GPU-Accelerated AI Inference for Local LLM Development with Docker Model Runner

April 5, 2025

About Me

- DevRel at Docker
- Former Docker Captain
- Docker Community Leader
- Distinguished Arm Ambassador
- Worked at Dell EMC, VMware, Redis

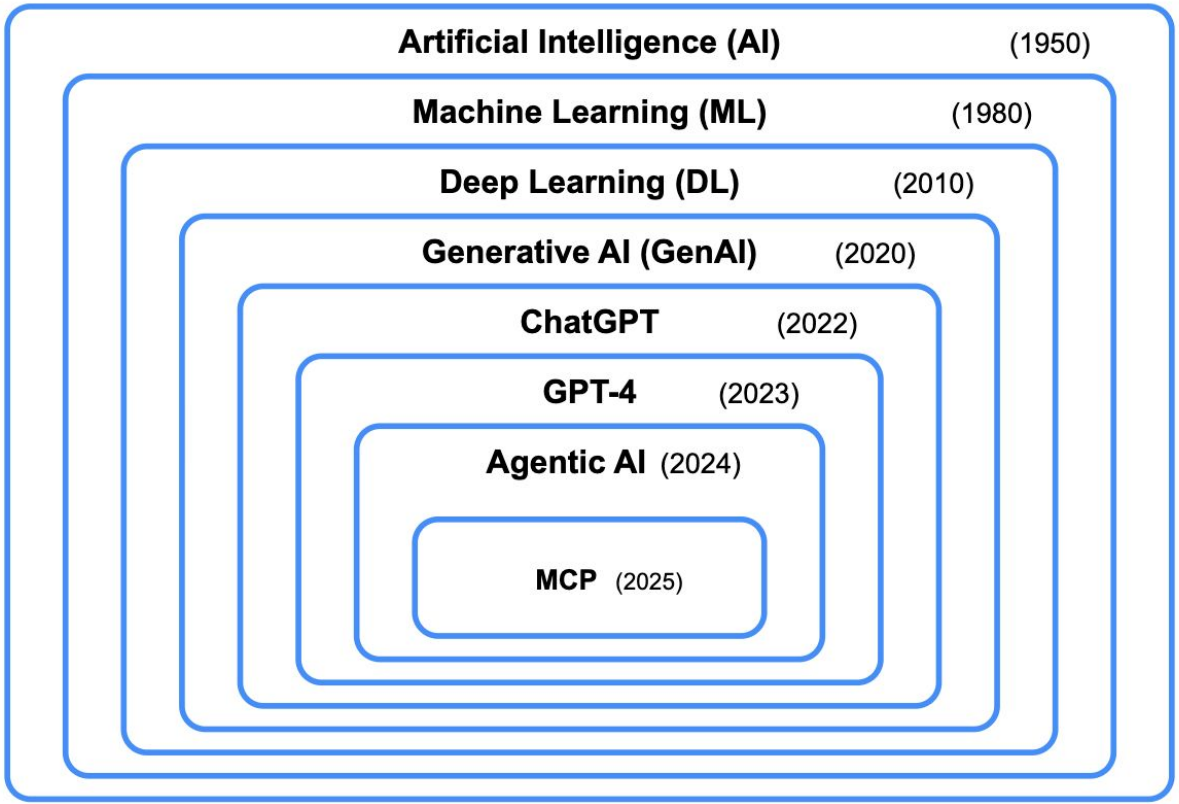
@ajeetsraina

Today’s agenda

01. Why GenAI?
02. Challenges with the traditional GenAI App
03. How Docker views GenAI
04. Introducing Docker Model Runner
05. Local Development Workflow for GenAI App
06. Testing Strategies for GenAI Applications
07. Q&A

Why GenAI?

The AI Landscape

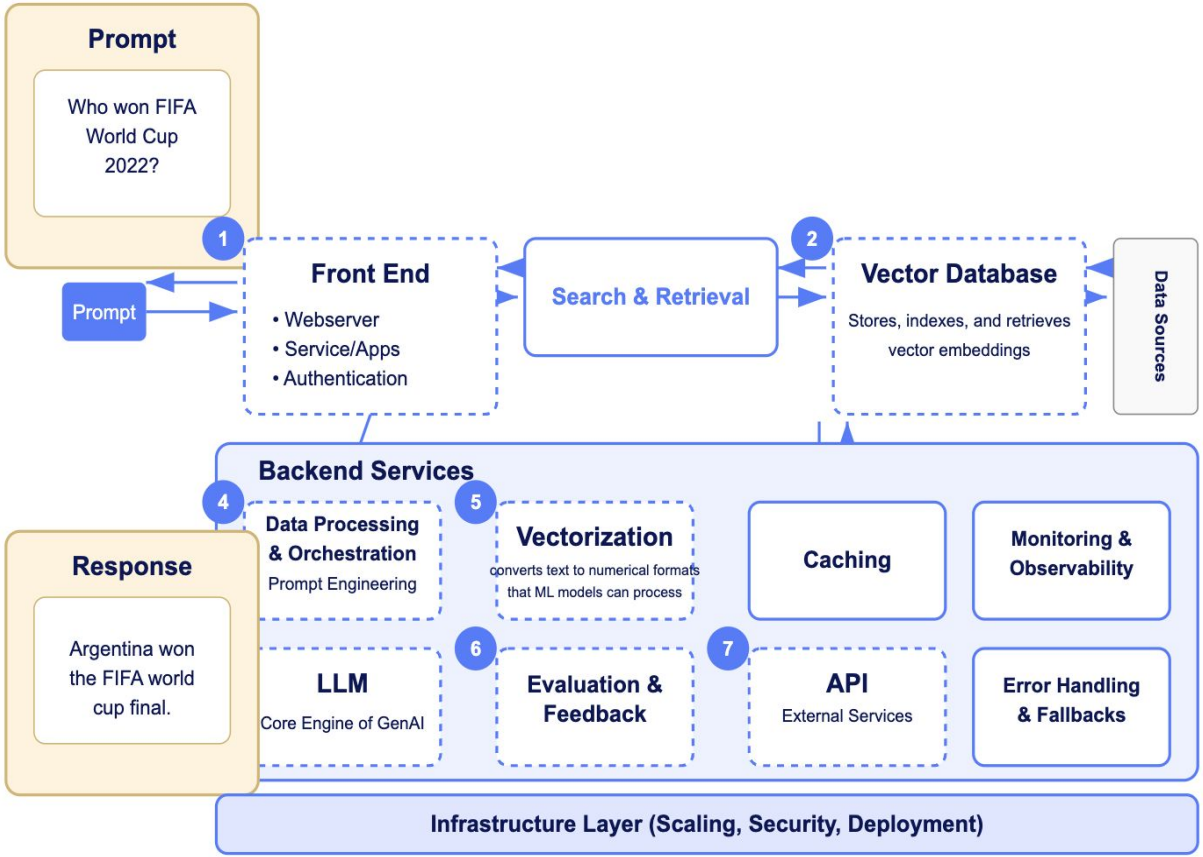


We've been chasing bigger, more powerful models for years. But what if "smaller" is actually the smarter path forward?

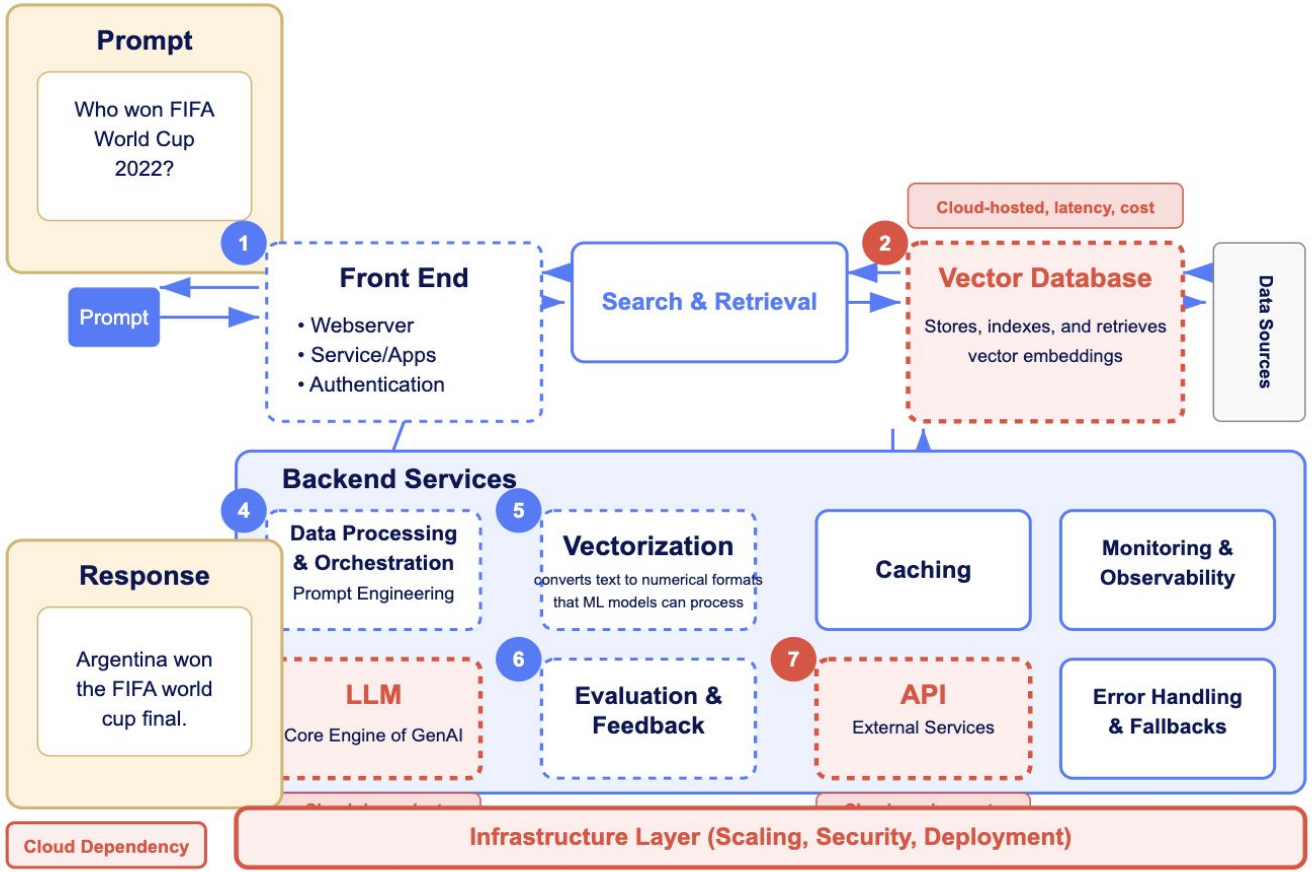
Honestly, I'm wondering if we've been thinking about AI all wrong

Small Language Models (SLMs) are projected to become a $5.45 billion market by 2032

A Typical GenAI Architecture

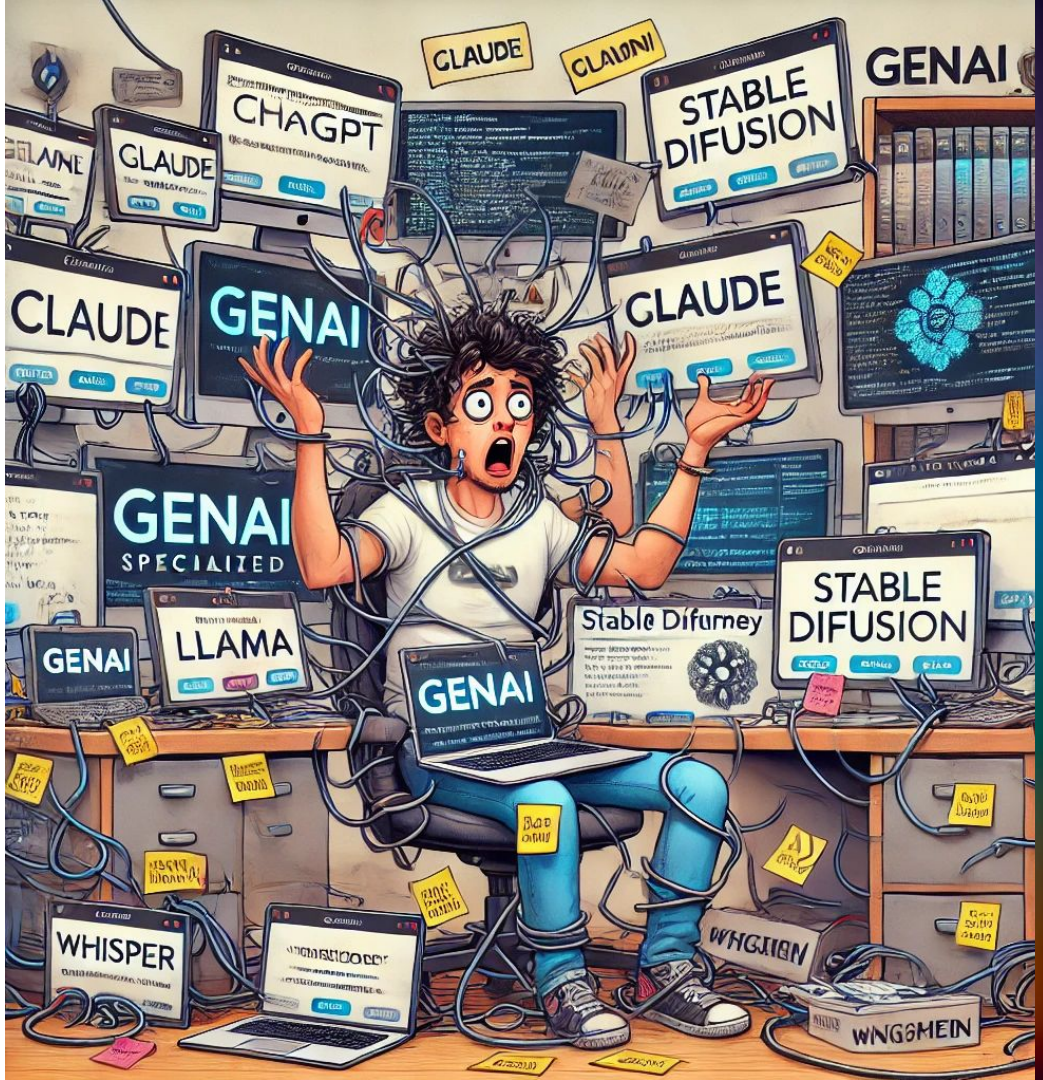


A Typical GenAI Architecture



Pain Points..

- “How do I keep my data private during Development?
- “These APIs costs are killing my budget”
- “Setting up local inference is too complex”
- “My Laptop can’t handle these models”
- “I need to switch models frequently during development”

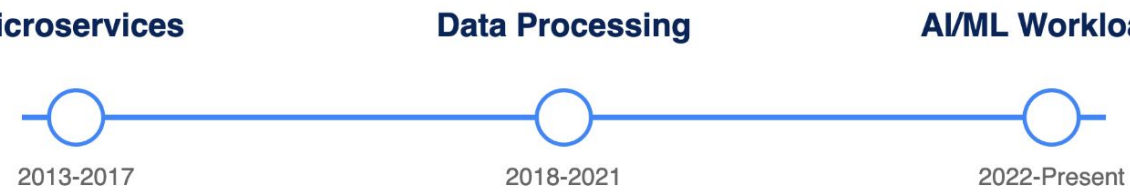


Docker's vision of GenAI

GenAI is just another container workload in your existing SDLC

Containers are the ideal solution for AI/ML

Consistency • Portability • Efficiency • No stack refactoring needed



88%

of AI/ML developers
use containers in their workflow

133%

increase in pull demand for
AI content on Docker Hub

Microservices

- Cloud-Native Apps
- CI/CD Pipelines
- Serverless

Data Processing

- Big Data Analytics
- Batch Processing
- ETL Workflows

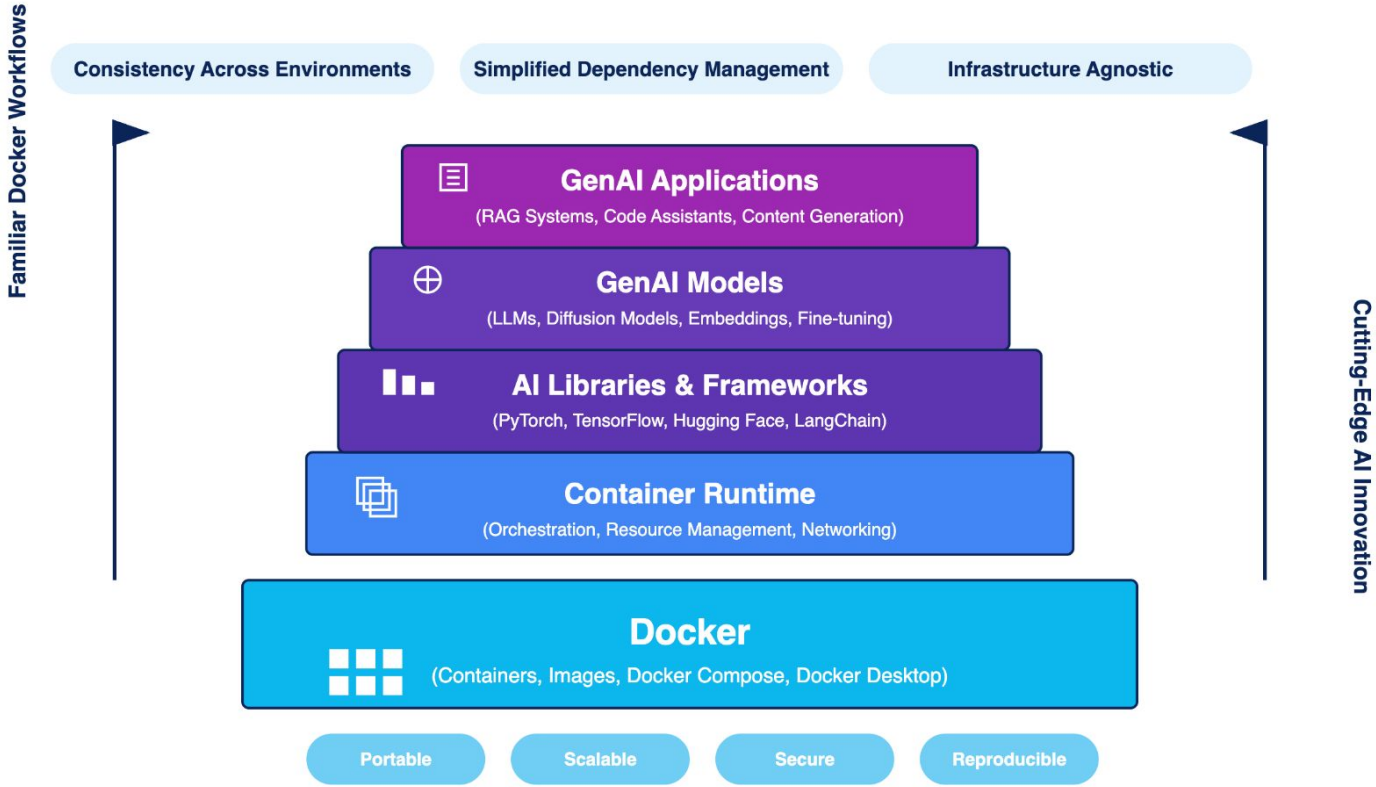
AI/ML Workloads

- Model Training
- Inference Engines
- MLOps

Source: Docker's 2025 State of Application Development Report

Docker: The Foundation for GenAI Development

Use the same underlying processes and tools, but with a new stack



So what's the story with containers and GPUs?

How GPU-Less container stories look like?

No hardware acceleration for model inference

Slower training and fine-tuning operations

Potential performance bottlenecks for larger models

How Docker is trying to solve this problem?

Introducing Docker Model Runner

Docker Model Runner

A GPU-accelerated LLM runner that enables developers to run AI models locally

Enhance development velocity

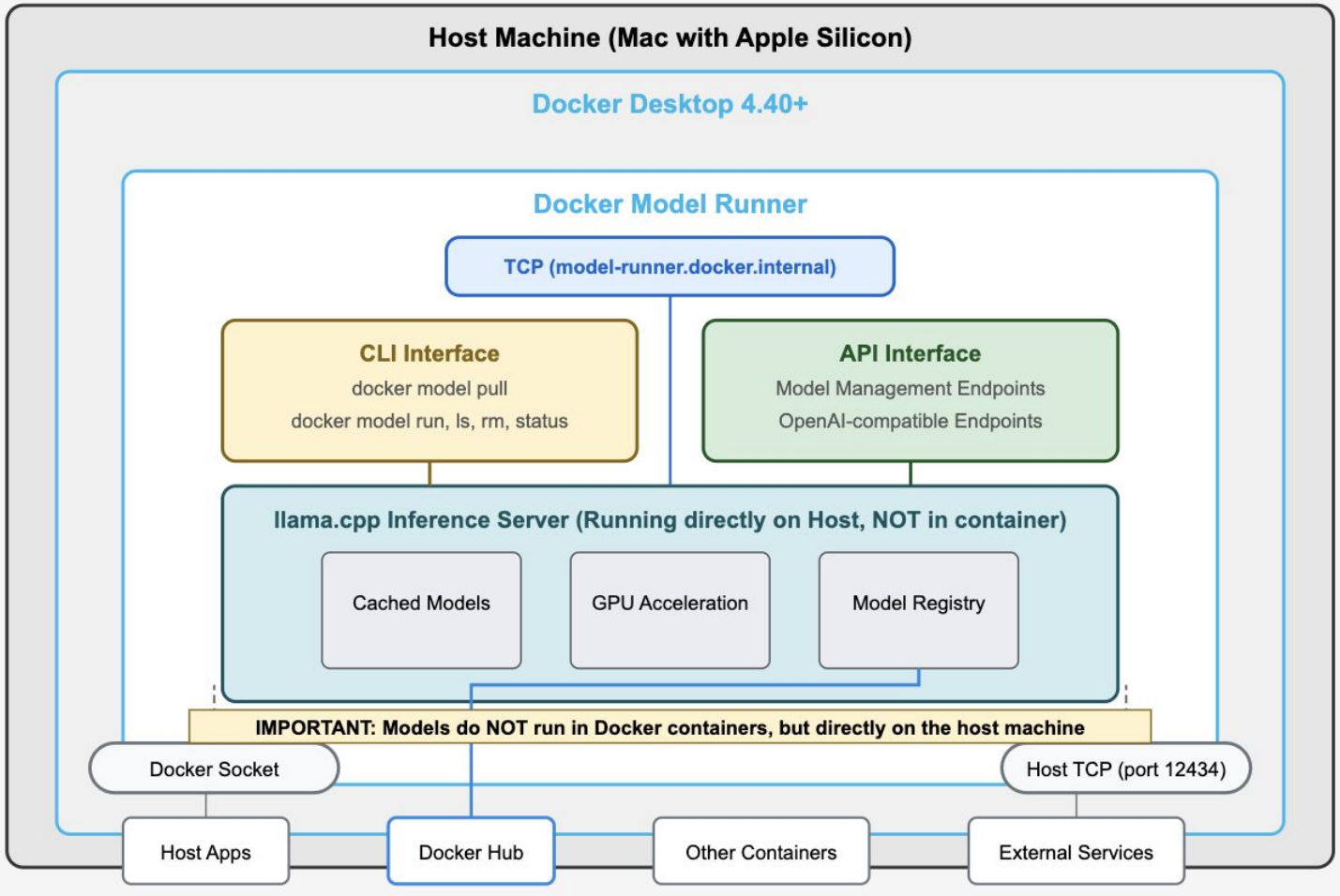
- ➔ Docker-built LLM runner optimized for Apple Silicon & GPU acceleration
- ➔ Seamless UX, making it easier to run AI models locally
- ➔ Inference Engine in Docker Desktop

Run AI Models Locally with Ease: No complex setup. Just pull a model and run it.

Scalable: Designed to integrate with containerized workflows, supporting both local development and cloud inference.

Optimized for Apple Silicon

Docker Model Runner Architecture



Windows support with NVIDIA GPU is coming soon.
(April 2025).

How is Model Runner different from Ollama?

Ollama Vs Model Runner

Both enable local LLM Inference

1. Native Docker Integration

- Unlike Ollama, which operates as a standalone tool, Model Runner is fully integrated into the Docker ecosystem
- The docker model CLI treats AI models as first-class citizens alongside containers, images, and volumes.
- Docker users can manage their models using familiar commands and patterns, with no need to learn a separate toolset or workflow.

2. Enterprise Needs

- Registry Integration: Works seamlessly with private corporate registries that support OCI artifacts
- Standardized Model Distribution: Teams can publish and share models using the same infrastructure as their container images
- Optimized Storage: Better handling of large, uncompressible model weights without unnecessary layer compression

Ollama Vs Model Runner

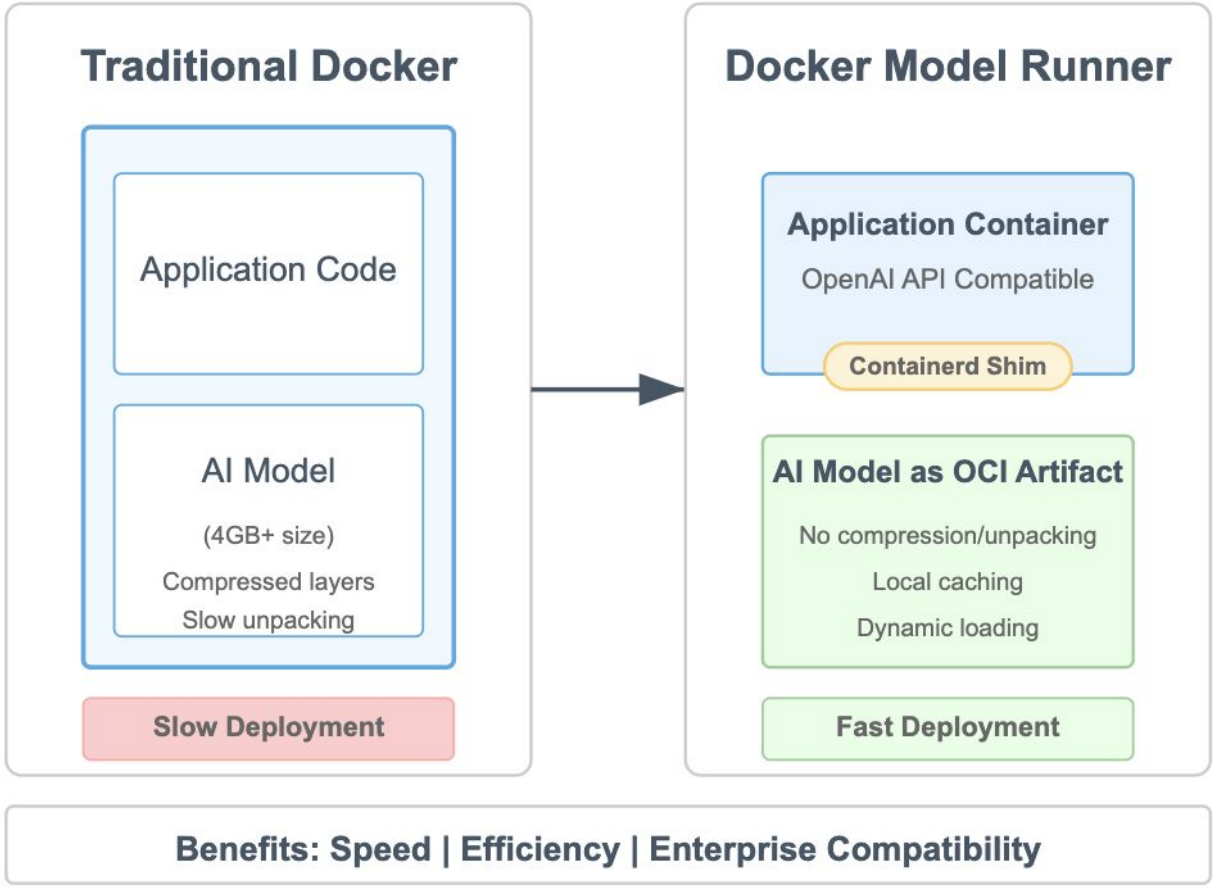
Both enable local LLM Inference

3. Production-ready architecture

Model Runner is designed with production workflows in mind from the ground up:

- **OCI Artifact Storage**: Models are stored as standardized OCI artifacts in Docker Hub or other compatible registries, enabling proper versioning and distribution through existing CI/CD pipelines.
- **Multiple Connection Methods**: Model Runner offers flexible integration options via Docker socket, DNS resolution, or TCP connections.
- **Better Resource Management:** By running as a host-level process with direct GPU access, Model Runner achieves better performance optimization than containerized solutions.

Docker Model Runner Architecture



How Model Runner works?

With Docker Model Runner, the AI model DOES NOT run in a container.

Instead, the runner uses a host-installed inference server (llama.cpp for now) that runs natively on your Mac rather than containerizing the model.

How Model Runner works?

Host-Level Installation:

- Docker Desktop installs `llama.cpp` directly on your host machine (not in a container)
- This allows direct access to the hardware GPU acceleration on Apple Silicon.

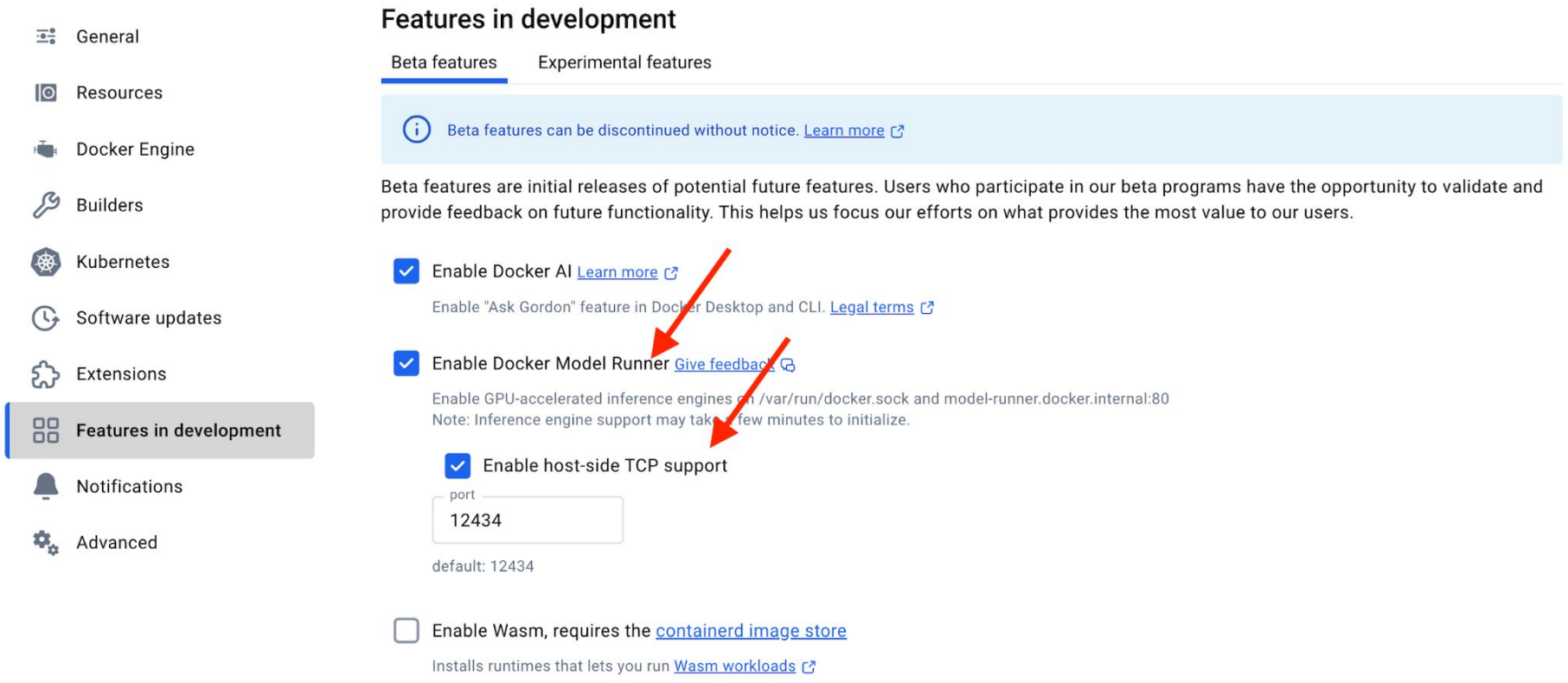
GPU Acceleration:

- By running directly on the host, the inference server can access Apple's Metal API
- This provides direct GPU acceleration without the overhead of containerization
- You can see the GPU usage in Activity Monitor when queries are being processed

Model Loading Process:

- When you run `docker model pull`, the model files are downloaded from Docker Hub
- These models are cached locally on the host machine's storage.
- Models are dynamically loaded into memory by `llama.cpp` when needed.

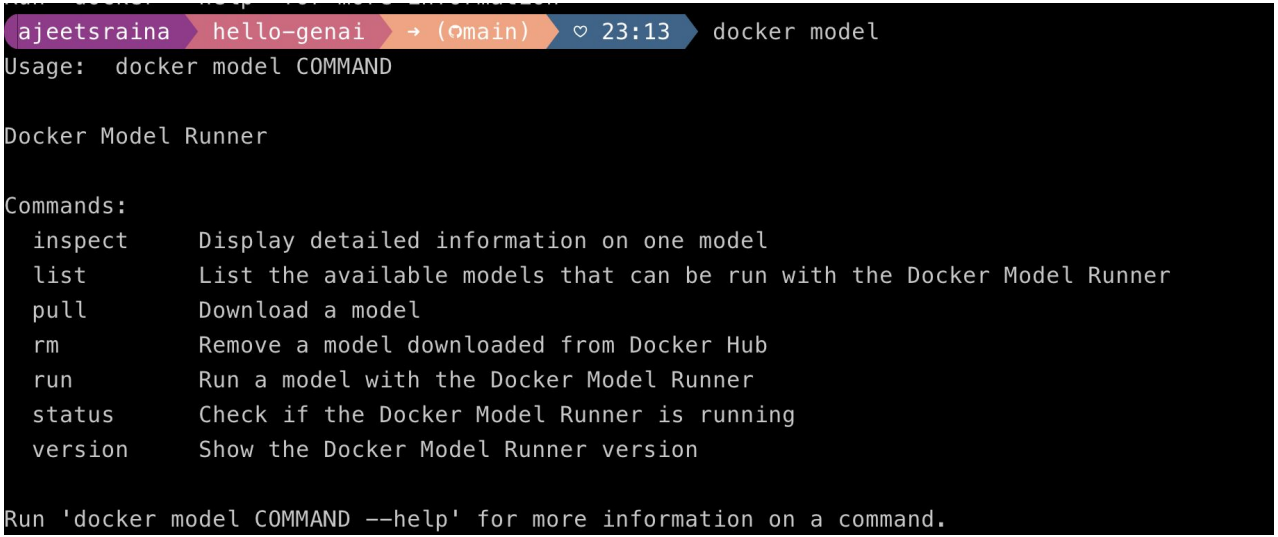
Enabling Model Runner



$ docker model

By using this new CLI, developers can:

- Pull models from registries (e,g Docker Hub)
- Run models locally with GPU acceleration
- Integrate models into their development workflows
- Test GenAI applications during development without relying on external APIs

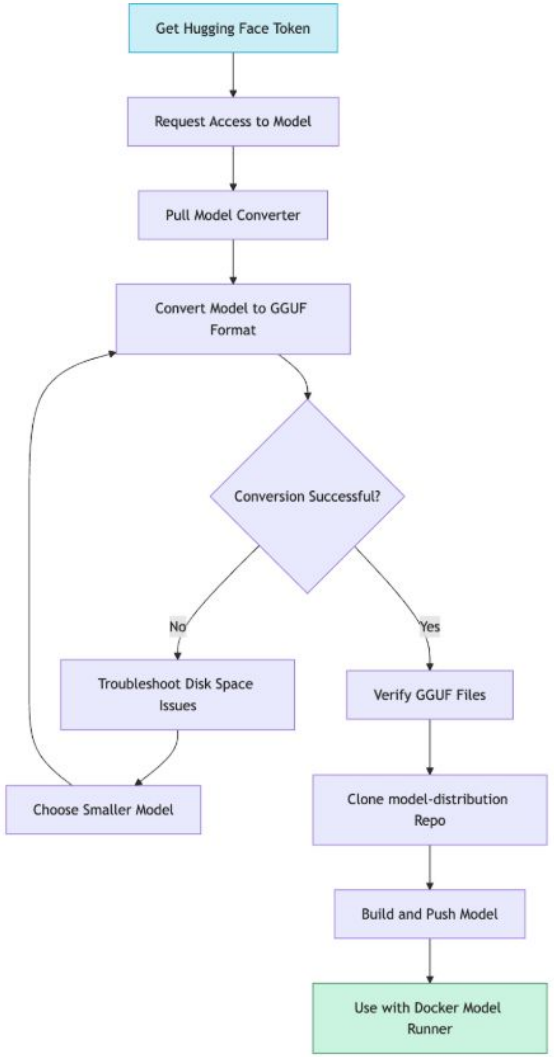


Pulling the Model from registries

How to build your custom model?

$ docker model build ??

- Not yet available



Managing Docker Models

Managing Models (using CLI)

Docker Model Runner provides both CLI and API interfaces for managing models:

```
# Docker Model management
POST /models/create              # Pull a model
GET /models                      # List available models
GET /models/{namespace}/{name}   # Get model details
DELETE /models/{namespace}/{name} # Remove a model
```

1. From Within Containers

Containers can access the Model Runner via the internal DNS name:

`http://model-runner.docker.internal/`

Managing Models

2. From Host via Unix Socket

Access via the Docker socket:

```
curl --unix-socket $HOME/.docker/run/docker.sock \
    localhost/exp/vDD4.40/engines/llama.cpp/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
```

3. From Host via TCP

When TCP host support is enabled, you can either:

- Use the specified port directly (default 12434)
- Use a helper container as a reverse-proxy:

```
docker run -d --name model-runner-proxy -p 8080:80 \
  alpine/socat tcp-listen:80,fork,reuseaddr tcp:model-runner.docker.internal:80
```

Managing Models (using API Endpoints)

OpenAI API Compatibility

The Model Runner implements OpenAI-compatible endpoints:

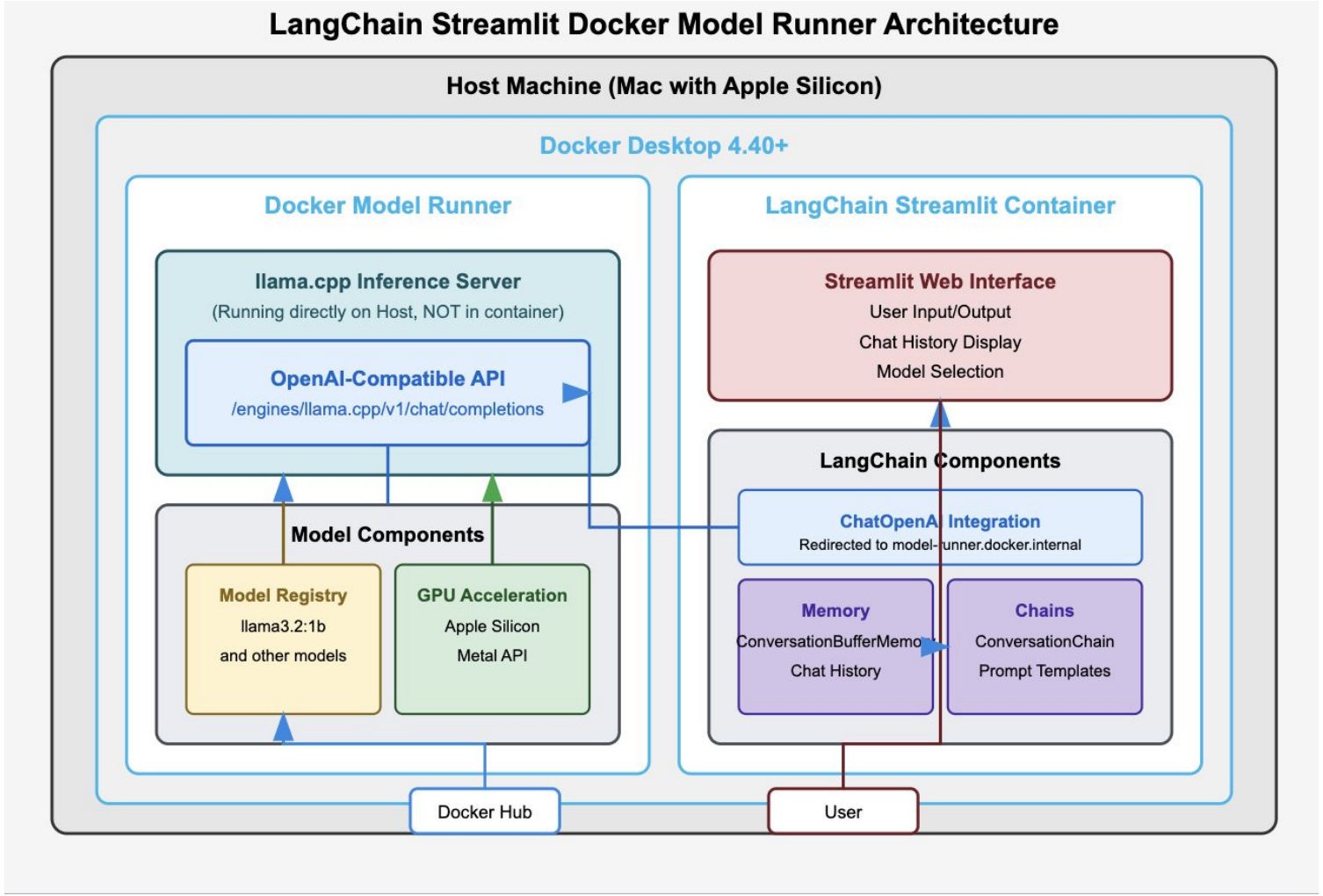
OpenAI-compatible endpoints

```
GET /engines/{backend}/v1/models
GET /engines/{backend}/v1/models/{namespace}/{name}
POST /engines/{backend}/v1/chat/completions
POST /engines/{backend}/v1/completions
POST /engines/{backend}/v1/embeddings
```

You can specify which model to use in the request payload, and the Model Runner will automatically load it if available.

Demo time!

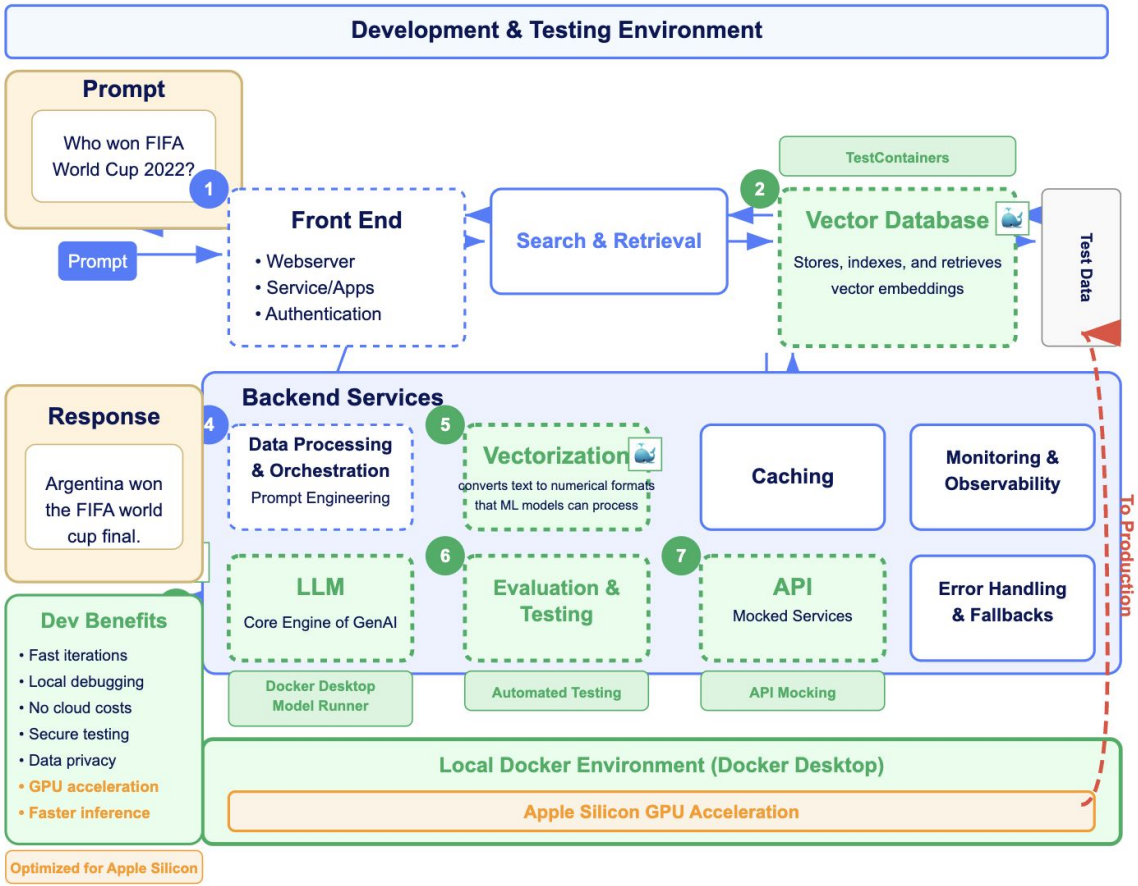
- Enabling the Model Runner
- Pulling and running the model locally

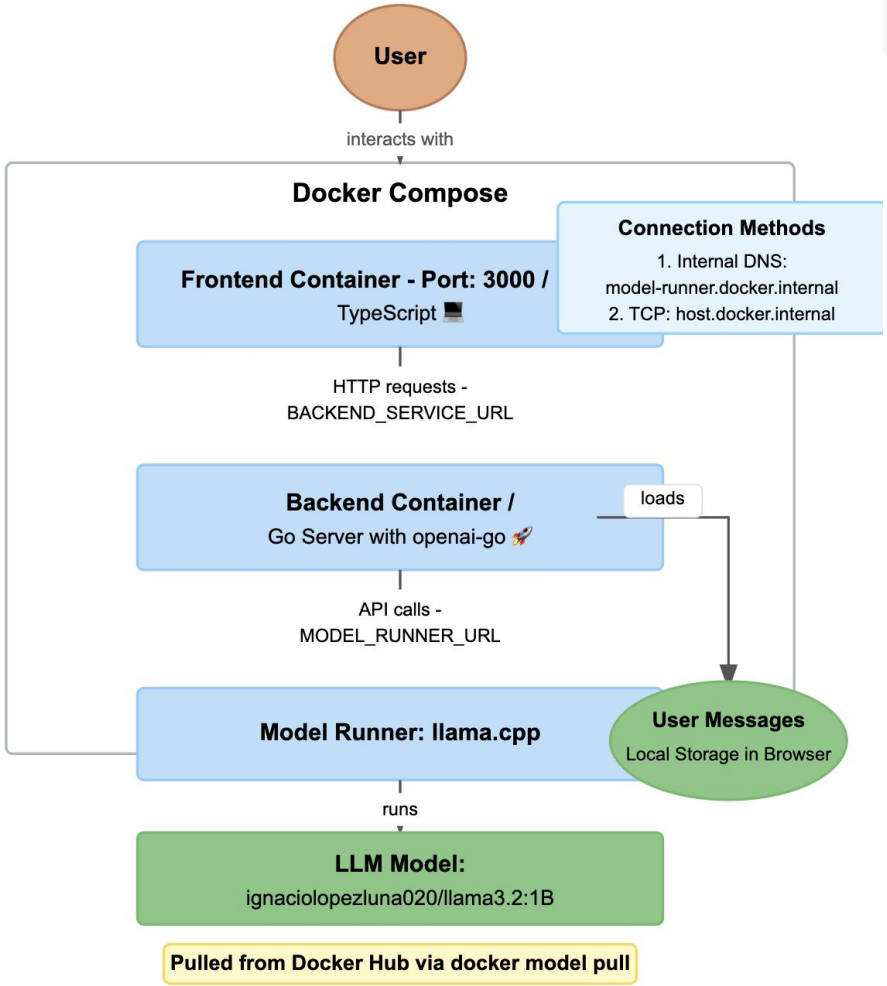

LangChain Streamlit Docker Model Runner Architecture
Host Machine (Mac with Apple Silicon)
Docker Desktop 4.40+
Docker Model Runner
llama.cpp Inference Server
(Running directly on Host, NOT in container)
OpenAI-Compatible API
/engines/llama.cpp/v1/chat/completions
Model Components
Model Registry
llama3.2:1b
and other models
GPU Acceleration
Apple Silicon
Metal API
LangChain Streamlit Container
Streamlit Web Interface
User Input/Output
Chat History Display
Model Selection
LangChain Components
ChatOpenAI Integration
Redirected to model-runner.docker.internal
Memory
ConversationBufferMemo
Chat History
Chains
ConversationChain
Prompt Templates
Docker Hub
User

Local Development Workflow for GenAI Applications

A Typical GenAI Architecture

Using Model Runner with GPU Acceleration

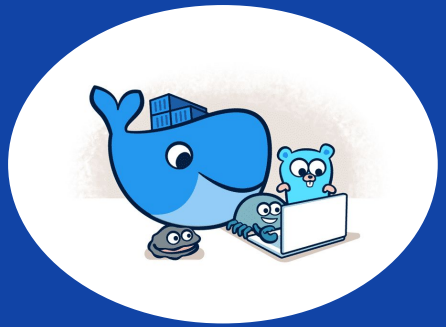



User
interacts with
Docker Compose
Connection Methods
1. Internal DNS:
model-runner.docker.internal
2. TCP: host.docker.internal
Frontend Container - Port: 3000 /
TypeScript
HTTP requests -
BACKEND_SERVICE_URL
Backend Container /
Go Server with openai-go
loads
API calls -
MODEL_RUNNER_URL
Model Runner: llama.cpp
User Messages
Local Storage in Browser
runs
LLM Model:
ignaciolopezluna020/llama3.2:1B
Pulled from Docker Hub via docker model pull

Demo time!

- Building a Chatbot application
- Using Model Runner (using Docker socket)
- Using Model Runner (using TCP support)

Wrapping up



References

- https://docs.docker.com/desktop/features/model-runner/
- https://hub.docker.com/u/ai

Thank you!