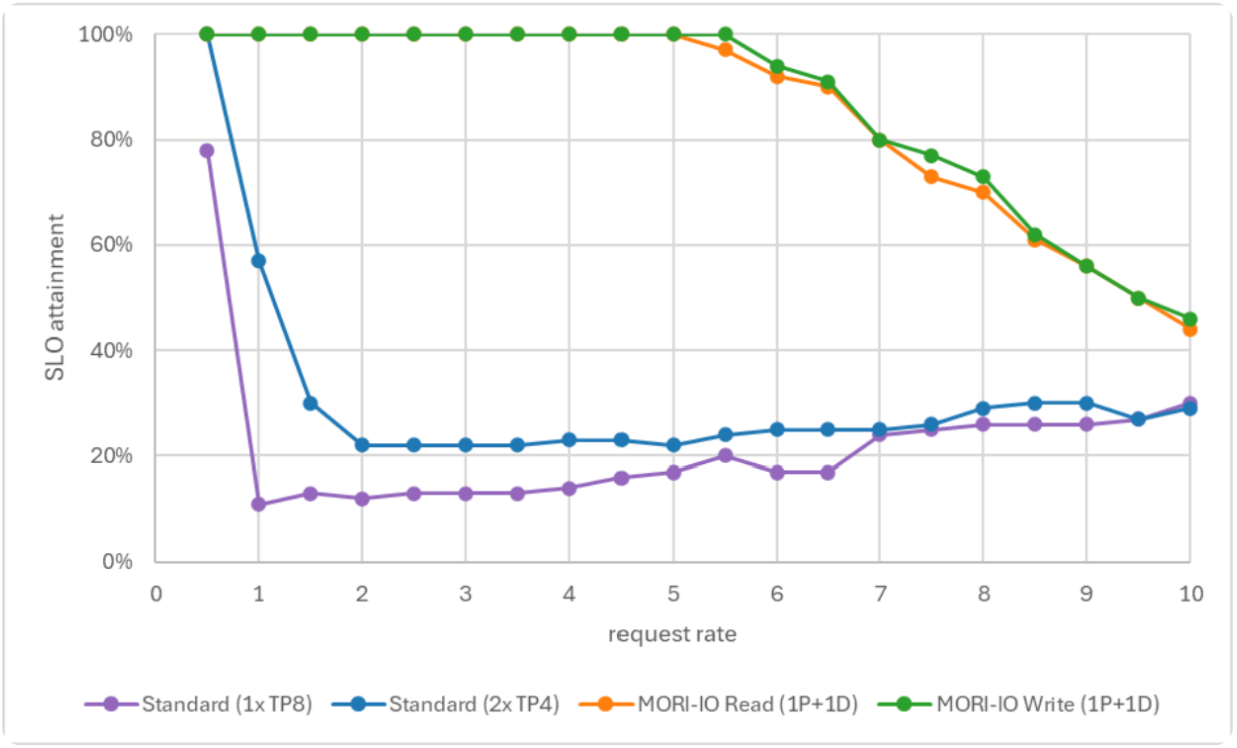


Figure 4: SLO attainment (% of requests meeting both TTFT and ITL targets) across request rates. Both disaggregated modes show higher SLO attainment than all standard serving configurations across all tested request rates.

[Next-Level Inference: Why Your Single-Node vLLM Setup Needs Prefill-Decode Disaggregation | vLLM Blog]

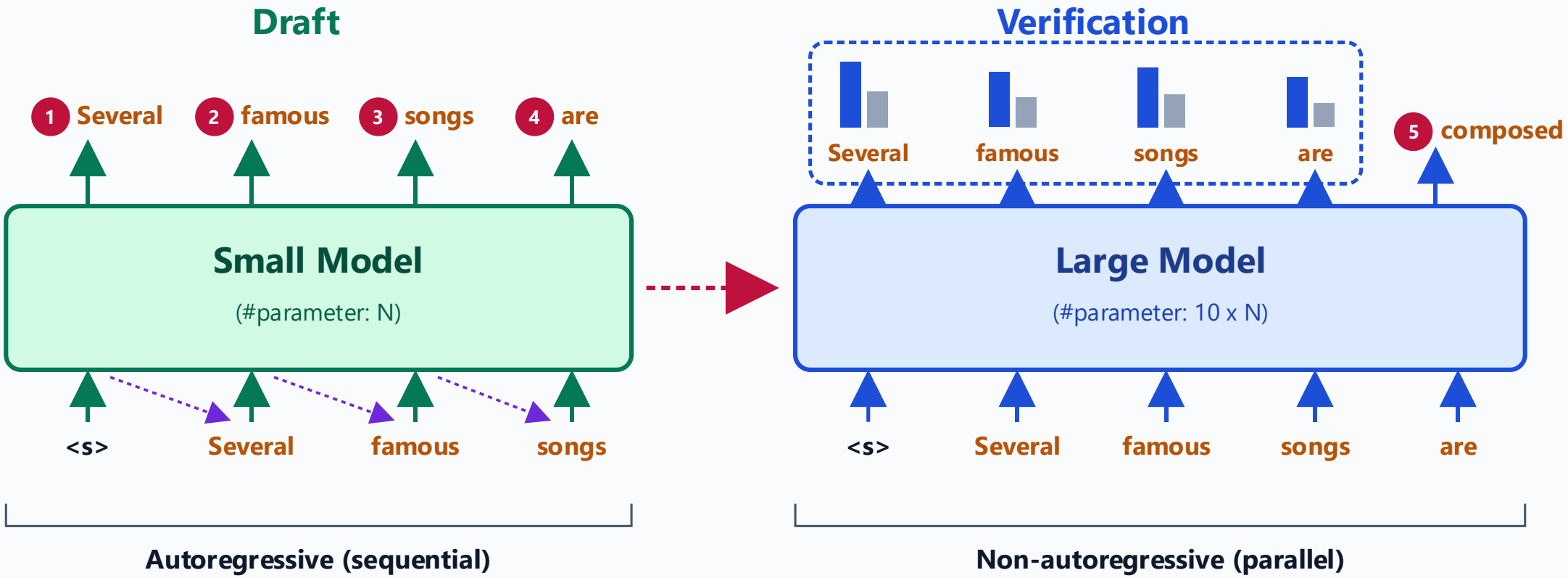
Quantization

Weights	KV Cache
FP8 (Hardware accelerated)	FP8
MXFP4/ NVFP4 (Hardware accelerated)	FP8 Per-Head-Per-Token
W4A16	INT8 Per-Head-Per-Token
INT8	TurboQuant

- Lower GPU memory requirements
- Data format compatible with Tensor Cores

- Lower GPU memory requirements for KV Cache
- Serve longer context length

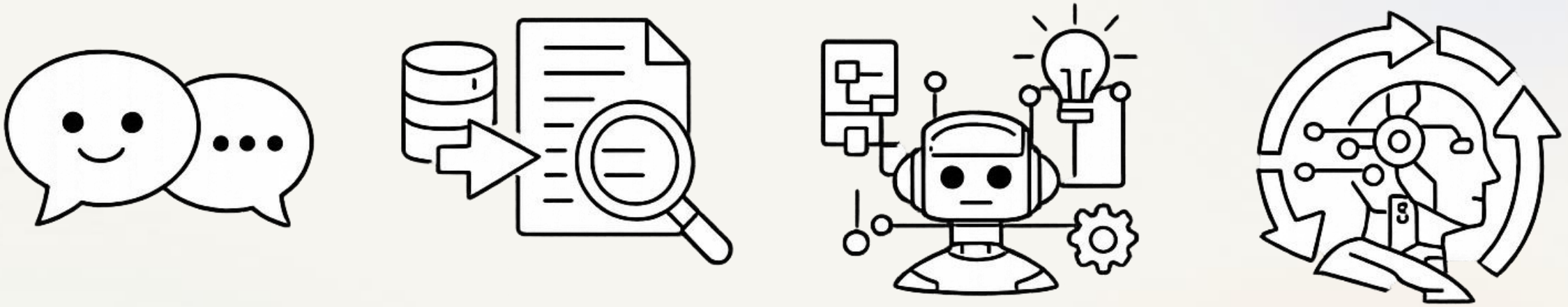
Speculative Decoding



Speculative Decoding: draft tokens proposed by Small Model are verified in parallel by Large Model

What are agents and tools?

How LLM capability progresses over time



Chat only | Basic Retrieval | Agentic Framework | Self-adapting Agents

time

What are Agents & Tools?

AGENT

TOOLS — FUNCTIONS THE AGENT CAN CALL

Query Database
Fetch customers, orders and revenue from the database

Build Charts
Turn data into bar, line and pie charts

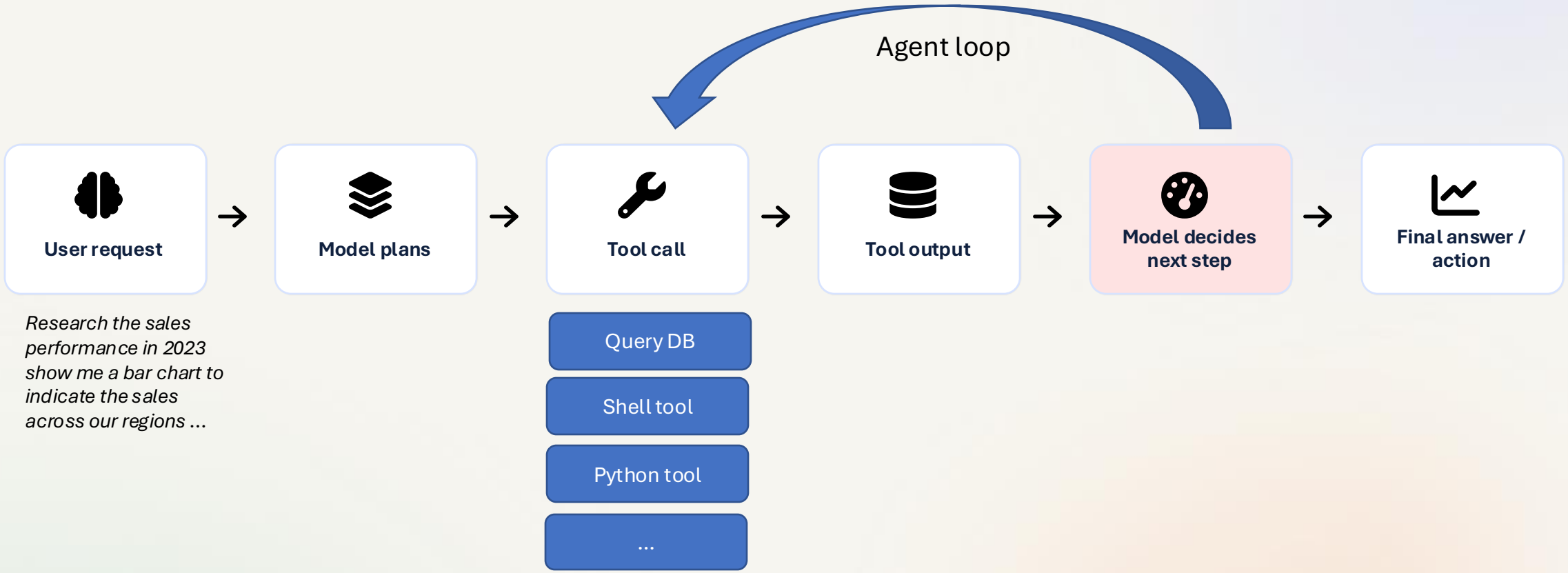
Write Reports
Compile findings into a professional PDF

Run Code
Process and transform data with Python

Manage Files
Create folders, move files and compress archives

... and many more ...

Agentic workflows turn one request into many inference calls



- After each tool result, the model must read new context and decide what to do next.
- One new tool result is one new model reasoning pass

Why every tool step hits the inference engine harder every turn

Classic chat pattern

1 user request

↓

1 model forward pass

↓

1 model response

Concurrency stays near the number of active users.

Agentic pattern

Plan (forward pass 1)

↓

Tool A

Reflect (forward pass 2)

↓

Tool B

Answer (forward pass 3)

One user request → multiple model forward passes within a short window.

Throughput at high concurrency is a necessity, not a luxury.

Serving models with vLLM

```
vllm serve stepfun-ai/Step-3.5-Flash-FP8 \
  --quantization fp8 \
  --tensor-parallel-size 2 \
  --enable-expert-parallel \
  --reasoning-parser step3p5 \
  --enable-auto-tool-choice \
  --tool-call-parser step3p5 \
  --hf-overrides '{"num_nextn_predict_layers": 1}' \
  --speculative-config '{"method": "step3p5_mtp", "num_speculative_tokens": 1}' \
  --disable-cascade-attn
```

- parallelism (TP, DP, EP, PP)
- reasoning / thinking
- tool / function calling
- quantization
- context length
- memory & performance
- …

How to choose a model?

https://recipes.vllm.ai

Pick a model, adjust for your GPUs, copy the `vllm serve` line that runs. Community-maintained recipes for NVIDIA H100/H200/B200/B300, Grace-Blackwell, and AMD MI300X/MI325X/MI355X. Full vLLM compatibility →

LATEST RECIPES newest 8

DeepSeek-V4-Pro
deepseek-ai

1600B/49B | FP8 | 1M ctx | Text

1M efficient long-context attention

Qwen3.6-27B
Qwen

27B | BF16 | 262K ctx | Multimodal | Text

Qwen3.6 flagship dense — single-GPU FP8 or 2x GPU BF16

Kimi-K2.6
moonshotai

1T/32B | INT4 | 262K ctx | Multimodal | Text

Multimodal agentic MoE model with DeepSeek-V3 backbone and MLA attention

Hy3-preview
tencent

295B/21B | BF16 | 262K ctx | Text

Hunyuan Hy3-preview MoE — 295B/21B on 8×H200 or 8×H20-3e(141GB) with MTP

MiniMax-M2.7
MiniMaxAI

230B/10B | FP8 | 197K ctx | Text

Latest M2 series release; verified accuracy on AIME25, GPQA-D, GSM8K; 196K context

GLM-5.1
zai-org

744B/40B | BF16 | 203K ctx | Text

Refreshed GLM-5 series MoE with improved reasoning, coding, and agentic performance

Trinity-Large-Thinking
arcee-ai

398B/13B | BF16 | 262K ctx | Text

Arcee AI's reasoning-focused sparse MoE (AfmoeForCausalLM) with structured <think> traces and...

gemma-4-26B-A4B-it
Google

26B/4B | BF16 | 131K ctx | Multimodal | Text

MoE multimodal model — 26B total / 4B active, 128 experts with top-8 routing

BROWSE BY PROVIDER

- **Arcee AI** arcee-ai — 1 recipe →
- **Ernie (Baidu)** baidu — 2 recipes →
- **Seed (ByteDance)** ByteDance-Seed — 1 recipe →
- **DeepSeek** deepseek-ai — 9 recipes →
- **Google** Google — 5 recipes →
- **inclusionAI** inclusionAI — 1 recipe →
- **InternLM** internlm — 1 recipe →
- **Jina AI** jinaai — 1 recipe →
- **Meituan LongCat** meituan-longcat — 1 recipe →
- **Meta** meta-llama — 3 recipes →
- **Microsoft** microsoft — 1 recipe →
- **MiniMax** MiniMaxAI — 4 recipes →
- **Mistral AI** mistralai — 3 recipes →
- **Moonshot AI** moonshotai — 5 recipes →
- **NVIDIA** nvidia — 2 recipes →
- **OpenAI** openai — 1 recipe →
- **InternVL (OpenGVLab)** OpenGVLab — 1 recipe →
- **PaddlePaddle** PaddlePaddle — 1 recipe →

PROVIDERS

- Arcee AI
- Ernie (Baidu)
- Seed (ByteDance)
- DeepSeek
- Google
  - gemma-4-26B-A4B-it
  - gemma-4-31B-it
  - gemma-4-E2B-it
  - gemma-4-E4B-it
  - translategemma-27b-it
- inclusionAI
- InternLM
- Jina AI
- Meituan LongCat
- Meta
- Microsoft
- MiniMax
- Mistral AI
- Moonshot AI
- NVIDIA

Verified on AMD MI300X

Copy | cURL | Bench

```
vllm serve google/gemma-4-26B-A4B-it \
  --tensor-parallel-size 1 \
  --enable-auto-tool-choice \
  --tool-call-parser gemma4 \
  --chat-template examples/tool_chat_template_gemma4.jinja \
  --reasoning-parser gemma4
```

HARDWARE

NVIDIA	H100 8×80G	H200 8×141G	B200 8×180G	GB200 NVL4 4×192G	B300 8×268G	GB300 NVL4 4×288G
AMD	MI300X 8×192G	MI325X 8×256G	MI355X 8×288G			
GOOGLE	TPU v7 (Ironwood) 1×192G	TPU v6e (Trillium) 8×32G				

This recipe runs on 1 of 8 GPUs on the selected node — add `--tensor-parallel-size N` via Advanced to scale up.

VARIANT ⓘ

BF16 64 GB

STRATEGY

Tensor Parallel | Tensor + Expert Parallel | Data + Expert Parallel

Single-node tensor parallel. Splits the model across all local GPUs. TP size is set to the GPU count at deploy time. The simplest multi-GPU strategy — works for all model architectures.

Latency oriented

NODES

Single-node | Multi-node (example: 2) 16×GPU

FEATURES

Tool Calling | Reasoning | Text Only

ADVANCED — performance tuning

Demo

Backed across the ecosystem: NVIDIA, AMD, Meta, Red Hat, Google, Mistral, Embedded LLM and open source community

The Northwind Database

Specialty Food Import/Export · 1994–1997

A real-world business dataset — customers, orders, products, employees across 21 countries

93 Customers
21 countries

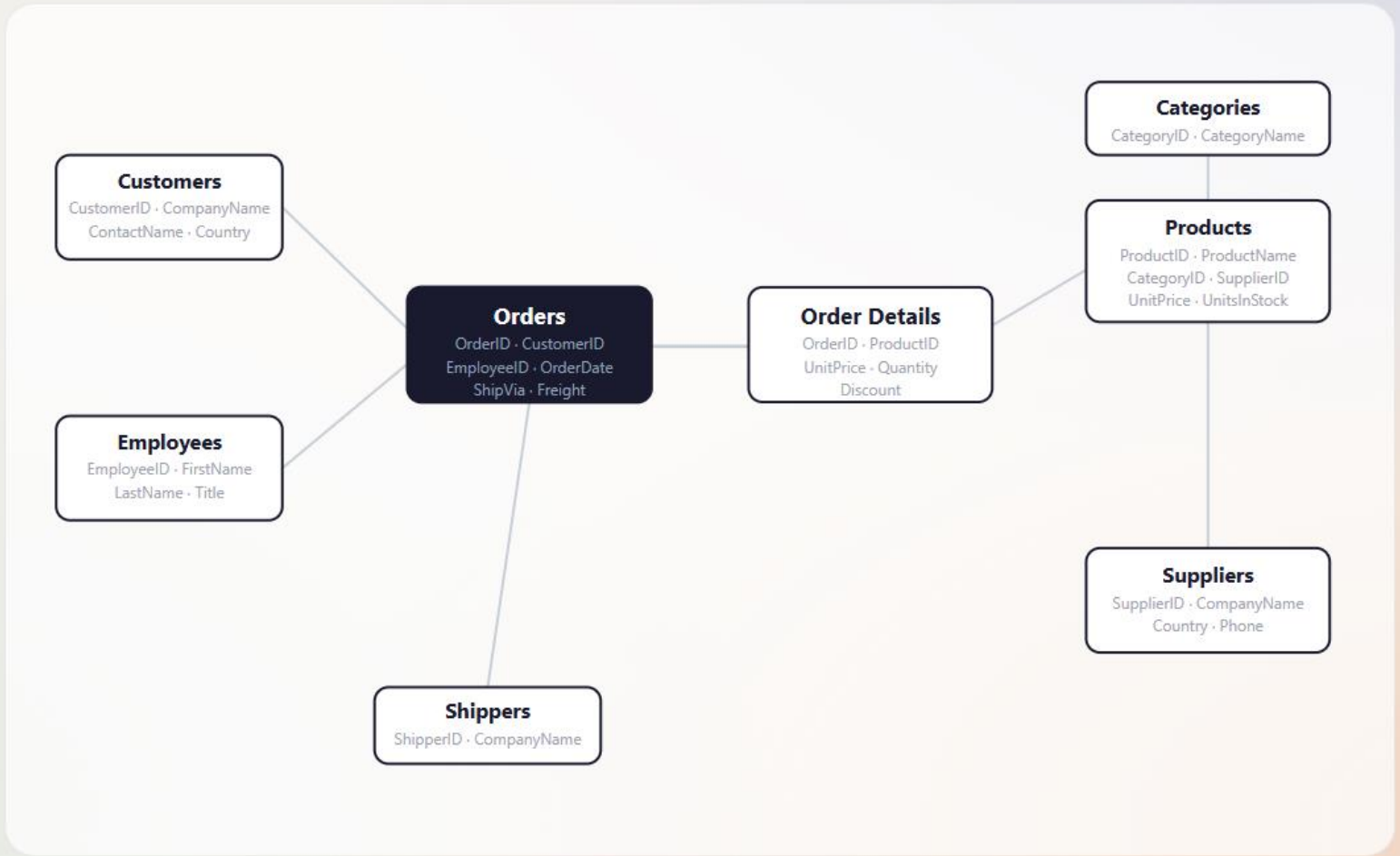
830 Orders
~2,155 line items

77 Products
8 categories

29 Suppliers
16 countries

9 Employees
Sales team

SCHEMA



https://github.com/jpwhite3/northwind-SQLite3/raw/main/dist/northwind.db

Meet Alex 🤖, your AI Business Analyst

Ask a business question — Alex queries the data, builds charts, writes reports, and organises files. Automatically.

LEVEL 1

Ask in Plain English

Type a business question. Alex writes and runs the SQL query for you — no code needed.

→ "Who are our top 5 customers?"

LEVEL 2

Instant Visualisation

Alex decides the best chart type and renders it live — bar, line, pie, and more.

→ "Show revenue by category as a chart"

LEVEL 3

Full PDF Report

Multi-step investigation: Alex drills into the data and produces a professional report with charts.

→ "Why is Germany revenue declining?"

LEVEL 4

Your AI Co-worker

Alex creates folders, moves files, zips archives, and generates reports — all in one command.

→ "Organise all PO files by supplier"

How Alex Gets Things Done

Five tools — each one unlocking a new level of capability

`execute_sql`

Query the Database

Reads and searches your business data — customers, orders, revenue, products — in real time.

L1 L2 L3 L4

`render_chart`

Build Charts

Turns data into bar, line, and pie charts automatically — Alex picks the right type.

L1 L2 L3 L4

`generate_report`

Write Reports

Compiles findings into a professional PDF report — charts, tables, and written analysis included.

L1 L2 L3 L4

`execute_python`

Process & Transform

Runs code to sort, filter, rename, and move data — handles tasks too complex for plain SQL.

L1 L2 L3 L4

`run_shell`

Manage Files

Creates folders, moves files, and compresses archives — directly on your computer, under your supervision.

L1 L2 L3 L4

https://github.com/EmbeddedLLM/northwind-demo

How a Tool is Built

A tool = a Python function + a wrapper that registers it with the agent framework (Kosong)

1 The Function — does the actual work

Plain Python: connect to SQLite, run query, format results as a table

```python
# ── Sync tool implementations ──────────────────────────────────────


def _execute_sql(query: str) -> str:
    try:
        conn = sqlite3.connect(DB_PATH)
        cur = conn.execute(query)
        rows = cur.fetchall()
        cols = [d[0] for d in cur.description] if cur.description else []
        conn.close()
    except Exception as e:
        return f"SQL ERROR: {e}"
    if not rows:
        return "Query returned no results."
    cw = [
        max(len(str(c)), max(len(str(r[i])) for r in rows)) for i, c in enumerate(cols)
    ]
    sep = "+-" + "-+-".join("-" * w for w in cw) + "-+"
    header = "| " + " | ".join(str(c).ljust(w) for c, w in zip(cols, cw)) + " |"
    body = [
        "| " + " | ".join(str(v).ljust(w) for v, w in zip(row, cw)) + " |"
        for row in rows
    ]
    return "\n".join(
        [sep, header, sep]
        + body
        + [sep, f"({len(rows)} row{'s' if len(rows) != 1 else ''})"]
    )

```

→ wrapped into

2 The Wrapper — registers it as an Agent Tool

Kosong `CallableTool2` — gives the tool a name, description, and schema the LLM can read

```python
# ── kosong CallableTool2 classes ──────────────────────────────────


class SqlTool(CallableTool2[SqlParams]):
    name = "execute_sql"
    description = (
        "Execute a SQLite SELECT query against Northwind. Returns table or SQL ERROR."
    )
    params = SqlParams

    def __init__(self, emit):
        super().__init__()
        self._emit = emit

    async def __call__(self, p: SqlParams) -> ToolReturnValue:
        await self._emit({"type": "sql_call", "query": p.query})
        loop = asyncio.get_event_loop()
        result = await loop.run_in_executor(_executor, _execute_sql, p.query)
        await self._emit(
            {"type": "sql_result", "result": result, "success": "ERROR" not in result}
        )
        return ToolOk(output=result)
```

Tools Available to Agents

Agents decide which tool to call, when to call it, and what to do with the result

Read & Write Files

Documents · data · configs

Execute Code

Scripts · shell · pipelines

Search

Files · code · knowledge

Web Access

Fetch · browse · research

Database / SQL

Query · retrieve · store

Email & Messaging

Email · Slack · WhatsApp

Calendar

Schedule · remind · coordinate

Task Management

Plan · track · coordinate

Sub-agents & APIs

Delegate · integrate · extend

Tools Across Sectors

Same agent pattern — different tools plugged in per industry

Sales & CRM

- Customer history lookup
- Pipeline & deal updates
- Follow-up email & call scheduling

Finance & Accounting

- Invoice OCR & matching
- Expense approval workflows
- Budget reports & anomaly alerts

HR & People

- Leave & attendance management
- Policy Q&A (internal knowledge)
- Onboarding & training tracker

IT & DevOps

- Ticket creation & escalation
- Log analysis & incident summary
- Code search & deploy triggers

Operations & Supply

- Stock level checks & PO triggers
- File & document organiser
- Procurement summary reports

Analytics & Monitoring

- KPI dashboards on demand
- IoT sensor monitoring & alerts
- Anomaly detection & team notify

Customer Service

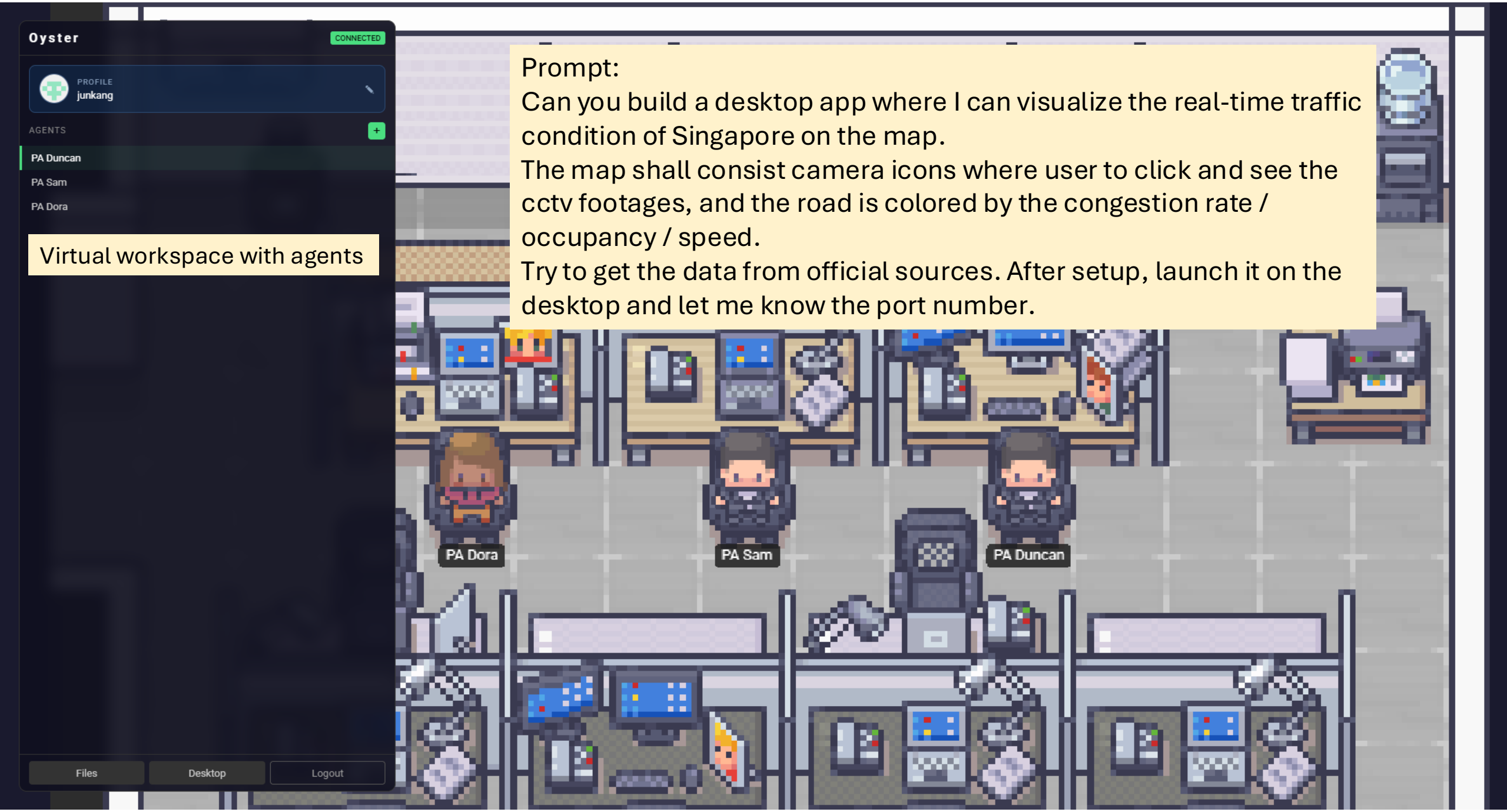
- Email history & thread recall
- FAQ & knowledge base lookup
- Ticket routing & status updates

Retail & E-commerce

- Product search & recommendations
- Order status & return processing
- Promo & pricing lookups

Legal & Compliance

- Contract review & clause search
- Policy & regulation lookup
- Audit trail & approval records


Oyster
CONNECTED
PROFILE
junkang
AGENTS
PA Duncan
PA Sam
PA Dora
Virtual workspace with agents
Files
Desktop
Logout
Prompt:
Can you build a desktop app where I can visualize the real-time traffic condition of Singapore on the map.
The map shall consist camera icons where user to click and see the cctv footages, and the road is colored by the congestion rate / occupancy / speed.
Try to get the data from official sources. After setup, launch it on the desktop and let me know the port number.
PA Dora
PA Sam
PA Duncan

Singapore Traffic Viewer

Live CCTV locations use the official LTA traffic-images feed via **data.gov.sg**. Colored road segments are a **local estimate** from those official CCTV images because the official LTA live speed-band feed was not anonymously accessible in this environment.

Official CCTV | Estimated congestion overlay

Source timestamp: **2026-04-26T18:42:15+08:00**
Cameras: **90**
Colored road segments: **88**
Last refreshed (UTC): **2026-04-26T10:43:05.402125+00:00**

Refresh now

Legend

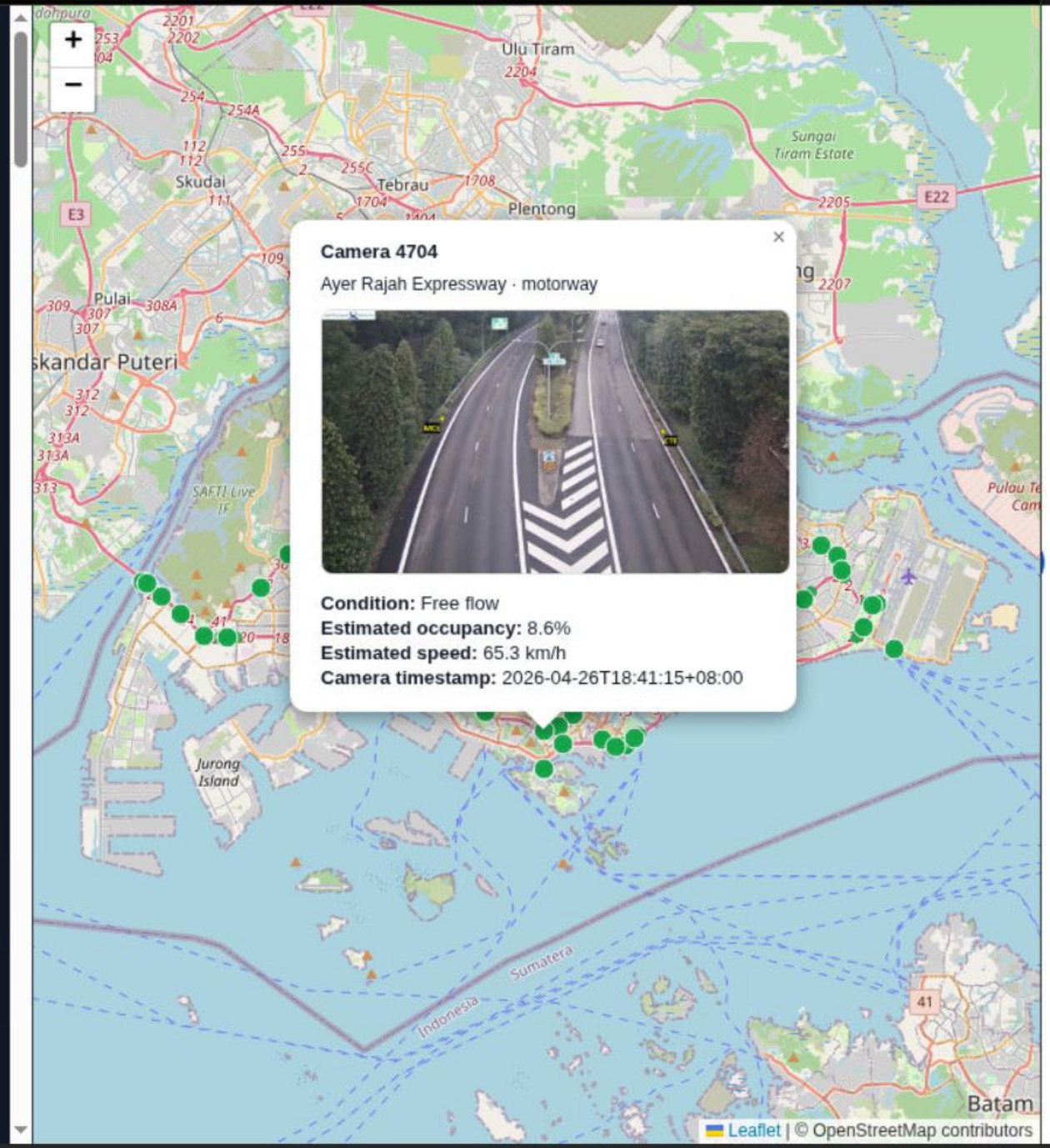
- Free flow
- Moderate
- Heavy
- Severe
- Unknown

Cameras

Camera 1001 · Free flow
East Coast Parkway · 64.3 km/h

Camera 1002 · Free flow
Pan-Island Expressway · 64.8 km/h

Camera 1003 · Free flow
Jalan Kolam Ayer · 62.2 km/h



Camera 4704

Ayer Rajah Expressway · motorway

Condition: Free flow
Estimated occupancy: 8.6%
Estimated speed: 65.3 km/h
Camera timestamp: 2026-04-26T18:41:15+08:00

Thank You

Backed across the ecosystem: NVIDIA, AMD, Meta, Red Hat, Google, Mistral, Embedded LLM and the open source community