

# AI Masterclass 2025

May 12-16, Trinity College Dublin

# Welcome!

# This is me

Diego Maniloff



dmanilof@redhat.com



Hugging Face



# Not a typo

Diego Maniloff



dmanilof@redhat.com



Hugging Face



# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 1
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 1
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 1
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 2
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 2
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 2
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 3
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7     input_shields=["guardrails::my-filter"],
8     output_shields=["guardrails::my-filter"],
9 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 3
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7     input_shields=["guardrails::my-filter"],
8     output_shields=["guardrails::my-filter"],
9 )
```

# Masterclass Outline

From scratch to an advanced RAG agent with guardrails

```
1 # Unit 3
2 agent = Agent(
3     llama_stack_client,
4     model="my-model",
5     instructions="You are a helpful assistant that can use tools to answer questions.",
6     tools=["basic-rag-tool", "advanced-rag-tool"],
7     input_shields=["guardrails::my-filter"],
8     output_shields=["guardrails::my-filter"],
9 )
```

# Unit 1

# Introduction to Llama Stack

# What we're doing today

1. Why Llama Stack?
2. Environment Setup
3. Our Own (tiny) Stack
4. Core Concepts
5. Let's use our stack
6. Let's break our stack
7. Takeaways
8. Implement RAG from scratch
9. Agents
10. Agents on Llama Stack
11. Execution model
12. An agent that can find its location
13. An agent that can search the web
14. An agent that can read files

## Format

1. , then
2. , then
3. , then

# Unit 1 Goals

- Understand the core concepts of Llama Stack
- Get our development environment working
- Prepare us for Units 2 and 3

# Don't worry, we'll take breaks!

- 8:30 - 9am: Breakfast
- 10am: Morning coffee break
- 12-1pm: Lunch
- 2:30pm: Afternoon coffee break

What is Llama Stack and why is it a good idea?

# AI System Design



# AI System Design

*I just need model inference, curl*

*<https://api.openai.com/v1/chat/completions>*

*and I'm done*



# AI System Design

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...
- **We need agents**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...
- **We need agents**
- **We need tool calling**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...
- **We need agents**
- **We need tool calling**
- **We need monitoring**

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...
- **We need agents**
- **We need tool calling**
- **We need monitoring**
- ...

# AI System Design

- I just need model inference, `curl https://api.openai.com/v1/chat/completions` and I'm done
- That's not very useful. Let's connect the model to our data. **We need retrieval.**
- Hmm, but exactly how useful is this? **We need evaluation and metrics.**
- But wait! Will it leak private information? **We need safety guardrails and content filtering.**
- ...
- **We need agents**
- **We need tool calling**
- **We need monitoring**
- ...
- How do we deploy this whole thing?

What is Llama Stack and why is it a good idea?

What is Llama Stack and why is it a good idea?

# Opinionated Organization

What is Llama Stack and why is it a good idea?

# Helpful Organization

What is Llama Stack and why is it a good idea?

# Organization

# Organization

What's there to organize?

- Inference
- Retrieval
- Evaluation
- Safety
- Agents
- Tool calling
- Monitoring
- Deployment
- ... and more!





# Dev Environment

# Environment Setup

Setting up the local development environment

Run the ollama server

```
ollama pull llama3.2:3b-instruct-fp16  
ollama serve
```

Clone the masterclass repo

```
git clone https://github.com/trustyaix-explainability/ai-masterclass-2025.git  
cd ai-masterclass-2025
```

Build and run the stack

```
uv run --with "llama-stack" llama stack build --config=distributions/masterclass-minimal/build.yaml  
  
source masterclass_minimal/bin/activate  
llama stack run distributions/masterclass-minimal/run.yaml
```



# Environment Setup

What happened?

```
$ tree distributions
distributions
├── masterclass-agents
│   ├── build.yaml
│   └── run.yaml
└── masterclass-minimal
    ├── build.yaml
    └── run.yaml
```

Distribution config files (from our repo)

```
$ ls -l masterclass_minimal
total 16
-rw-r--r--  1 diego  staff   43B May  9 09:27 CACHEDIR.TAG
drwxr-xr-x  39 diego  staff  1.2K May  9 09:27 bin
drwxr-xr-x   3 diego  staff   96B May  9 09:27 lib
-rw-r--r--  1 diego  staff  176B May  9 09:27 pyvenv.cfg
```

Python virtual environment (generated by the build command)

```
$ tree ~/.llama/distributions/masterclass_minimal
/Users/diego/.llama/distributions/masterclass_minimal
├── kvstore.db
└── masterclass_minimal-build.yaml
```

Distribution directory (generated by the build command)

# Our Own (tiny) Stack

# Our Own Stack

🙌 We just ran our first stack!

```
cat distributions/masterclass-minimal/run.yaml
```

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/dist}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

# Core Concepts

- Api
- Provider
- Resource
- Distributions

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/dist}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

- Api
- Provider
- Resource
- Distributions

An API is a collection of endpoints.

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/dist}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

- Api
- Provider
- Resource
- Distributions

An API is a collection of endpoints.

Providers implement APIs ...

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/dist}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

- Api
- Provider
- Resource
- Distributions

An API is a collection of endpoints.

Providers implement APIs ...

... and provide resources.

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/dist}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

- Api
- Provider
- Resource
- Distributions

An API is a collection of endpoints.

Providers implement APIs ...

... and provide resources.

A distribution is a collection of specific providers (+ config).

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
providers:
  inference:
    - provider_id: ollama
      provider_type: remote::ollama
      config:
        url: ${env.OLLAMA_URL:http://localhost:11434}
  telemetry:
    - provider_id: meta-reference
      provider_type: inline::meta-reference
      config:
        service_name: ${env.OTEL_SERVICE_NAME:}
      sinks: ${env.TELEMETRY_SINKS:console,sqlite}
      sqlite_db_path: ${env.SQLITE_STORE_DIR:~/llama/distro}
models:
  - metadata: {}
    model_id: meta-llama/Llama-3.2-3B-Instruct
    provider_id: ollama
    model_type: llm
    provider_model_id: llama3.2:3b-instruct-fp16 # from ollama
server:
  host: localhost
  port: 8321
```

# Core Concepts

APIs map to resources

Inference, Eval and Post Training	Model
Safety	Shield
Tool Runtime	ToolGroup
DatasetIO	Dataset
VectorIO	VectorDB
Scoring	ScoringFunction
Eval	Model and Benchmark



# Let's use our stack

# Let's use our stack

From the CLI

```
llama-stack-client --help
```

```
llama-stack-client providers list
```

```
llama-stack-client models list
```

```
llama-stack-client inference chat-completion --message "hi"
```

```
ChatCompletionResponse(  
    completion_message=CompletionMessage(  
        content='How can I assist you today?',  
        role='assistant',  
        stop_reason='end_of_turn',  
        tool_calls=[]  
    ),  
    logprobs=None,  
    metrics=[  
        Metric(metric='prompt_tokens', value=12.0, unit=None),  
        Metric(metric='completion_tokens', value=17.0, unit=None),  
        Metric(metric='total_tokens', value=29.0, unit=None)  
    ]  
)
```

# Let's use our stack

From the Python client

```
import rich
from llama_stack_client import LlamaStackClient

client = LlamaStackClient(base_url="http://0.0.0.0:8321")
models = client.models.list()
rich.print(models)
```

```
[  
    Model(  
        identifier='meta-llama/Llama-3.2-3B-Instruct',  
        metadata={},  
        api_model_type='llm',  
        provider_id='ollama',  
        provider_resource_id='llama3.2:3b-instruct-fp16',  
        type='model',  
        model_type='llm'  
    )  
]
```

.../notebooks/client.ipynb

```
import rich

response = client.inference.chat_completion(
    model_id="meta-llama/Llama-3.2-3B-Instruct",
    messages=[
        {"role": "system", "content": "You are a friendly AI language model."},  
        {"role": "user", "content": "Write a two-sentence poem about llamas."}
    ],
)
rich.print(response)

ChatCompletionResponse(  
    completion_message=CompletionMessage(  
        content='Here is a short poem about a llama:\n\nWith a coat of brown and white,  
        role='assistant',  
        stop_reason='end_of_turn',  
        tool_calls=[]
    ),  
    logprobs=None,  
    metrics=[  
        Metric(metric='prompt_tokens', value=30.0, unit=None),  
        Metric(metric='completion_tokens', value=37.0, unit=None),  
        Metric(metric='total_tokens', value=67.0, unit=None)
    ]
)
```

# Let's use our stack

From the REST API

<http://localhost:8321/docs>

```
curl -X POST http://localhost:8321/v1/inference/chat \
-H "Content-Type: application/json" \
-d '{"model_id": "meta-llama/Llama-3.2-3B-Instruct", "text": "Hello, how can I help you today?"}'
```

FastAPI 0.1.0 OAS 3.1

[/openapi.json](#)

## default

`POST /v1/inference/batch-chat-completion` Endpoint

`POST /v1/inference/batch-completion` Endpoint

`POST /v1/inference/chat-completion` Endpoint

`POST /v1/inference/completion` Endpoint

`POST /v1/inference/embeddings` Endpoint

`POST /v1/openai/v1/chat/completions` Endpoint

`POST /v1/openai/v1/completions` Endpoint

`GET /v1/telemetry/traces/{trace_id}/spans/{span_id}` Endpoint

`POST /v1/telemetry/spans/{span_id}/tree` Endpoint

`GET /v1/telemetry/traces/{trace_id}` Endpoint

`POST /v1/telemetry/events` Endpoint

`POST /v1/telemetry/spans` Endpoint

`POST /v1/telemetry/traces` Endpoint

`POST /v1/telemetry/spans/export` Endpoint

`GET /v1/providers/{provider_id}` Endpoint

`GET /v1/providers` Endpoint

`GET /v1/models/{model_id}` Endpoint

`DELETE /v1/models/{model_id}` Endpoint

`GET /v1/models` Endpoint

`POST /v1/models` Endpoint

`GET /v1/openai/v1/models` Endpoint



# Let's break our stack

# Add stuff to our stack

*Let's break some things ...*

- Add an OpenAI provider in the provider section

```
- provider_id: openai
  provider_type: remote::openai
  config:
    api_key: ${env.OPENAI_API_KEY:}
```

- Add an OpenAI model in the model section

```
- model_id: openai/gpt-4o
  provider_id: openai
  provider_model_id: openai/gpt-4o
```

# Add stuff to our stack

*Let's break more things ...*

- Add a tool runtime provider

```
$ llama stack list-providers | grep -i tool
tool_runtime    inline::rag-runtime
tool_runtime    remote::bing-search
tool_runtime    remote::brave-search
tool_runtime    remote::model-context-protocol
tool_runtime    remote::tavily-search
tool_runtime    remote::wolfram-alpha
                blobfile,chardet,pypdf,tqdm,numpy,scikit-learn,scipy,nltk,sentenceop:
                requests
                requests
                mcp
                requests
                requests
```

# Add stuff to our stack

*Let's break even more things ...*

- Add the agents Api

```
version: "2"
image_name: masterclass_minimal
apis:
  - inference
  - telemetry
  - agents
providers:
[ ... ]
```

# The providers registry

# The providers registry

Different providers have different dependencies

```
File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/distribution/impls.py"
    impls = await resolve_implementations(run_config, provider_registry or get_provider_registry(run_config), dist_registry)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/distribution/providers/base.py"
    return await instantiate_providers(sorted_providers, router_apis, dist_registry)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/distribution/providers/base.py"
    impl = await instantiate_provider(provider, deps, inner_implementations, dist_registry)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/distribution/providers/base.py"
    impl = await fn(*args)
    ^^^^^^^^^^

File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/providers/remotely.py"
    from .openai import OpenAIInferenceAdapter
File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/providers/remotely.py"
    from llama_stack.providers.utils.inference.litellm_openai_mixin import LiteLLMOpenAIMixin
File "/Users/diego/src/ai-masterclass-2025/masterclass_minimal/lib/python3.12/site-packages/llama_stack/providers/utils.py"
    import litellm
ModuleNotFoundError: No module named 'litellm'
```

# The providers registry

Different providers have different dependencies

```
File "/Users/diego/src/ai-masterclass-2025/masterclass_m:  
    import litellm  
ModuleNotFoundError: No module named 'litellm'
```

llama-

stack/llama\_stack/providers/registry/inference.py

```
1  def available_providers() -> list[ProviderSpec]:  
2      return [  
3          [...],  
4          remote_provider_spec(  
5              api=Api.inference,  
6              adapter=AdapterSpec(  
7                  adapter_type="openai",  
8                  pip_packages=["litellm"],  
9                  module="llama_stack.providers.remote",  
10                 config_class="llama_stack.providers.r",  
11                 provider_data_validator="llama_stack.",  
12             ),  
13         ),  
14     ]
```

# The providers registry

Apis can depend on other apis

Traceback (most recent call last):

```
[ ... ]  
File "/Users/diego/.local/share/uv/python/cpython-3.12.7-macos-aarch64-none/lib/python3.12/asyncio/runners.py", line :  
    return runner.run(main)  
          ^^^^^^^^^^^^^^  
  
File "/Users/diego/.local/share/uv/python/cpython-3.12.7-macos-aarch64-none/lib/python3.12/asyncio/runners.py", line :  
    return self._loop.run_until_complete(task)  
          ^^^^^^^^^^  
  
File "/Users/diego/.local/share/uv/python/cpython-3.12.7-macos-aarch64-none/lib/python3.12/asyncio/base_events.py", li:  
    return future.result()  
          ^^^^^^  
  
File "/Users/diego/src/llama-stack/llama_stack/distribution/stack.py", line 226, in construct_stack  
    impls = await resolve_implementations(run_config, provider_registry or get_provider_registry(run_config), dist_registry)  
          ^^^^^^^^^^  
  
File "/Users/diego/src/llama-stack/llama_stack/distribution/resolver.py", line 136, in resolve_implementations  
    return await instantiate_providers(sorted_providers, router_apis, dist_registry)  
          ^^^^^^  
  
File "/Users/diego/src/llama-stack/llama_stack/distribution/resolver.py", line 245, in instantiate_providers  
    deps = {a: impls[a] for a in provider.spec.api_dependencies}  
          ~~~~^  
  
KeyError: <Api.safety: 'safety'>
```

# The providers registry

Apis can depend on other apis

Traceback (most recent call last):

[ ... ]

File "/Users/diego/src/llama-stack/llama\_stack/distribut:  
deps = {a: impls[a] for a in provider.spec.api\_dependencies}  
~~~~~^\_\_^

KeyError: <Api.safety: 'safety'>

cat ~/src/llama-stack/llama\_stack/providers/registry/agents.py

llama-stack/llama\_stack/providers/registry/agents.py

```
1  def available_providers() -> list[ProviderSpec]:  
2      return [  
3          InlineProviderSpec(  
4              api=Api.agents,  
5              provider_type="inline::meta-reference",  
6              pip_packages=[  
7                  "matplotlib",  
8                  "pillow",  
9                  "pandas",  
10                 "scikit-learn",  
11             ],  
12             + kvstore_dependencies(),  
13             module="llama_stack.providers.inline.agents",  
14             config_class="llama_stack.providers.inline.AgentConfig",  
15             api_dependencies=[  
16                 Api.inference,  
17                 Api.safety,  
18                 Api.vector_io,  
19                 Api.vector_dbs,  
20                 Api.tool_runtime,  
21                 Api.tool_groups,  
22             ],  
23             ),  
24         ]
```

# Takeaways

## Concepts

- Llama Stack is a way to organize your LLM applications
- Providers are the main building blocks of Llama Stack
- You can group providers and resources into distributions

## Dev gotchas

- There's a lot going on! Even in our minimal disto
- Providers have their own (pip) dependencies
- APIs can depend on other APIs
- The build command specifies the providers list so you can create a virtual environment with the right dependencies

(Optional)

# Alternative Dev Env Setup

# Alternative Env Setup

Single virtualenv

- Our repo has a `pyproject.toml` file for `uv`

```
cd ~/src
git clone https://github.com/trustyyai-explainability/ai-masterclass-2025
git clone https://github.com/meta-llama/llama-stack.git # to get uv
```

```
cat pyproject.toml
```

```
uv sync
source .venv/bin/activate
```

- You can then incrementally update your venv via

```
--print-deps and uv add
```

```
llama stack build \
--config distributions/masterclass-minimal/build.yaml \
--image-type venv \
--print-deps
```

```
uv add [ ... ]
```

- As you `uv add` stuff, your `pyproject.toml` will update itself
- The `llama-stack` dependency is configured a local path and editable

```
1 [project]
2 name = "ai-masterclass-2025"
3 version = "0.1.0"
4 description = "AI Masterclass 2025 (for the dev setup"
5 readme = "README.md"
6 requires-python = "≥3.12"
7 dependencies = [
8     "llama-stack",
9 ]
10
11 [tool.uv.sources]
12 llama-stack = { path = "../llama-stack", editable = true }
```

# Alternative Env Setup

## VSCode / Cursor integration

- Our repo also has a `launch.json` file for the VSCode debugger
  - You can use this to run the stack with the debugger attached, and set breakpoints in the stack code

```
cat .vscode/launch.json
```

```
1 {
2     "version": "0.2.0",
3     "configurations": [
4         {
5             "name": "LS debug -- minimal-run.yaml",
6             "type": "python",
7             "request": "launch",
8             "module": "llama_stack.cli.llama",
9             "args": ["stack", "run", "distributions/masterrc"],
10            "console": "integratedTerminal",
11            "justMyCode": false,
12            "cwd": "${workspaceFolder}",
13            "envFile": "${workspaceFolder}/.env",
14            "env": {
15                "PYTHONPATH": "${workspaceFolder}"
16            }
17        },
18        {
19            "name": "LS debug -- agents-run.yaml",
20            "type": "python",
21            "request": "launch",
22            "module": "llama_stack.cli.llama",
23            "args": ["stack", "run", "distributions/masterrc"],
24            "console": "integratedTerminal",
25            "justMyCode": false,
26            "cwd": "${workspaceFolder}",
27            "envFile": "${workspaceFolder}/.env",
```



# Implement RAG from scratch

# Implement RAG from scratch

- Update or create a new distribution
- Add a vector database
- Implement a basic RAG pipeline in a notebook
- (No Agents for now)

# Implement RAG from scratch

```
vector_db_id = "my-vector-db"
client.vector_dbs.register(
    vector_db_id=vector_db_id,
    embedding_model="all-MiniLM-L6-v2",
    embedding_dimension=384,
)
```

```
client.tool_runtime.rag_tool.insert(
    documents=documents,
    vector_db_id=vector_db_id,
    chunk_size_in_tokens=512,
)
```

```
client.tool_runtime.rag_tool.query(
    content=prompt, vector_db_ids=[vector_db_id]
)
```

```
db_response = client.vector_io.query(
    vector_db_id=vector_db_id,
    query=prompt,
)
```

```
f"""
Please answer the given query using the context below.
```

```
QUERY:
{prompt}
```

```
CONTEXT:
{prompt_context}
"""
```

```
response = client.inference.chat_completion(
    messages=messages,
    model_id="meta-llama/Llama-3.2-3B-Instruct",
)
```

# Implement RAG from scratch (Agent version)

```
rag_agent = Agent(  
    client,  
    model="meta-llama/Llama-3.2-3B-Instruct",  
    instructions="You are a helpful assistant",  
    tools = [  
        {  
            "name": "builtin::rag/knowledge_search",  
            "args" : {  
                "vector_db_ids": [vector_db_id],  
            }  
        }  
    ],  
)
```

# Agents

# What is an agent?

Some definitions

From Hugging Face's Agents Course:

*An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.*

From Anthropic's Building Effective Agents:

- Workflows are systems where LLMs and tools are orchestrated through predefined code paths.
- Agents, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.

# Tools

From Hugging Face's Agents Course

---

- LLMs can only receive text inputs and generate text outputs. They have no way to call tools on their own.
- When we talk about providing tools to an Agent, we mean teaching the LLM about the existence of these tools and instructing it to generate text-based invocations when needed.
- From the user's perspective, it appears as if the LLM directly interacted with the tool, but in reality, it was the Agent that handled the entire execution process in the background.
- How do we give tools to an LLM?

# Agents on Llama Stack

## Basic concepts

- Agent Configuration
  - Model
  - Instructions
  - Tools
  - Shields
- Session
  - conversation thread with an agent
- Turn
  - a single interaction within a session
- Step
  - individual actions within a turn (inference, tool execution, shield calls, etc)

```
agent = Agent(  
    llama_stack_client,  
    model="my-model",  
    instructions="You are a helpful assistant that can use  
    tools=[ "my-tool", "my-other-tool"],  
    input_shields=[ "guardrails::my-filter"],  
    output_shields=[ "guardrails::my-filter"],  
)
```

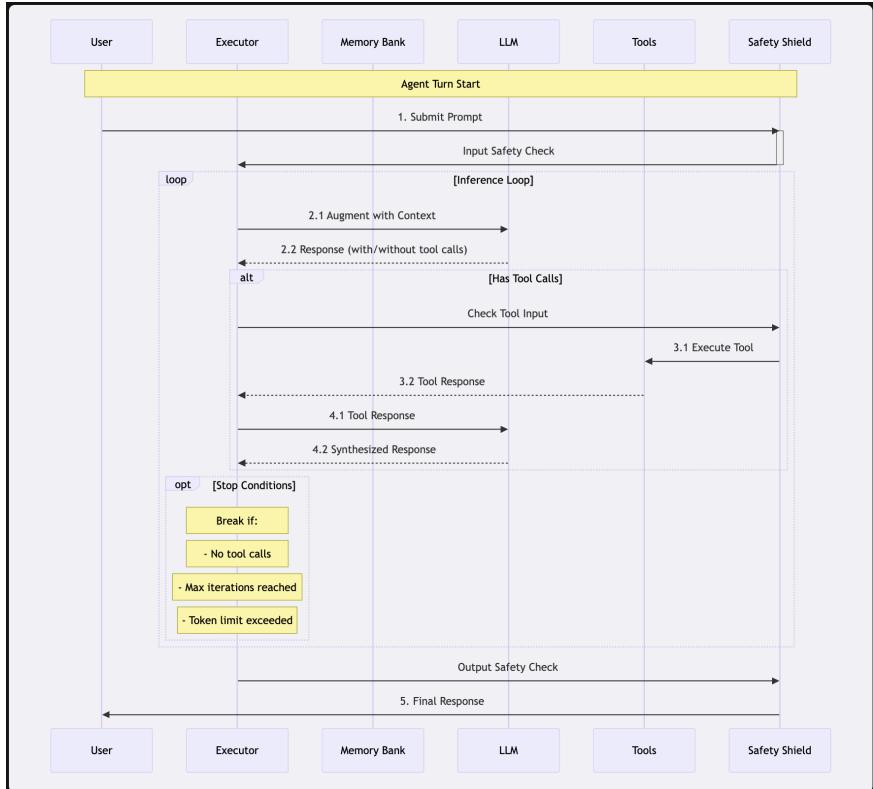
```
session_id = agent.create_session("test-session")
```

```
response = agent.create_turn(  
    messages=[{"role": "user", "content": prompt}],  
    session_id=session_id,  
)
```

# Agents on Llama Stack

## Execution model

1. Initial Safety Check
2. Inference Loop
  - Augment with context
  - Generate a response, potentially with tool calls and safety checks
  - Repeat
3. Final Safety Check



# Execution model, details

[llama\\_stack/providers/inline/agents/meta\\_reference/agent\\_instance.py](#)

```
async def run(self, session_id, turn_id, input_messages, sampling_params, stream, documents):
    # 1. Input Safety Checks
    if len(self.input_shields) > 0:
        # Run input safety shields
        async for res in self.run_multiple_shields_wrapper( ... ):
            yield res

    # 2. Main Agent Execution
    async for res in self._run( ... ):
        if isinstance(res, CompletionMessage):
            final_response = res
            break
        else:
            yield res

    # 3. Output Safety Checks
    if len(self.output_shields) > 0:
        # Run output safety shields
        async for res in self.run_multiple_shields_wrapper( ... ):
            yield res
```



An agent that can find its  
location

# An agent that can find its location

A tool to get a user's location

```
from llama_stack_client.lib.agents.client_tool import client_tool

@client_tool
def get_location(query: str = "location"):
    """
    Provide the location upon request.

    :param query: The query from user
    :returns: Information about user location
    """
    import geocoder

    try:
        g = geocoder.ip('me')
        if g.ok:
            return f"Your current location is: {g.city}, {g.state}, {g.country}"
        else:
            return "Unable to determine your location"
    except Exception as e:
        return f"Error getting location: {str(e)}"
```

# An agent that can find its location

Create an agent to use the `get_location` tool

```
agent = Agent(  
    client,  
    model="meta-llama/Llama-3.2-3B-Instruct",  
    instructions="You are a helpful assistant.",  
    tools=[get_location],  
)  
  
user_prompts = ["Where am I?"]  
session_id = agent.create_session("test-session")  
for prompt in user_prompts:  
    rich.print(f"User> {prompt}")  
  
    response = agent.create_turn(  
        messages=[{"role": "user", "content": prompt}],  
        session_id=session_id,  
        stream=True,  
    )  
    for log in AgentEventLogger().log(response):  
        log.print()
```

.../notebooks/tool.ipynb



An agent that can search the  
web

# An agent that can search the web

Llama Stack's built-in web search tool

- `builtin::websearch`
- Give your previous agent a tool to search the web
- Combine it with the `get_location` tool
- Use it to provide answers that are relevant to the user's location



An agent that can read files

# Model Context Protocol

Spin up a local MCP server

```
npx -y supergateway --port 8000 --stdio 'npx -y @modelcontextprotocol/server-filesystem my-local-files'
```

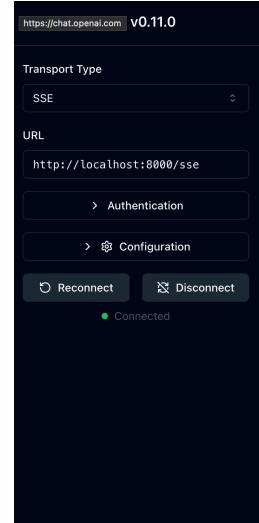
```
[supergateway] Starting ...
[supergateway] Supergateway is supported by Supermachine (hosted MCPs) - https://supermachine.ai
[supergateway]   - outputTransport: sse
[supergateway]   - Headers: (none)
[supergateway]   - port: 8000
[supergateway]   - stdio: npx -y @modelcontextprotocol/server-filesystem my-local-files
[supergateway]   - ssePath: /sse
[supergateway]   - messagePath: /message
[supergateway]   - CORS: disabled
[supergateway]   - Health endpoints: (none)
[supergateway] Listening on port 8000
[supergateway] SSE endpoint: http://localhost:8000/sse
[supergateway] POST messages: http://localhost:8000/message
[supergateway] Child stderr: Secure MCP Filesystem Server running on stdio

[supergateway] Child stderr: Allowed directories: [ '/Users/diego/src/ai-masterclass-2025/my-local-files' ]
```

# Model Context Protocol

Inspect the MCP server

```
npx -y @modelcontextprotocol/inspector
Starting MCP inspector...
🌐 Proxy server listening on port 6277
🔍 MCP Inspector is up and running at http://127.0.0.1:6277
```



The MCP Inspector interface is shown in two panels. The left panel displays configuration options: Transport Type (SSE), URL (http://localhost:8000/sse), Authentication (disabled), Configuration (disabled), Reconnect (button), and Disconnect (button). A green dot indicates the connection is active. The right panel shows a list of tools: `read_file`, `read_multiple_files`, and `create_file`. The `read_file` tool is selected, showing its description: "Read the complete contents of a file from the file system. Handles various text encodings and provides detailed error messages if the file cannot be read. Use this tool when you need to examine the contents of a single file. Only works within allowed directories." It includes a "path" input field and a "Run Tool" button.

https://chat.openai.com v0.11.0

Resources Prompts Tools Ping Sampling Roots

Transport Type  
SSE

URL  
http://localhost:8000/sse

> Authentication

> Configuration

Reconnect Disconnect  
Connected

Tools

List Tools  
Clear

`read_file`

Read the complete contents of a file from the file system. Handles various text encodings and provides detailed error messages if the file cannot be read. Use this tool when you need to examine the contents of a single file. Only works within allowed directories.

`read_multiple_files`

Read the contents of multiple files simultaneously. This is more efficient than reading files one by one when you need to analyze or compare multiple files. Each file's content is returned with its path as a reference. Failed reads for individual files won't stop the entire operation. Only works within allowed directories.

`create_file`

Create a new file or completely overwrite an existing file. This tool is useful for testing or generating large amounts of data. It supports both binary and text files.

path

Run Tool

# An agent that can read files

- Create a local directory with your files for testing
- Setup the filesystem MCP server
- Add a MCP provider to your stack
- Create an MCP agent



A “reasoning and acting”  
(ReAct) agent

# A “reasoning and acting” (ReAct) agent

- Compare a basic agent with a chain of prompts vs a ReAct agent
- ReAct agents are more flexible and can handle more complex tasks
- Should be able to ask "Are there any weather-related risks in my area?" and figure out it first needs to call the `get_location` tool and then use the `builtin::websearch` tool.
- Learn more about ReAct agents:
  - <https://www.ibm.com/think/topics/react-agent>
  - ReAct: Synergizing Reasoning and Acting in Language Models

# **Thank You!**