

# Rust + WebAssembly: 构筑大模型生态的基础设施

刘 鑫 ([sam@secondstate.io](mailto:sam@secondstate.io))

Second State 工程师

WasmEdge Maintainer

# 分享事项

- Python + Docker 组合存在的问题
- Rust + WebAssembly组合的优势
- 实践项目分享

# 现状： Python + Docker 组合 – 1

- Python
  - 易：易学、易用
  - 大：社区活跃、生态繁荣、资源丰富
  - 广：适用于各种类型的AI任务，深入每一个环节
- Docker Container
  - 可移植性：利于分发
  - 隔离性：保证了安全性和稳定性
  - 可扩展性：资源利用率高



# 现状： Python + Docker 组合 – 2

- 痛点1: Python 在性能、并行性、内存管理等处于劣势

**Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices.** Each version represents a successive refinement of the original Python code. “Running time” is the running time of the version. “GFLOPS” is the billions of 64-bit floating-point operations per second that the version executes. “Absolute speedup” is time relative to Python, and “relative speedup,” which we show with an additional digit of precision, is time relative to the preceding line. “Fraction of peak” is GFLOPS relative to the computer’s peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

## COMPUTER SCIENCE

### There’s plenty of room at the Top: What will drive computer performance after Moore’s law?

Charles E. Leiserson, Neil C. Thompson\*, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, Tao B. Schardl



# 现状： Python + Docker 组合 – 3

- 痛点2： Python + C/C++/Rust 混合编程维护成本高、移植能力差

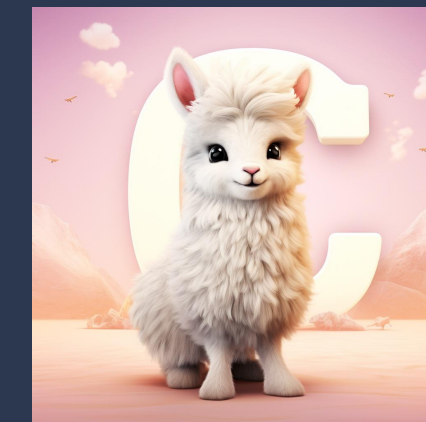
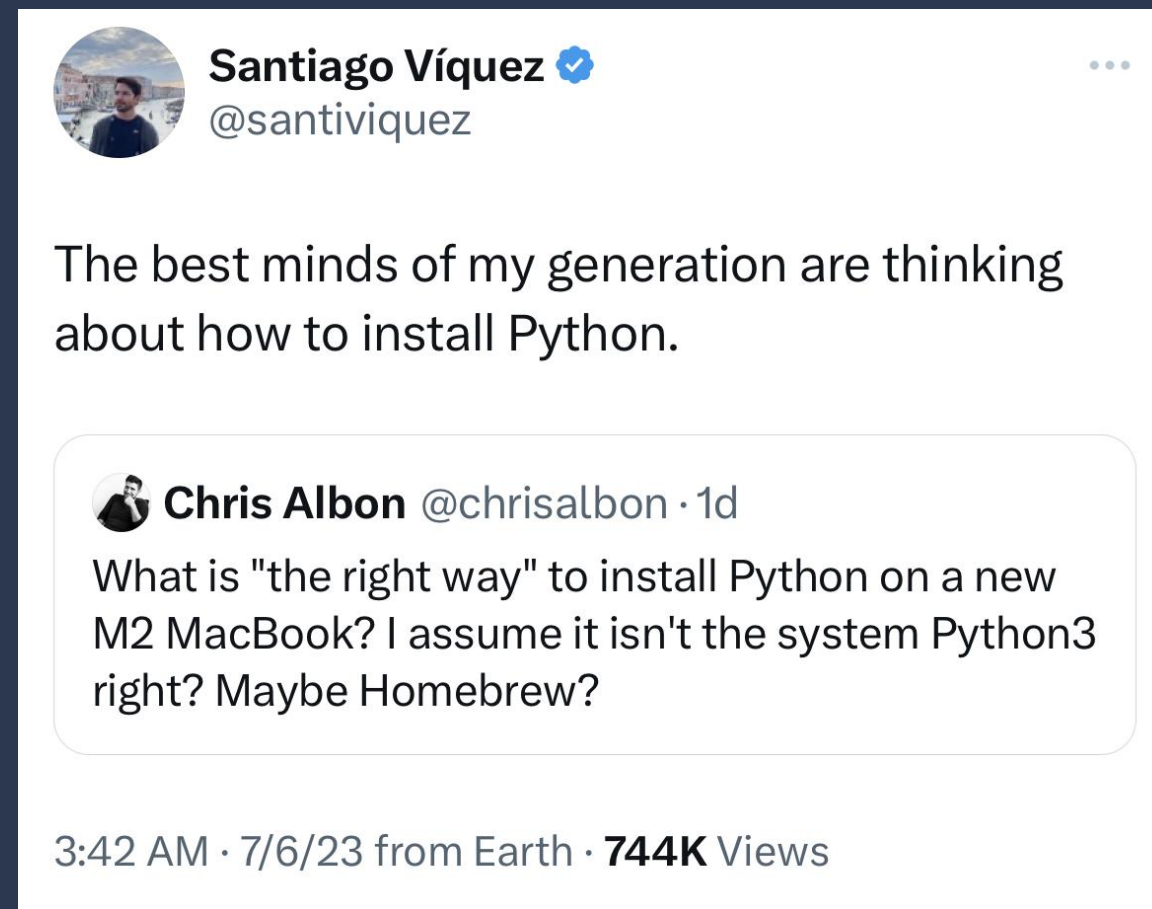
```
466     py::class_<Graph>(m, "Graph")
467         .def(py::init<>(&Graph::create),
468             py::arg("name"))
469         .def_static("deserialize",
470                     &Graph::deserialize,
471                     py::arg("fname"),
472                     py::return_value_policy::move)
473         .def("get_name",
474             &Graph::get_name)
475         .def("create_op",
476             &helper::add_op,
477             py::arg("name"),
478             py::arg("kind"),
479             py::arg("attrs") = py::none(),
480             py::arg("input_ops") = py::none(),
481             py::arg("subgraph") = py::none(),
482             py::return_value_policy::reference_internal)
483         .def("create_const_op",
484             &helper::add_const_op,
485             py::return_value_policy::reference_internal)
486         .def("remove_op",
487             &Graph::remove_op,
488             py::arg("op"))
```

```
761     py::class_<Tensor, std::unique_ptr<Tensor, py::nodelete>>(m, "Tensor")
762         .def(py::init<>([])(xir::Tensor* tensor) { return tensor; }))
763         .def_static("clone",
764                     py::overload_cast<const Tensor*>(&Tensor::clone))
765         .def_property("name",
766                     &Tensor::get_name,
767                     &Tensor::rename)
768         .def_property_readonly("ndim",
769                                 &Tensor::get_dim_num)
770         .def_property_readonly("dims",
771                                 &Tensor::get_shape)
772         .def_property_readonly(
773             "dtype",
774             [](const Tensor* tensor) {
775                 return helper::to_lower(tensor->get_data_type().to_string());
776             })
777         .def_property_readonly("producer",
778                                 py::overload_cast<>(&Tensor::get_producer),
779                                 py::return_value_policy::reference_internal)
780         .def("get_element_num",
781             &Tensor::get_element_num)
782         .def("get_data_size",
783             &Tensor::get_data_size)
```

上述代码片段源自 [https://github.com/Xilinx/Vitis-AI/blob/master/src/vai\\_runtime/xir/src/python/wrapper/wrapper.cpp](https://github.com/Xilinx/Vitis-AI/blob/master/src/vai_runtime/xir/src/python/wrapper/wrapper.cpp)



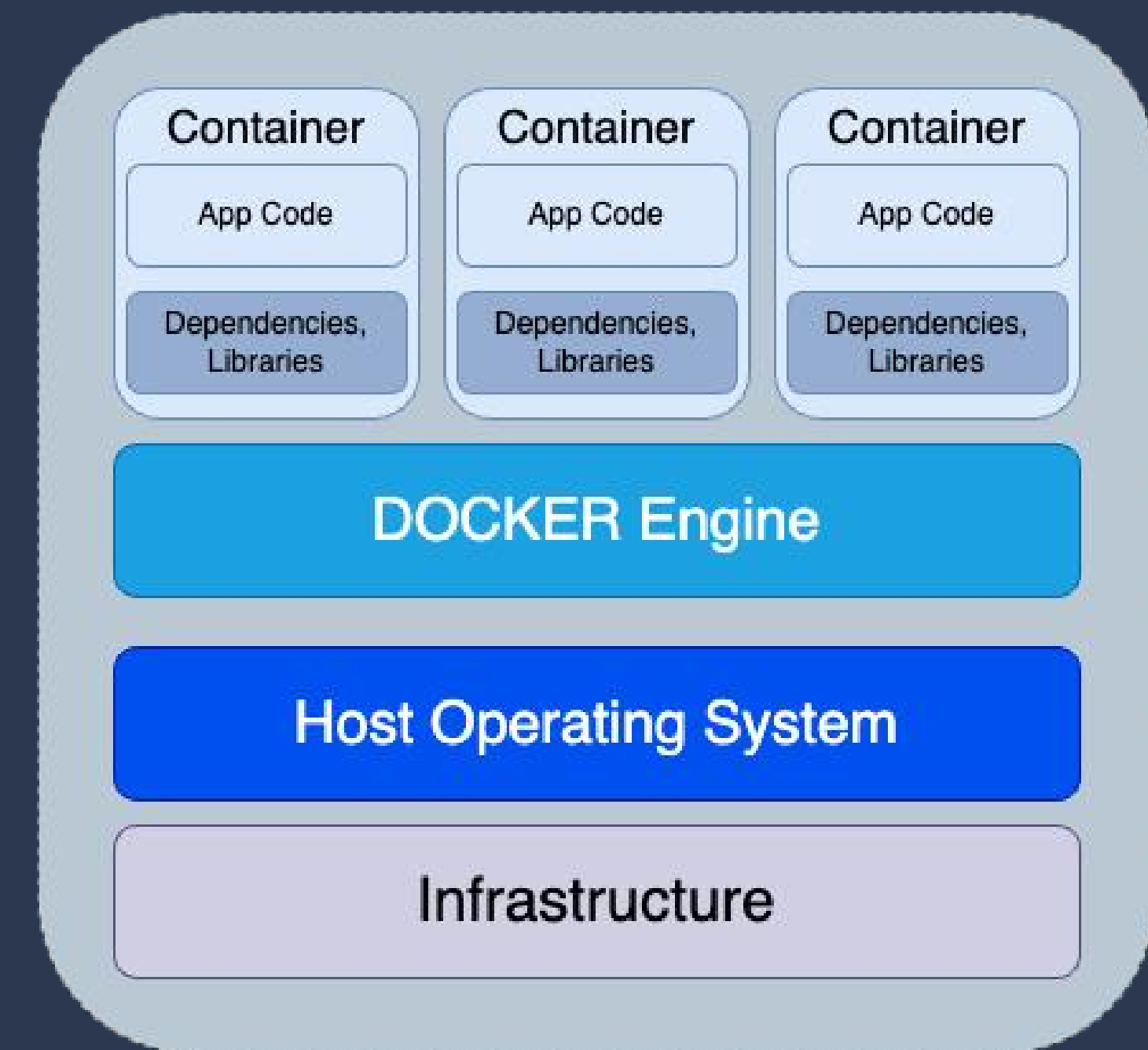
# 现状： Python + Docker 组合 – 4



LangChain图片引用自 <https://deepwisdom.feishu.cn/docx/XDYbdhq7ro0KPexuXUkcB6mynmh#VygBdZS9poOgNfxKoFrcm6Bensh>

# 现状： Python + Docker 组合 – 5

- 痛点3： Docker Container 的局限性
  - 冷启动性能：秒级
  - 磁盘空间：GiB级
  - 硬件加速器的支持：需要特定版本
  - 可移植性：依赖于CPU架构
  - 安全性：依赖于宿主OS的用户权限



部分文字引用自[https://wasmedge.org/wasm\\_linux\\_container](https://wasmedge.org/wasm_linux_container)

图片源自<https://medium.com/@shivraj.jadhav82/webassembly-wasm-docker-vs-wasm-275e317324a1>



# AGI最佳： Rust + WebAssembly – 1

- Rust — AGI 时代的最佳选择
  - 性能与内存安全兼备
  - 无惧并发
  - 强大而富有表现力的类型系统
  - 现代化的包管理工具Cargo
  - 快速成长的生态系统： ndarray, llm, candle, burn, ...





# AGI最佳： Rust + WebAssembly – 2

- WebAssembly — 更快、更轻、更安全

Aspect	Container	WASM
Size	10s or 1000 MBs	Few MBs
Performance	Far away from native	Native Speed
Startup Time	Seconds	Miliseconds
Runs in Web Browser	No	Yes
Cross Platform/Portability	Specific to System Architecture	High
System Interaction	Container contains OS and File System	Use WASI to access OS
Standards	OCI	W3C and OCI

WASM Vs Docker Performance Chart

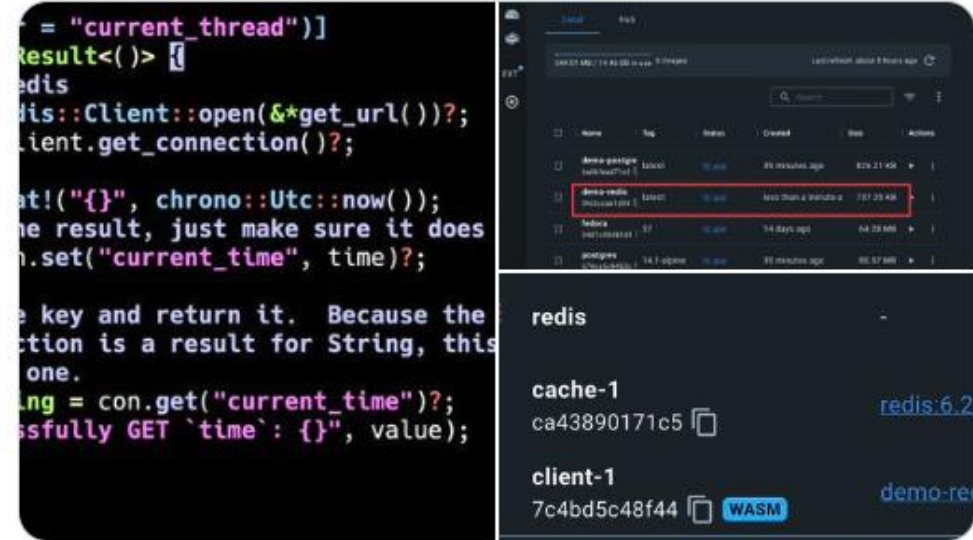
图片源自 <https://medium.com/@shivraj.jadhav82/webassembly-wasm-docker-vs-wasm-275e317324a1>

**wasmedge** @realwasmedge · Feb 12

A complete Redis app running inside a secure Wasm container managed by Docker + #Wasm. Total app size is 0.7MB and starts in milliseconds. (A comparable Linux container app for #redis is easily 50+MB).

[github.com/WasmEdge/wasme...](https://github.com/WasmEdge/wasme...)

@Redisinc @Docker



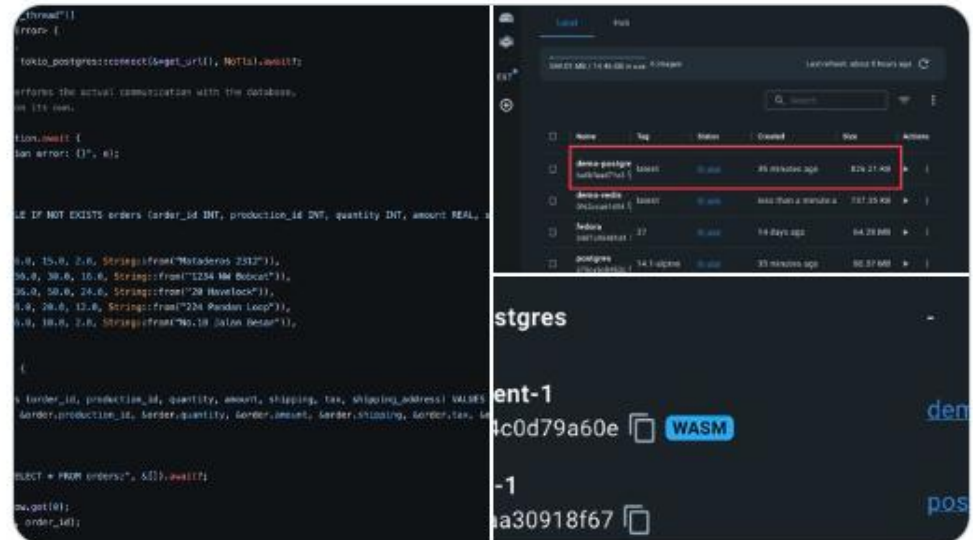
The screenshot shows a code editor with Redis client code in Rust/Wasm and a Docker container list. The container list shows a container named 'redis' with a size of 0.7 MB, which is significantly smaller than the 50+ MB mentioned in the tweet.

**wasmedge** @realwasmedge · Feb 15

A #PostgreSQL client app running inside a secure Wasm container managed by Docker + #Wasm. Total app size is 0.8MB. It runs anywhere and starts in milliseconds. (A comparable Linux container is easily 50MB).

[github.com/WasmEdge/wasme...](https://github.com/WasmEdge/wasme...)

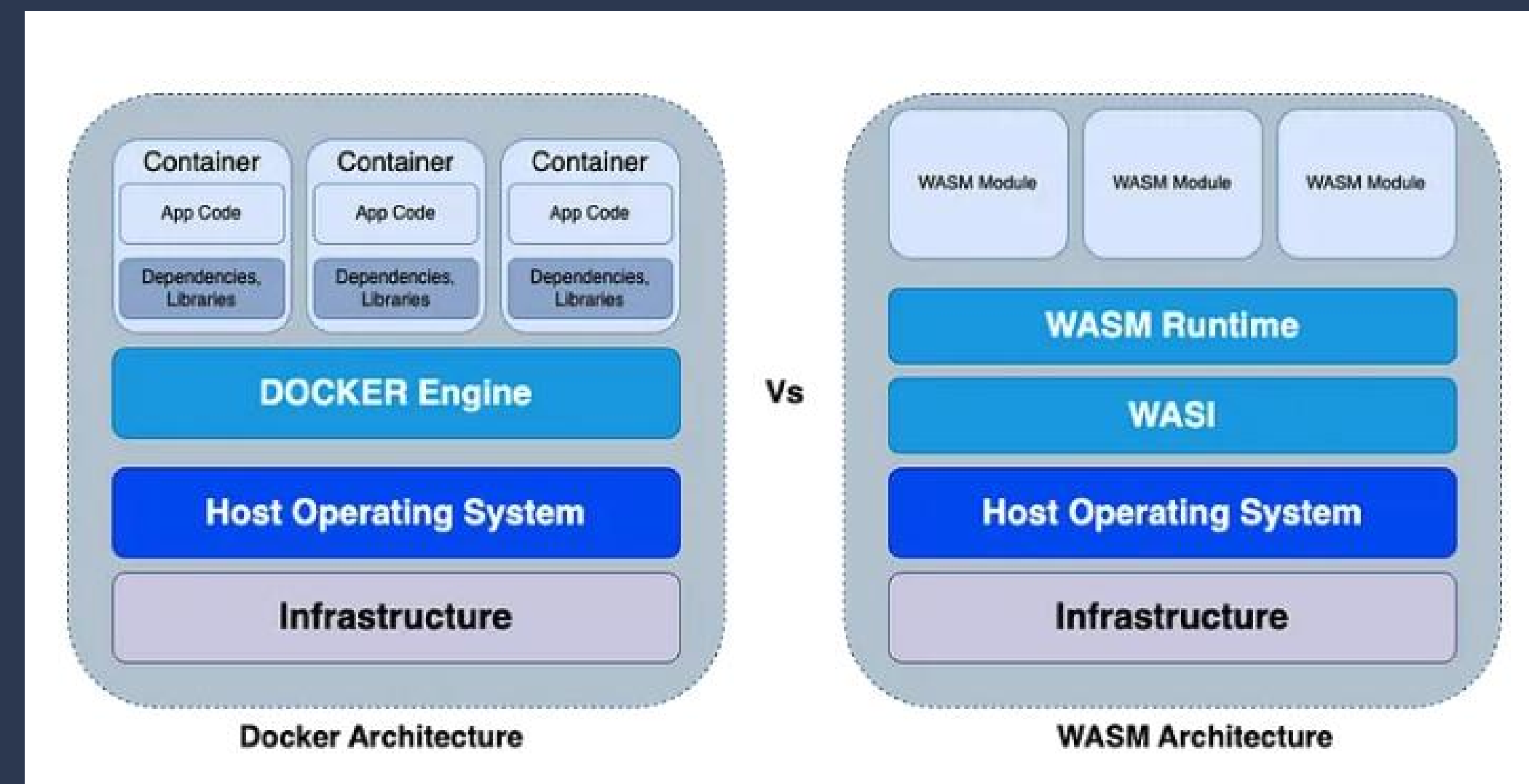
@PostgreSQL @planetpostgres @Docker



The screenshot shows a code editor with PostgreSQL client code in Rust/Wasm and a Docker container list. The container list shows a container named 'postgres' with a size of 0.8 MB, which is significantly smaller than the 50+ MB mentioned in the tweet.

# AGI最佳： Rust + WebAssembly – 3

- WebAssembly — 更快、更轻、更安全
  - 沙盒机制提供了更安全的生产环境
    - 保护用户数据、系统资源
    - 字节码验证防止恶意代码
    - Wasm模块间相互隔离的执行环境

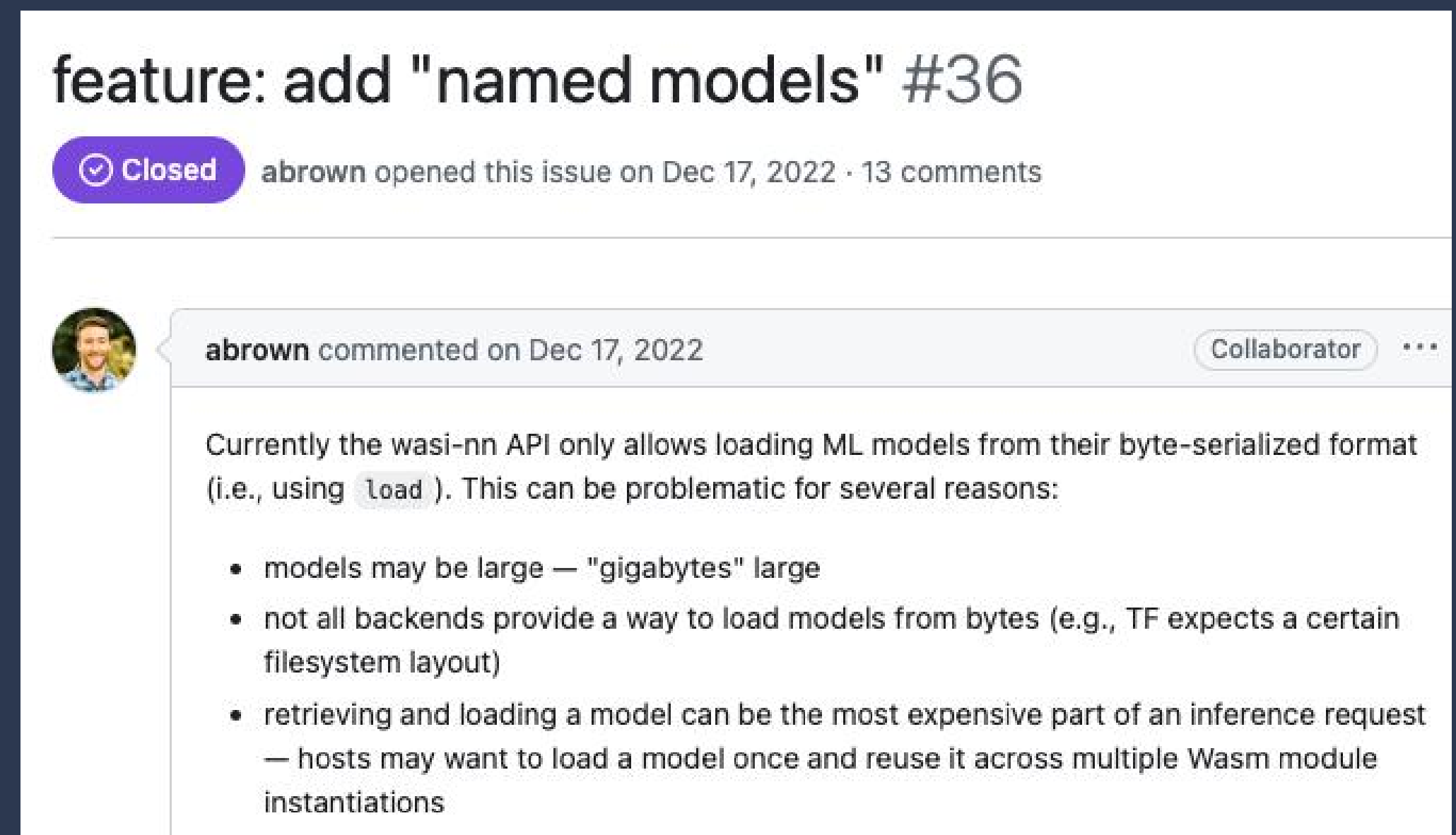


图片源自<https://medium.com/@shivraj.jadhav82/webassembly-wasm-docker-vs-wasm-275e317324a1>



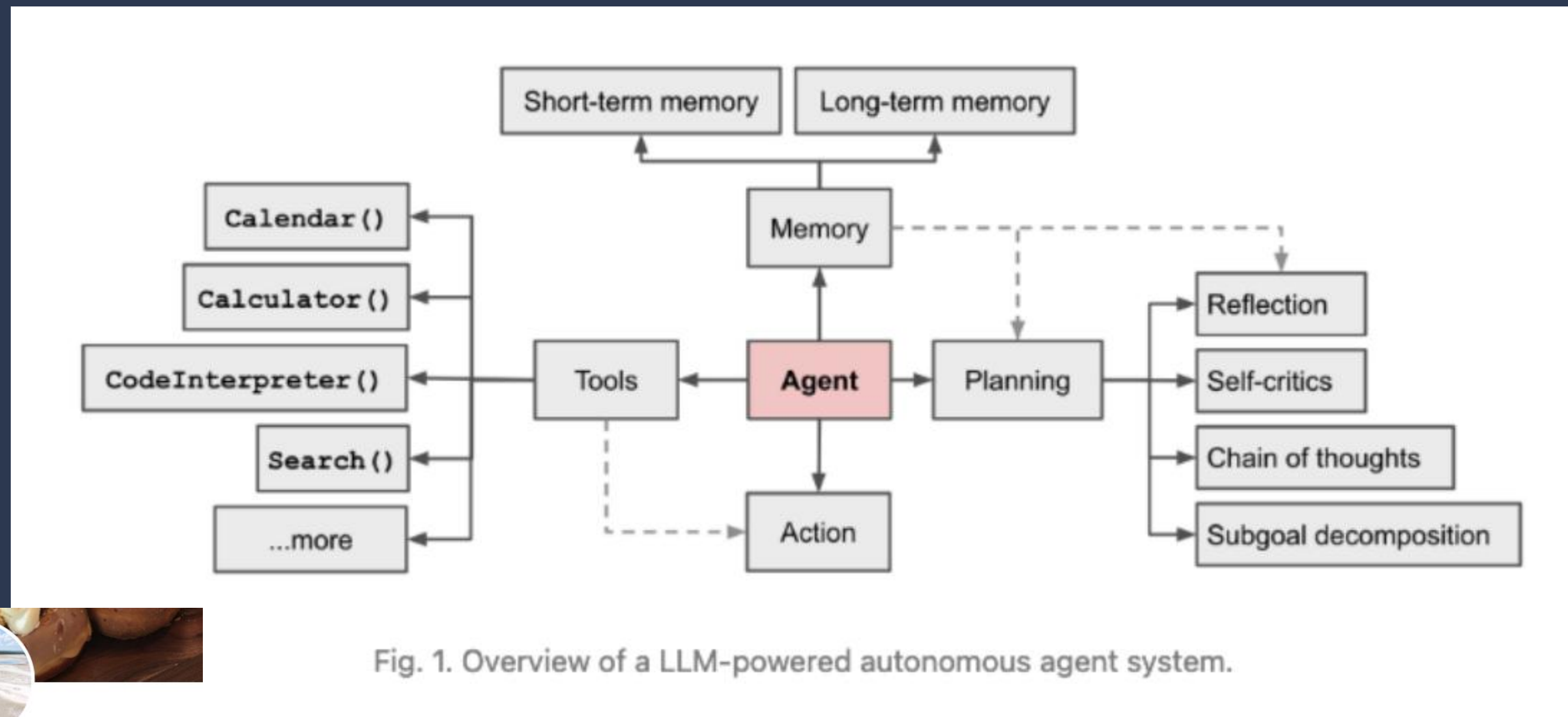
# AGI最佳: Rust + WebAssembly – 4

- WASI-NN 标准
  - 主流机器学习推理引擎
    - TensorFlow, PyTorch, OpenVINO, ...
- 针对大模型的功能扩展
  - Llama2.c, llama.cpp, ...



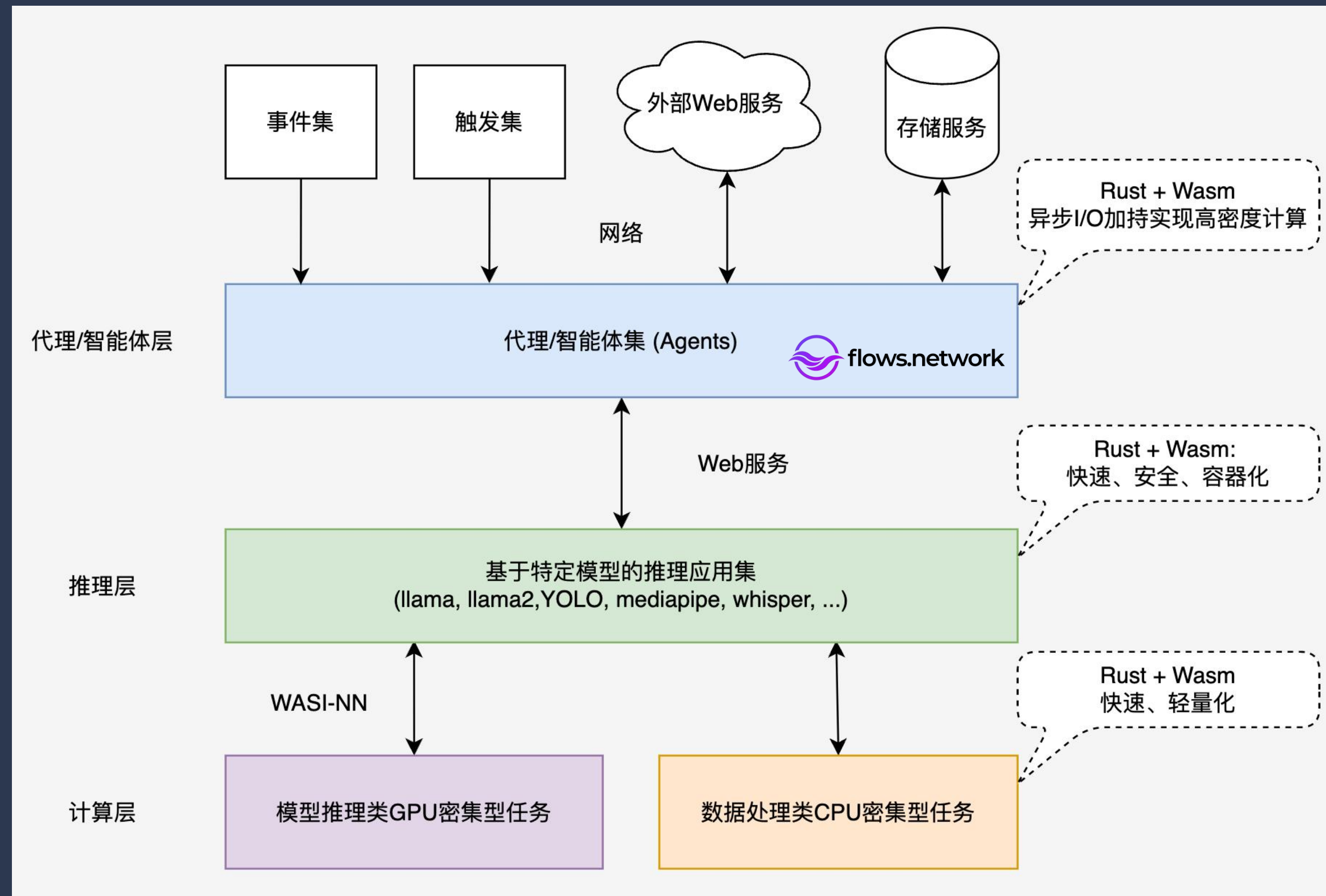
# 项目：基于Rust + Wasm构建Agent – 1

- Agent 概念模型





# 项目：基于Rust + Wasm构建Agent – 2





# 项目：基于Rust + Wasm构建Agent – 3

- Code Review Bot
  - 总结PR做了什么
  - 每一个commit做了哪些改动
  - 改动涉及的风险和潜在问题

The screenshot shows a GitHub pull request interface. At the top, a green 'Merged' badge is visible next to the title '[Rust] Optimize host function #2394'. Below this, a comment from user 'alabulei1' is shown, dated 'Apr 5'. The comment text reads: 'Hello, I am a [serverless review bot](#) on [flows.network](#). Here are my reviews of code commits in this PR.'

The review content is structured as follows:

- Overall Summary:**

This set of patches involves modifications primarily to the Rust bindings for the wasmedge-sys comp workflow, and the test files. The primary focus of these changes is to improve memory management, number of host functions, and update assertions in tests.
- Potential issues and errors:**
  1. Removing the check for host function capacity and the 'MAX\_HOST\_FUNC\_LENGTH' environmei excessive resource consumption, affecting system stability and performance.
  2. The `test_crazy_function_creation` function may consume a significant amount of system reso time to finish. Additionally, increasing the loop count from 10 to 1,000,000 in `function.rs` might lea concerns and resource exhaustion.
  3. Changing the assertions in tests to use `assert!` instead of `assert_eq!` (and vice versa) could li outcomes and potential false test failures.
  4. Mutexes and RwLocks in the code can potentially cause deadlocks if not implemented carefully.
  5. Updating the Rust CI workflow to restrict the number of test threads to 1 across all platforms cou process.
- Important Findings:**
  1. Memory management has been improved through the addition of support for removing real funct creation of a footprint for host functions.
  2. Changes have been made to the Rust bindings CI workflow, including updating the operating sysi compiler versions matrix, as well as modifying the cargo test commands to set the test-threads t
  3. The various changes to the test files involve updating assertions and modifying test scenarios, af that are being checked and the performance of the tests.

Given these changes, it's important to ensure that potential issues such as deadlocks, resource exhai failures are mitigated. Also, be cautious when removing limitations on host functions, as it could impa stability and performance.

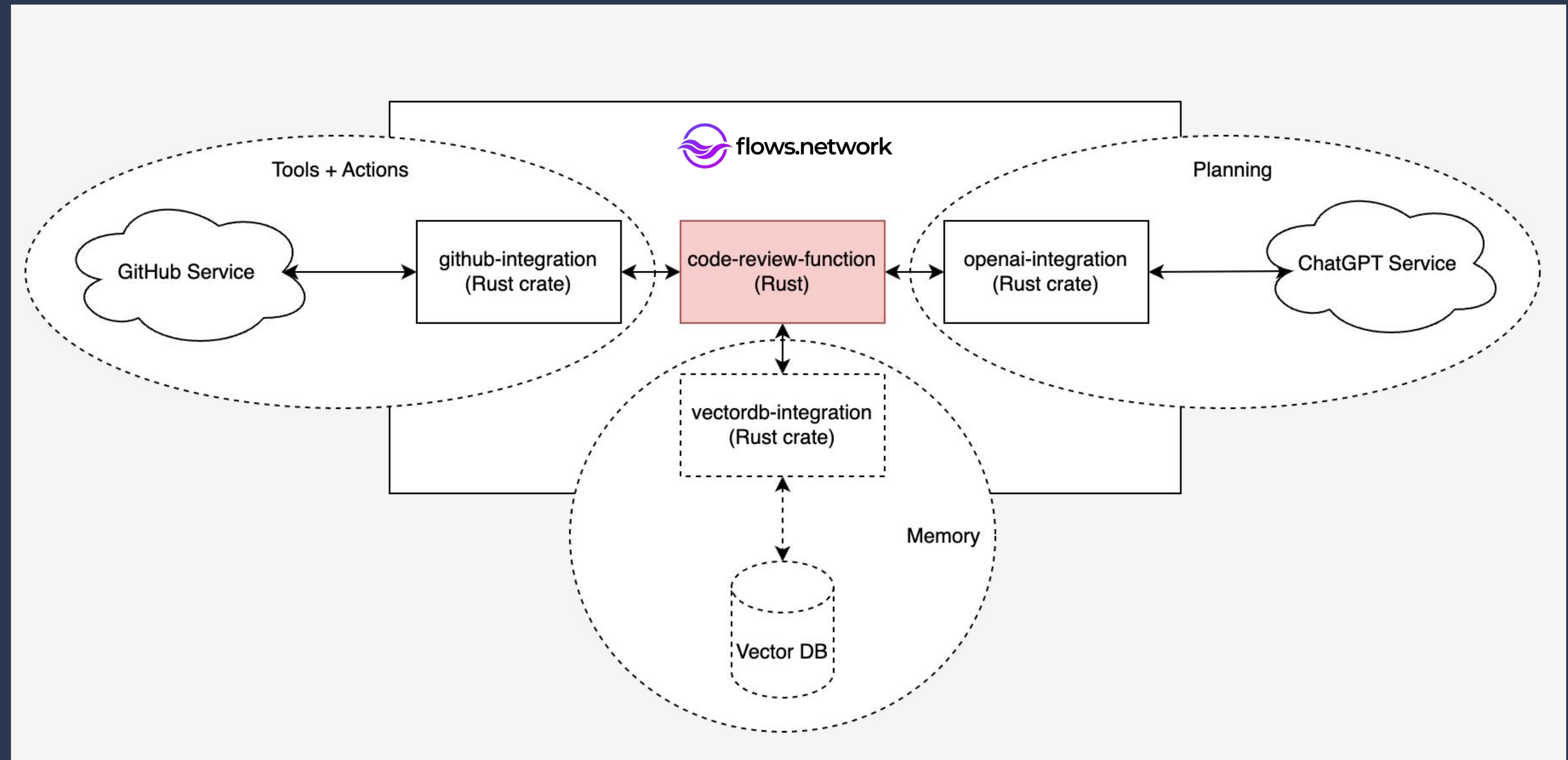
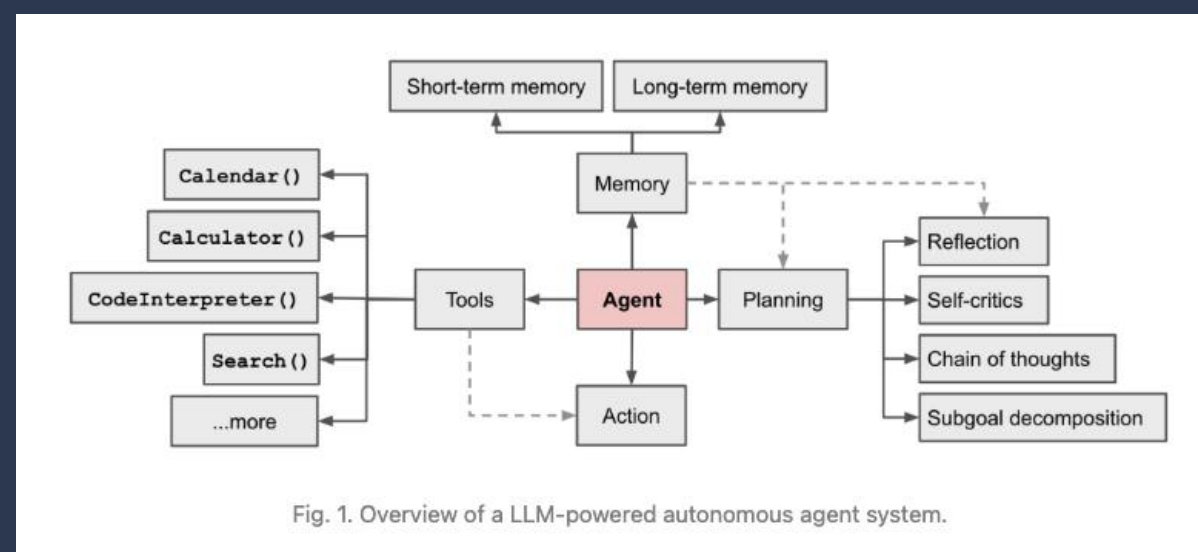
Below the main review, there is another comment from user 'juntao' dated 'last month', which also follows the same bot template. It reviews a pull request titled '[WIP] plugin opencvmini #2403' and provides detailed observations on a C++ code snippet for `WasmEdgeOpenCvMini` class, including points about license identifiers, header files, template classes, and namespace blocks.

《那些让 ChatGPT review 代码的程序员，后来都怎么样了？》



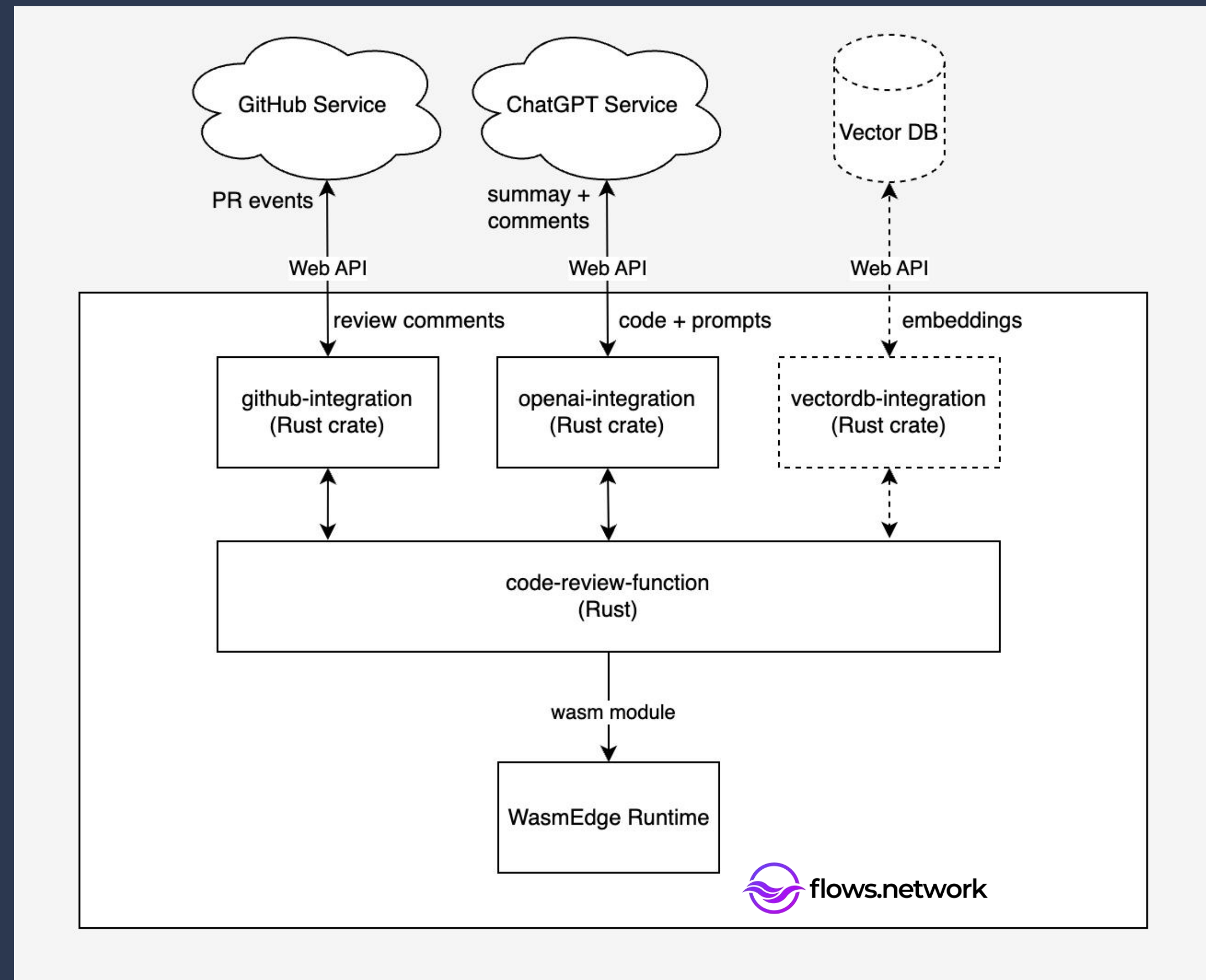
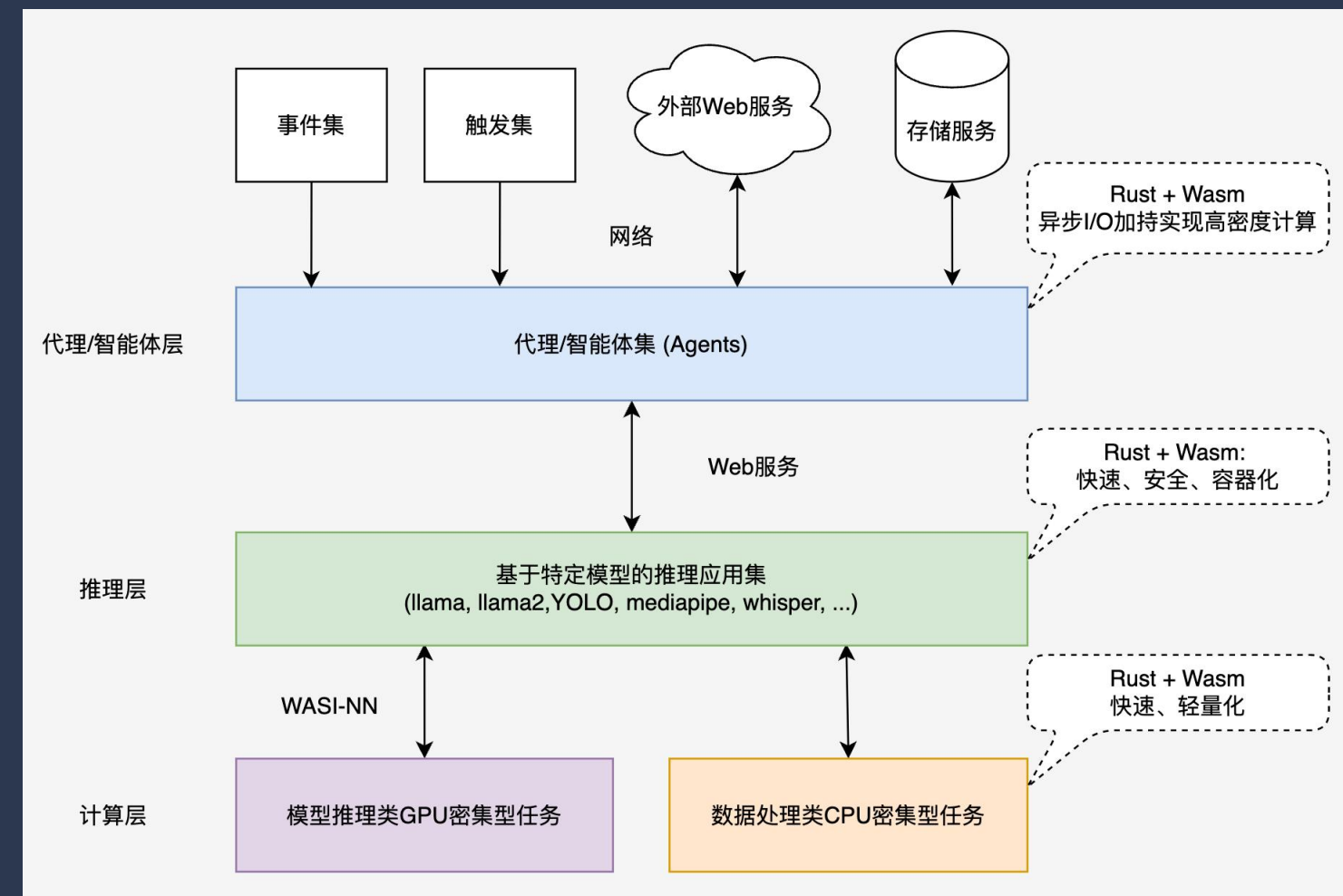
# 项目：基于Rust + Wasm构建Agent – 4

- Code Review Bot



# 项目：基于Rust + Wasm构建Agent – 5

- Code Review Bot





# THANKS

—  
软件正在重新定义世界

Software Is Redefining The World