



Running Your Own LLM

Emily Apsey, Senior Manager Technical Marketing Engineering | 17 November 2023



Agenda

- Self-Managed vs Hosted API-Based LLMs
- Choosing and Using Self-Managed LLMs
- Building Local Copilots
- Optimizing for Size and Speed
- Emerging LLM Applications

Considerations for Choosing Self-Managed vs. Hosted API-based LLMs

Advantages of Self-Managed LLMs

- Training flexibility
- Throughput/batch processing
- Cost at scale
- Compliance and data governance
- Portability
- Controlled latency

Advantages of Hosted API-based LLMs

- Time to market
- Ease-of-use / integration into apps
- IT/infrastructure management
- Access to proprietary models

Rapid ecosystem evolution changes tradeoffs

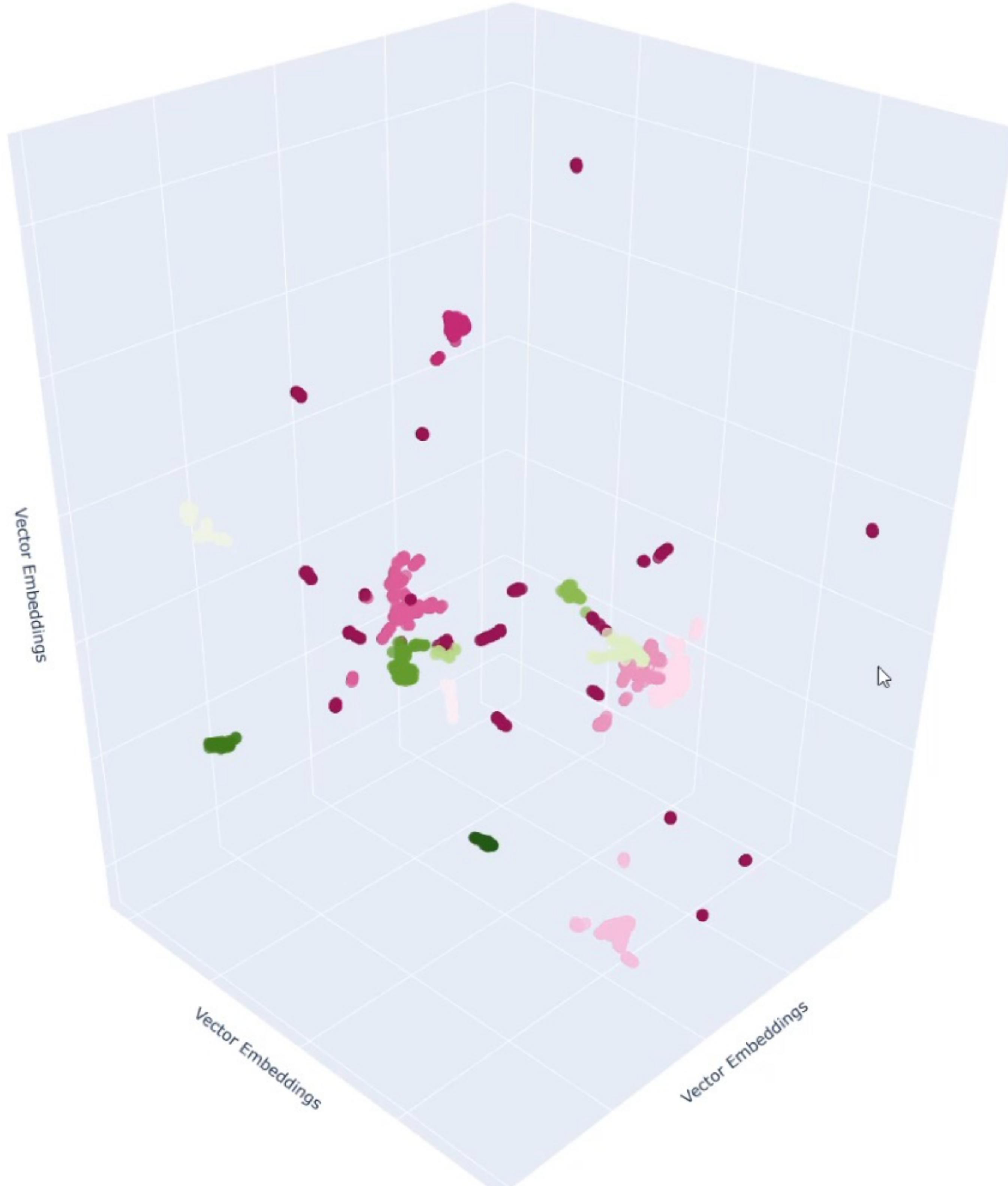
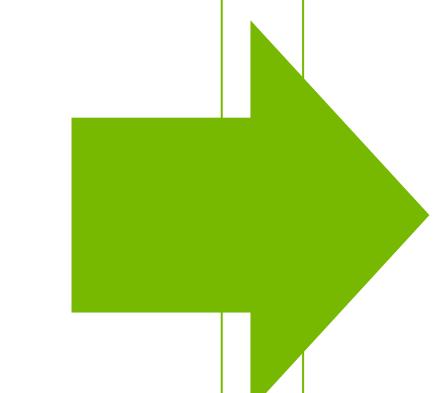
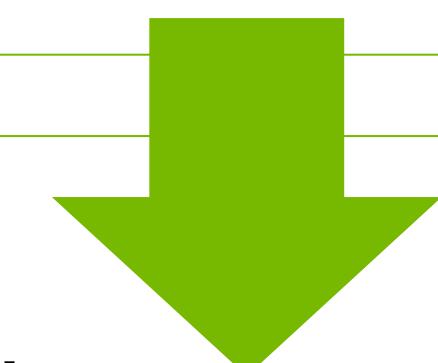
- Model hubs
- Inference engines, servers, and orchestrators
- Optimizers

Example App 1: Enterprise-scale Batch Data Processing

Enterprise-Scale Batch Data Processing

"I recently purchased the Celestial Cruiser from Star Bikes and have mixed feelings about my experience. The bike itself is beautifully designed and runs smoothly, however, the customer service I received was subpar. The delivery process was delayed, and the staff was not very helpful in addressing my concerns. Overall, I am satisfied with my bike, but I wish the customer service was better. I give it 3.5 out of 5 stars."

```
{  
    "overall_sentiment": "mixed",  
    "positive": [  
        "beautifully designed",  
        "runs smoothly"  
    ],  
    "negative": [  
        "subpar customer service",  
        "delayed delivery",  
        "unhelpful staff"  
    ],  
    "model": "Celestial Cruiser",  
    "rating": 3.5  
}
```



Products with the most positive sentiments:

- Stellar Speedster: 88% positive sentiments, 1640 positive reviews, 223 negative reviews
- Orion Trailblazer: 82% positive sentiments, 1940 positive reviews, 425 negative reviews
- Cosmic Commuter: 81% positive sentiments, 1070 positive reviews, 250 negative reviews

Common positive themes:

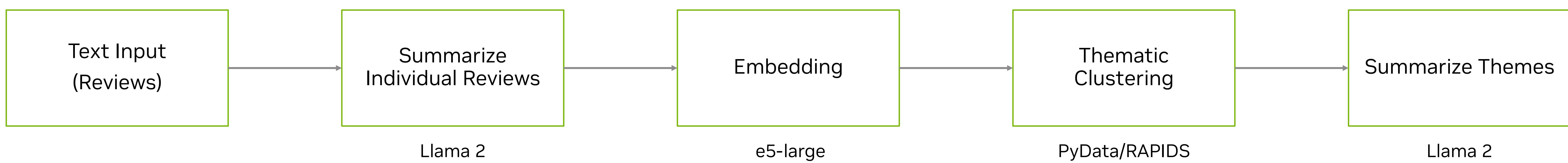
- Lightweight
- Easy to maneuver
- Sleek

Common negative themes

- Brakes not responsive
- Heavy
- Difficult to control

Product-specific highlights: the Stellar Speedster's lightweight and joy to ride features, the Orion Trailblazer's sleek and sturdy design, and the Cosmic Commuter's comfortable and efficient ride.

How It Works



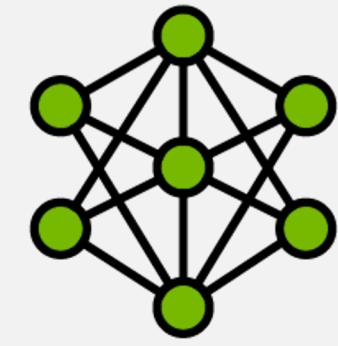
Benefits of self-managed, batch-processing deployment configuration:

- Cost efficiencies relative to always-on LLMs, direct benefit from continuous perf improvements (token/\$)
- Ease of adoption, similar to existing developer workflows

Choosing and Using Self- Managed LLMs

Interpreting Model Classes

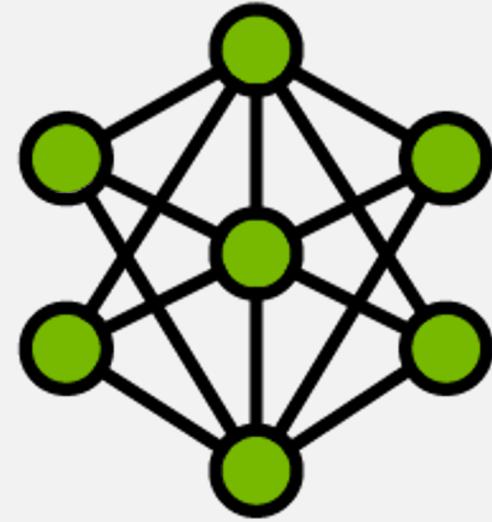
Fastest Responses



Up to 20B

Deploy at the edge, low-latency, high-throughput
Domain-specific applications

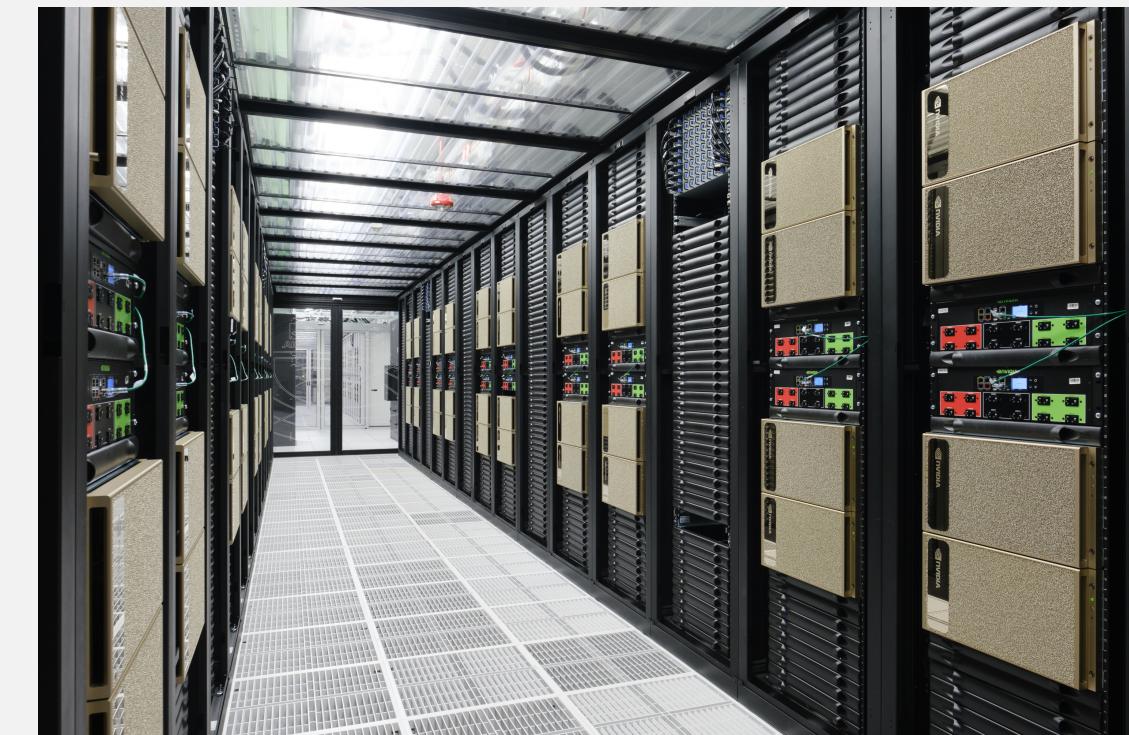
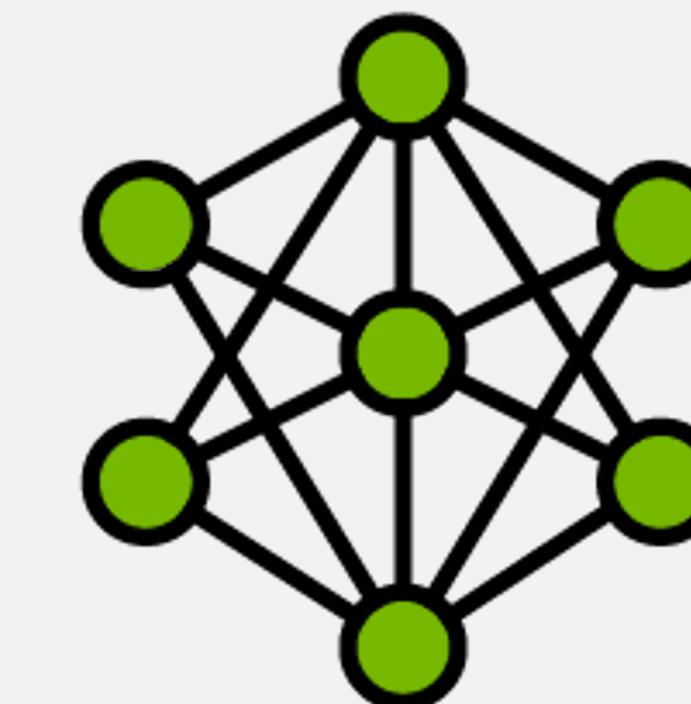
Balance of Quality & Latency



20B-175B

Need single-node HW
General-purpose applications

Solving Complex Problems



Over 175B

Need specialized software/techniques
Most demanding applications

Software for Efficient Inference

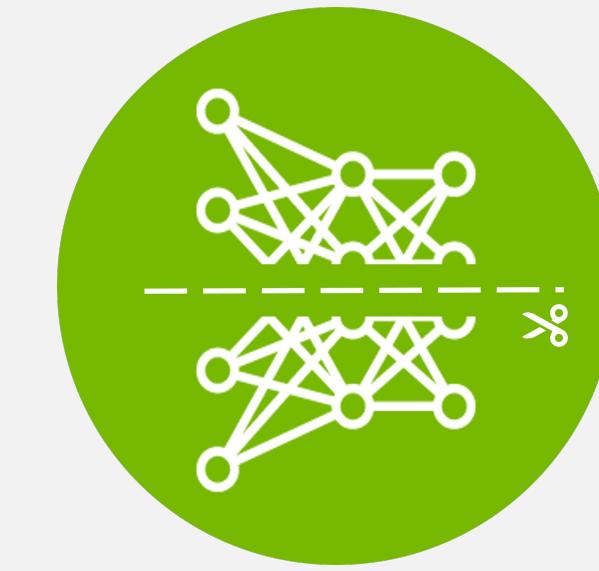


LLM

Inference-optimized
versions of SOTA LLMs

Accelerated workflow for at-
scale inference

Open-source AI kernels of
cutting-edge techniques



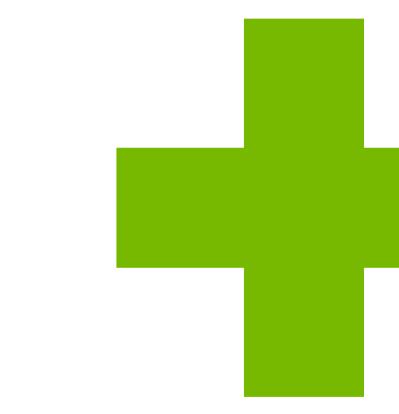
Inference Server

Parallelization (multi-device,
multi-node)

Concurrency, scheduling and
ensembling

Efficient inter-process
communication

Token output streaming



Optimizer

Dynamic batching

Quantization /
mixed precision

Kernel fusion

NVIDIA LLM Inference Stack

NeMo Service

Experiment with & deploy LLMs on-prem or cloud



NeMo Framework: E2E model lifecycle

Guardrails, data curation, pretrained models, prebuilt e2e pipelines for deploying LLMs



Triton Inference Server: Model Serving

High Performance Inference Serving for AI model production deployments



TensorRT-LLM: Model Optimizer

Speed-of-Light LLM optimization

TensorRT

NVIDIA kernel libs

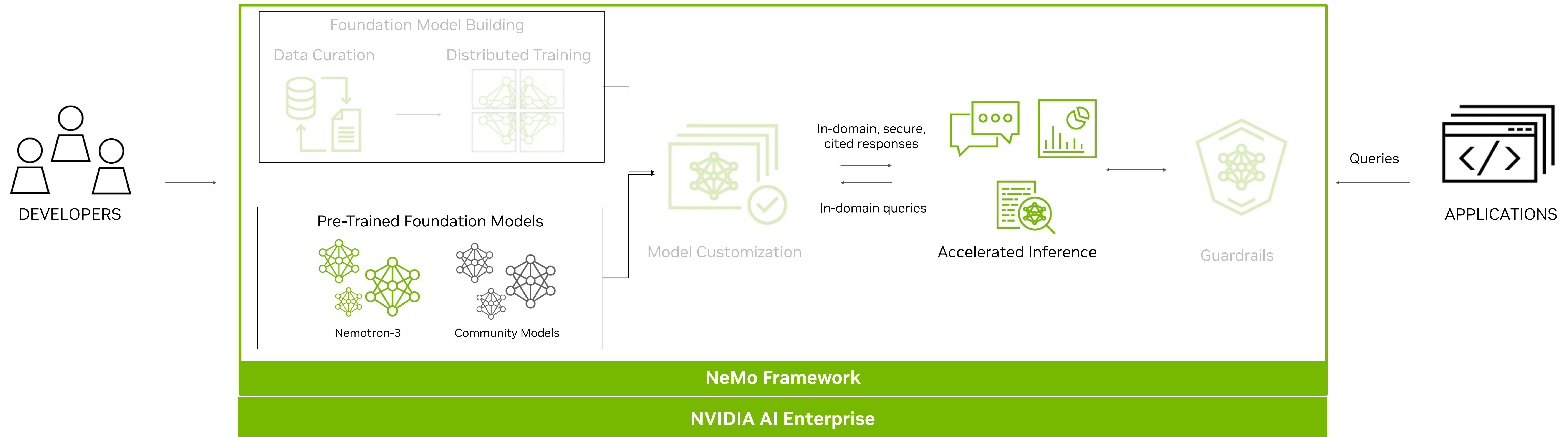
NVIDIA AI Enterprise

The Runtime for Production AI



NVIDIA NeMo Framework

From foundation model to application



Microsoft
Azure



ORACLE®



DELL Technologies

Hewlett Packard
Enterprise

Lenovo™

Example App 2: Portable Copilots

Why Deploy at the Edge?



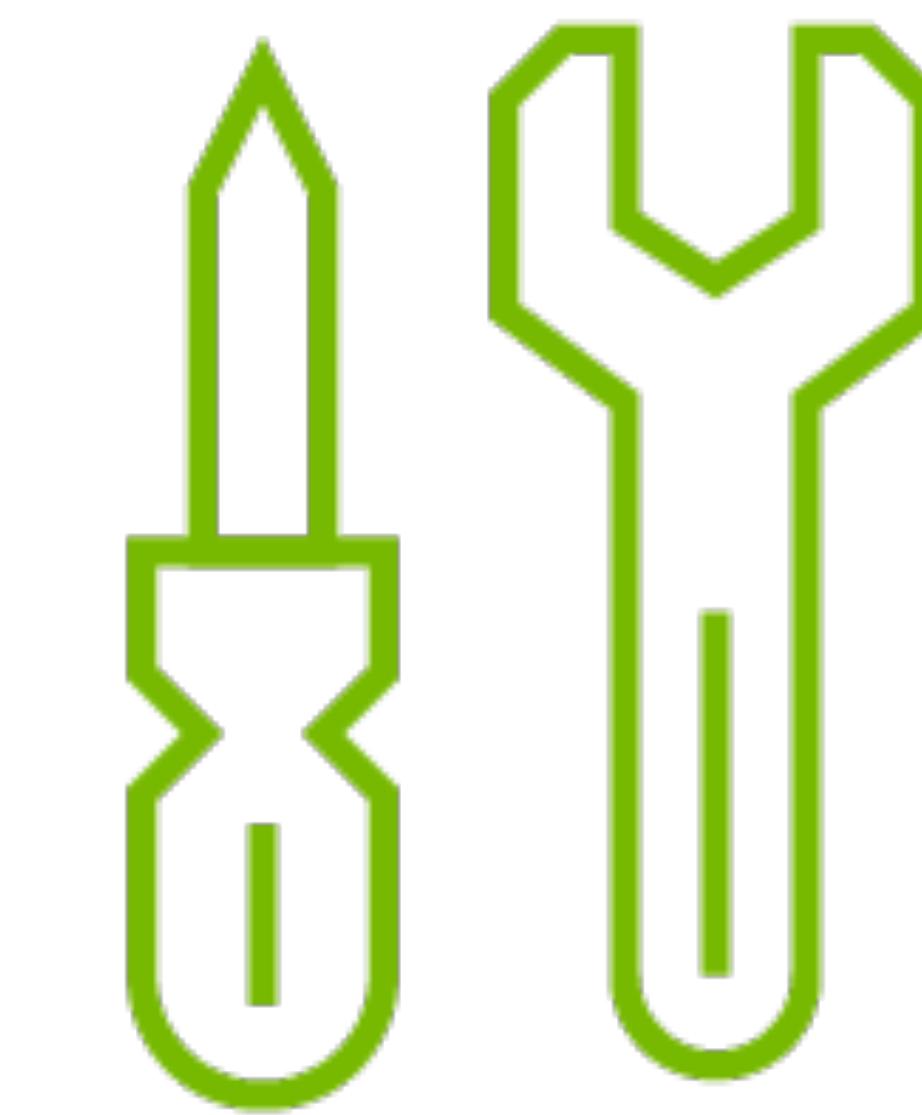
User needs

Lower latency
Reduced dependency on internet access



Security & Compliance

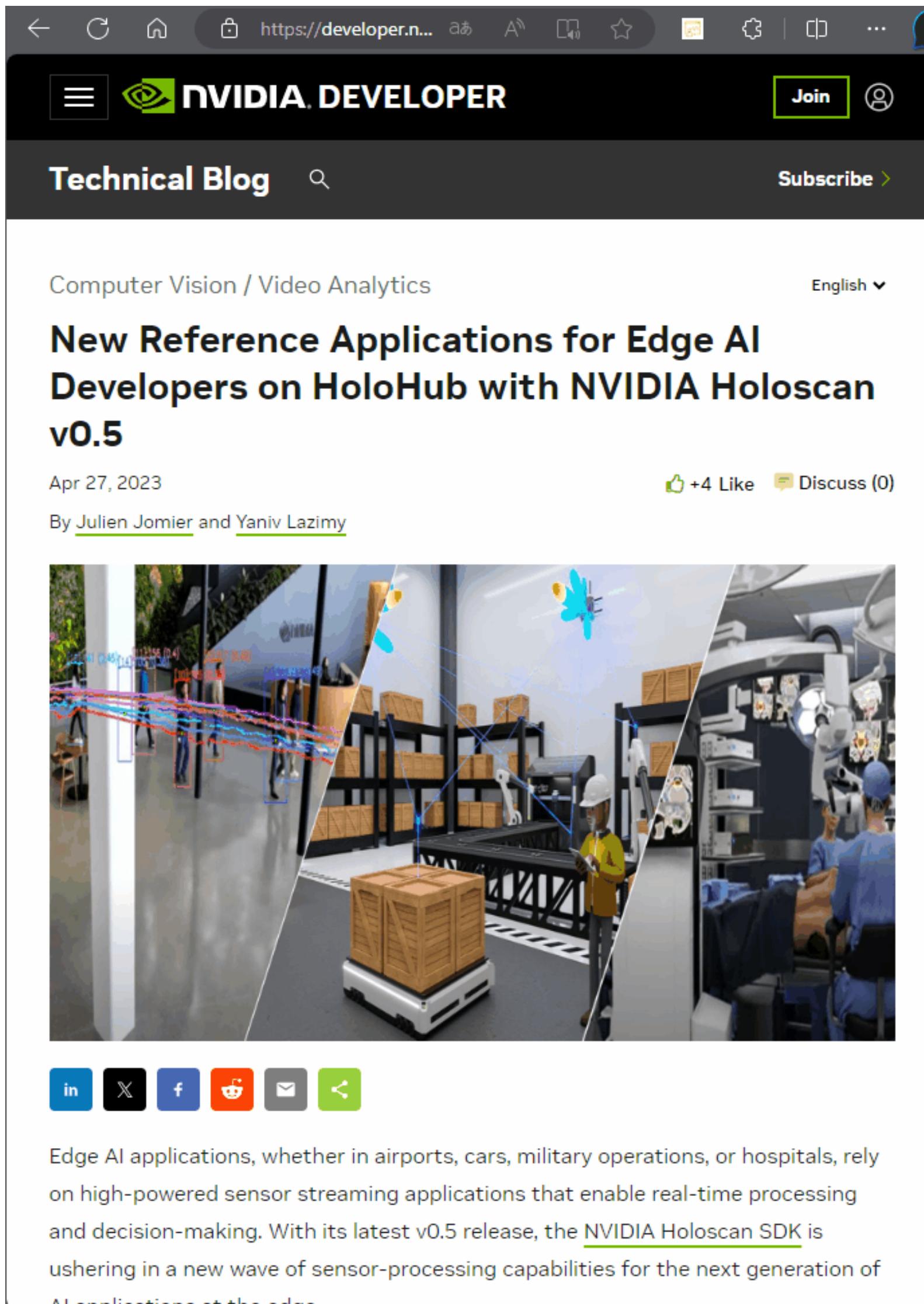
Data locality and governance



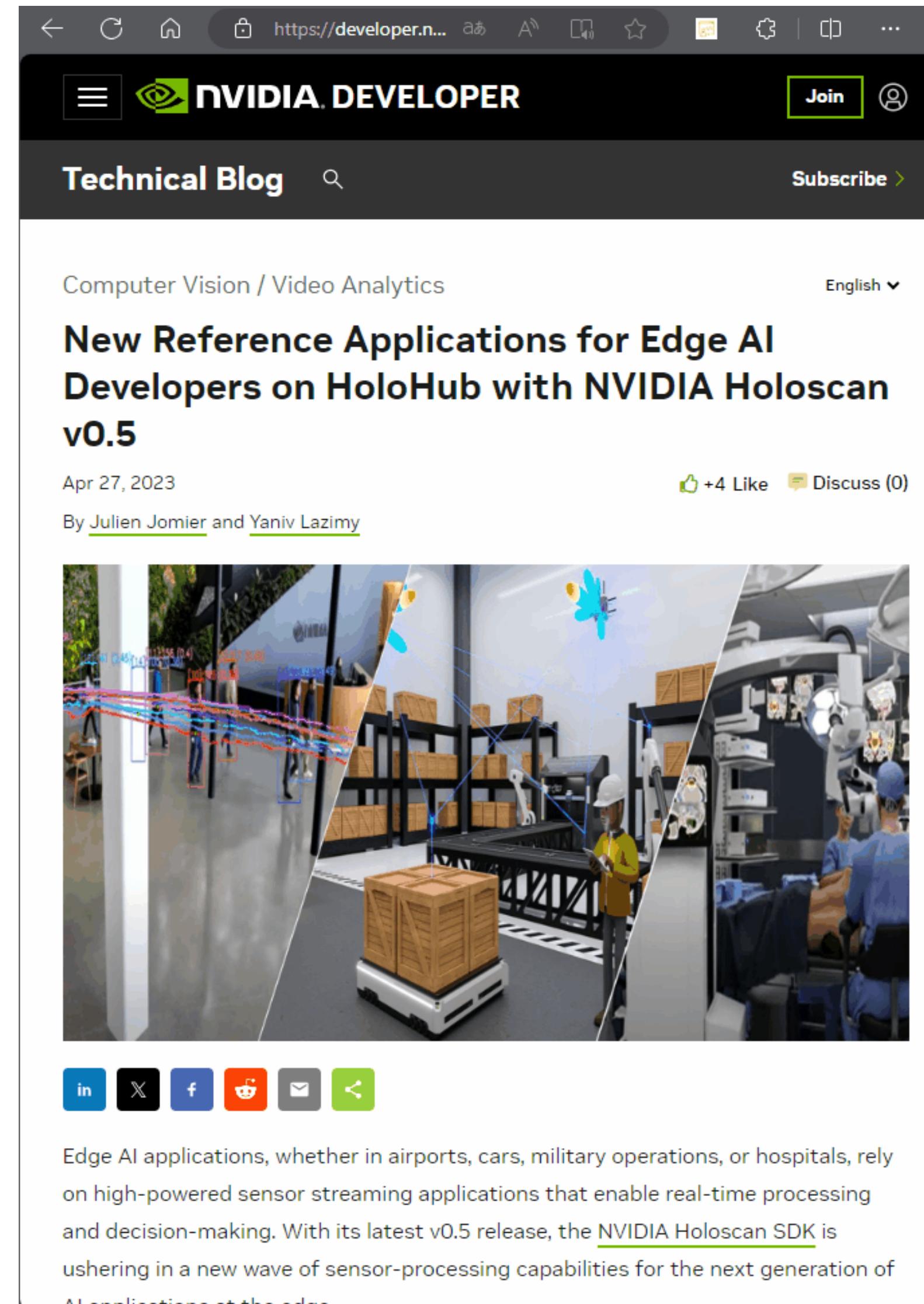
Tools and Efficiency

Take advantage of multipurpose hardware

Example Copilots



Summarization



Question Answering

This screenshot shows the DPSE COPILOT DASHBOARD interface. On the right, a code editor displays a Python file named "holoscan_sample.py" with code for defining an operator and a simple application. On the left, a sidebar shows "DPSE COPILOT AGENTS" and a "Chat" section with a message from the copilot asking "Hello! How can I help you today?".

```
from holoscan.conditions import CountCondition
from holoscan.core import Application, Operator, OperatorSpec
# define custom Operators for use in the demo

class HelloWorldOp(Operator):
    """Simple hello world operator.

    This operator has no ports.

    On each tick, this operator prints out hello world messages.
    """

    def setup(self, spec: OperatorSpec):
        pass

    def compute(self, op_input, op_output, context):
        print("")
        print("Hello World!")
        print("")

# Now define a simple application using the operator defined above

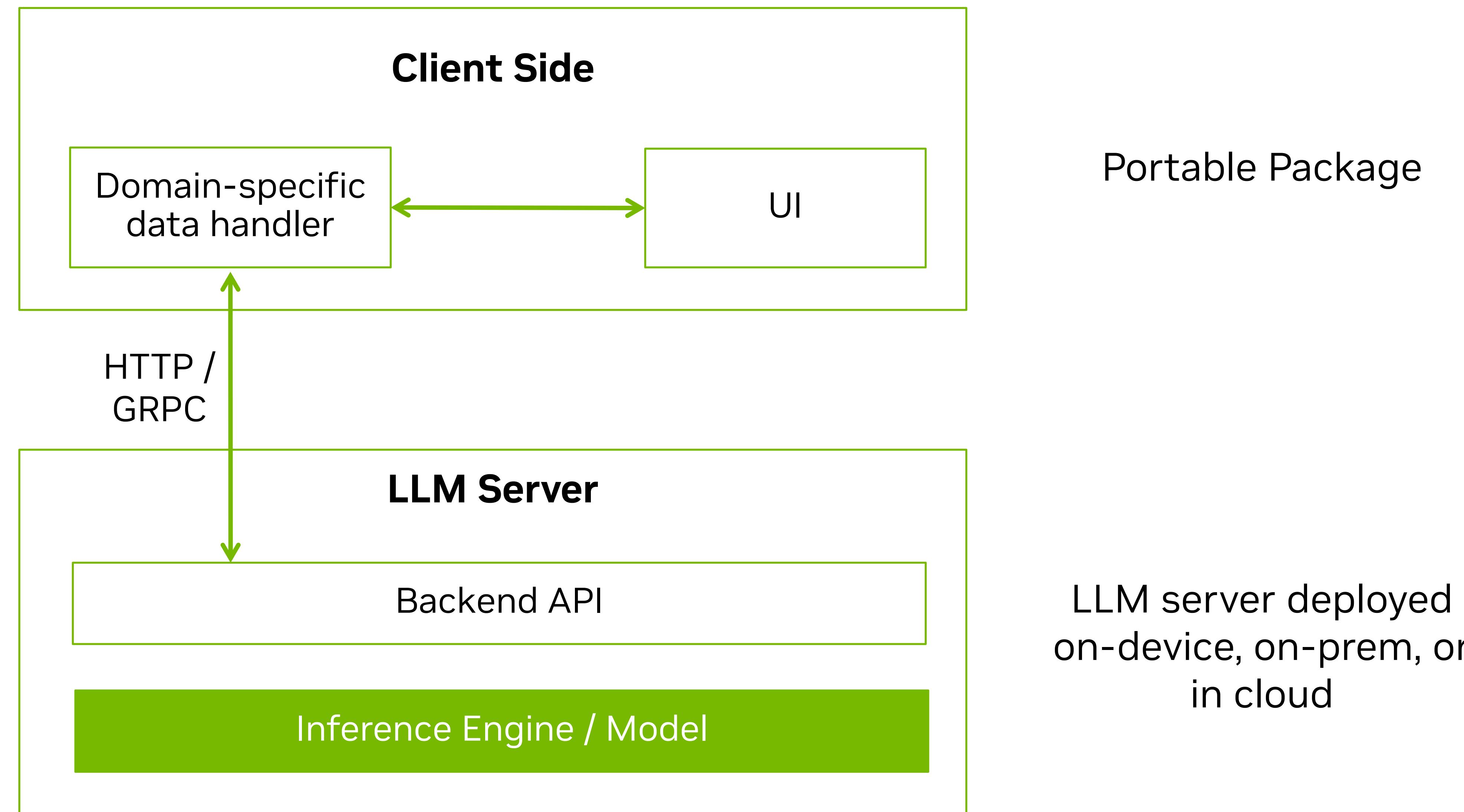
# Define the HelloWorldApp workflow
class HelloWorldApp(Application):
    def compose(self):
        #TODO
        pass

    def main():
        app = HelloWorldApp()
        app.run()

if __name__ == "__main__":
    main()
```

Chatbot and interactive code generation
with VS Code Extension

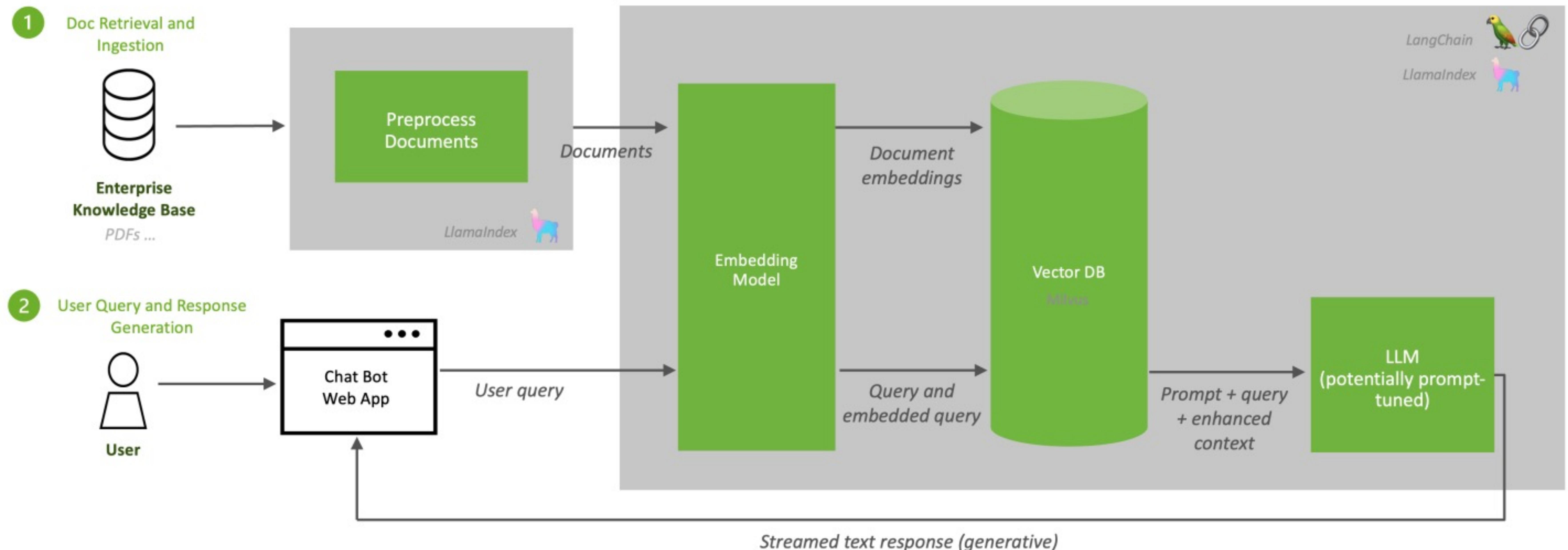
How It Works



Building Local Copilots

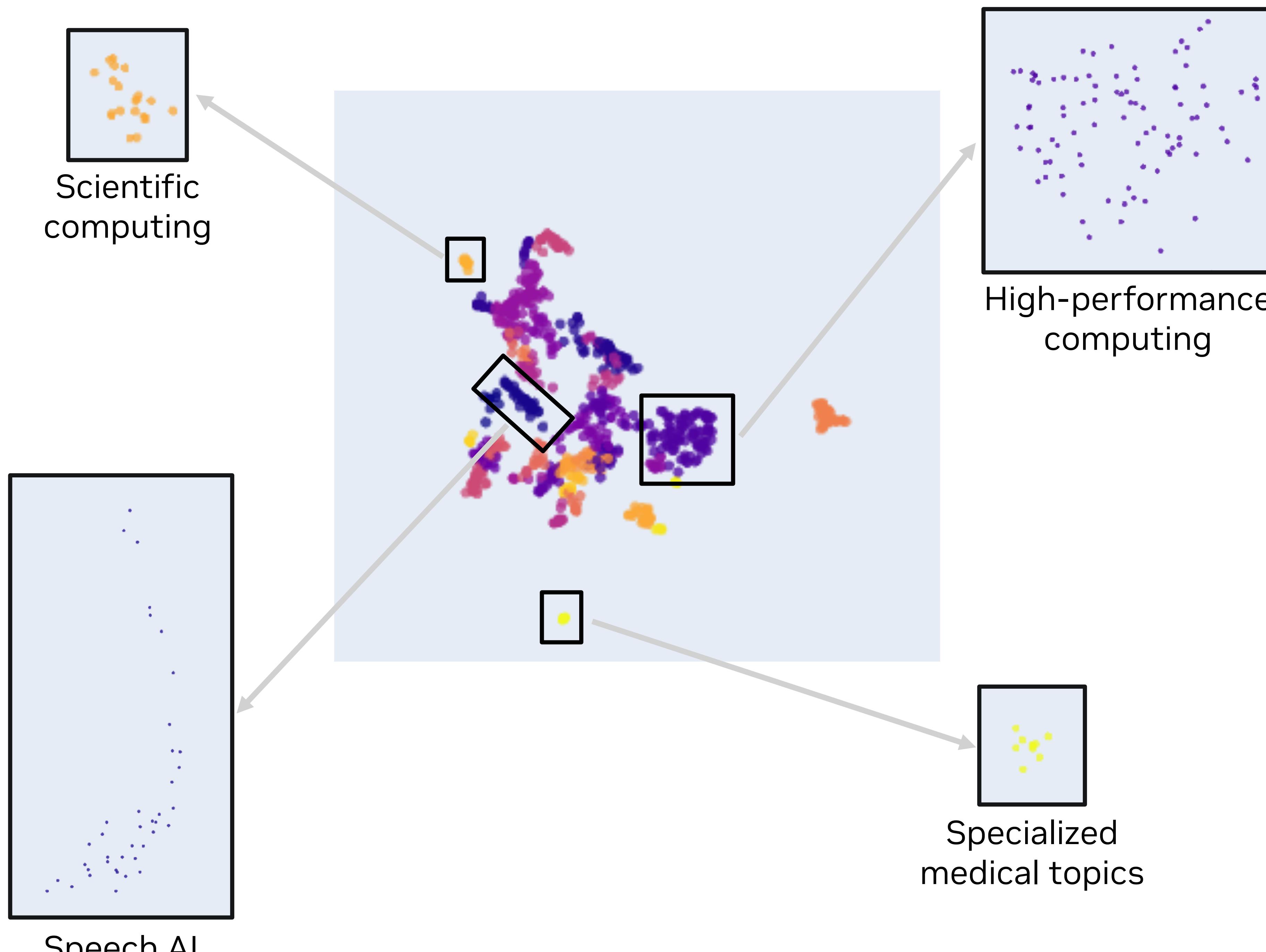
Canonical RAG Workflow

Self-managed examples



Embeddings and the Vector Database

Searching via semantic similarity



2D representation of a 768-dimension embedding space

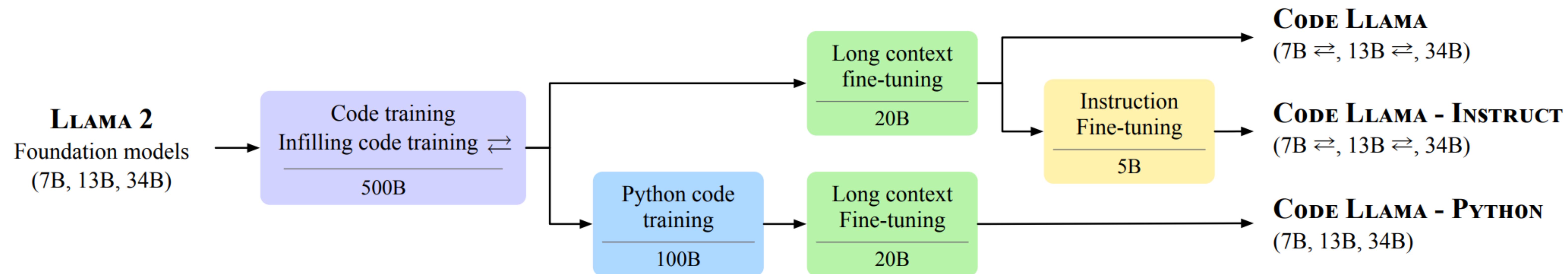
- Embeddings are data (text, image, or other data) represented as numerical vectors
 - Input text to embedding model to output vector
- Part of **semantic search**
 - Model trained to embed similar inputs close together
- Other use cases: classification, clustering, topic discovery
- Many pretrained and trainable embedding model sources
 - Modern ones are often deep neural networks

Query: Who will lead the construction team?
Chunk 1: The construction team found lead in the paint.
Chunk 2: Ozzy has been picked to lead the group.

Chunk 1 shares more keywords with the query, but semantic search can differentiate the meanings of "lead" and understand that "team" and "group" are similar, so Chunk 2 may be more helpful for the query.

Code Generation Models

Building from foundation models



Key capabilities: Completion, Infilling, Zero-shot

Other popular code models:

- Trained on code only: AlphaCode (Li et al., 2022) , InCoder (Fried et al., 2023) StarCoder (Li et al., 2023)
- Fine-tuned from a general language model: Codex (Chen et al., 2021), Code Llama



Optimizing for
Size and Speed

TensorRT-LLM in the LLM Ecosystem

Open-source library for custom deployments

Challenges

- Multi-GPU/Multi-Node support for latest large models
- Variable workload (short/long query and responses)
- Maximize GPU efficiency for different serving scenarios latency/throughput/TCO

TensorRT-LLM

- For deployments with high amounts of customization or control required
 - New or customized model architectures
 - Fine-tuned control over optimizations, quantization, & performance
- In-flight batching, reduced precision/quantization, mixed precision

Most developers should start with NeMo Framework

NeMo Service

Experiment with & deploy LLMs on-prem or cloud

NeMo Framework

Guardrails, data curation, pretrained models, prebuilt e2e pipelines for deploying LLMs

Triton Inference Server

High Performance Inference Serving for AI model production deployments

TensorRT-LLM

Speed-of-Light LLM optimization

TensorRT

NVIDIA Internal kernel libs

NVIDIA LLM Inference Stack

Llama E2E Example

2x faster with just 10 lines of code

1. Instantiate Llama & load the weights

```
builder_config = builder.create_builder_config([...])
trtllm_llama = trtllm.models.LLama([...])
load_from_hf_llama(tensorrt_llm_llama, hf_llama, [...])

network = builder.create_network()
network.set_named_parameters(trtllm_llama.named_parameters())
engine = builder.build_engine(network, builder_config)
serialize_engine(engine, path)

with open(path, 'rb') as f:
    engine_buffer = f.read()
llama = trtllm.runtime.GenerationSession(engine_buffer, [...])

output = llama.decode(input, input_lengths, sampling_config)
```

Instantiating, building, and executing inference in TRT-LLM

Llama E2E Example

2x faster with just 10 lines of code

1. Instantiate Llama & load the weights
2. Build & serialize the engines

```
builder_config = builder.create_builder_config([...])
trtllm_llama = trtllm.models.LLama([...])
load_from_hf_llama(tensorrt_llm_llama, hf_llama, [...])

network = builder.create_network()
network.set_named_parameters(trtllm_llama.named_parameters())
engine = builder.build_engine(network, builder_config)
serialize_engine(engine, path)

with open(path, 'rb') as f:
    engine_buffer = f.read()
llama = trtllm.runtime.GenerationSession(engine_buffer, [...])

output = llama.decode(input, input_lengths, sampling_config)
```

Instantiating, building, and executing inference in TRT-LLM

Llama E2E Example

2x faster with just 10 lines of code

1. Instantiate Llama & load the weights
2. Build & serialize the engines
3. Load the engines

```
builder_config = builder.create_builder_config([...])
trtllm_llama = trtllm.models.LLama([...])
load_from_hf_llama(tensorrt_llm_llama, hf_llama, [...])

network = builder.create_network()
network.set_named_parameters(trtllm_llama.named_parameters())
engine = builder.build_engine(network, builder_config)
serialize_engine(engine, path)

with open(path, 'rb') as f:
    engine_buffer = f.read()
llama = trtllm.runtime.GenerationSession(engine_buffer, [...])

output = llama.decode(input, input_lengths, sampling_config)
```

Instantiating, building, and executing inference in TRT-LLM

Llama E2E Example

2x faster with just 10 lines of code

1. Instantiate Llama & load the weights
2. Build & serialize the engines
3. Load the engines
4. Run optimized inference using TRT-LLM runtime offering

```
builder_config = builder.create_builder_config([...])
trtllm_llama = trtllm.models.LLama([...])
load_from_hf_llama(tensorrt_llm_llama, hf_llama, [...])

network = builder.create_network()
network.set_named_parameters(trtllm_llama.named_parameters())
engine = builder.build_engine(network, builder_config)
serialize_engine(engine, path)

with open(path, 'rb') as f:
    engine_buffer = f.read()
llama = trtllm.runtime.GenerationSession(engine_buffer, [...])

output = llama.decode(input, input_lengths, sampling_config)
```

Instantiating, building, and executing inference in TRT-LLM

Optimization in TensorRT-LLM

Standard Dynamic Batching vs Inflight Batching

Serving with Inflight Batching: Comparison

Response from int4 quantized weights of Llama-2-13b-chat-hf with variable generation lengths. Maximum serving batch size is 4 for each server.

Question template: "Please provide one good name for a company that makes ____."

w/o Inflight batching

w/ Inflight batching

- colorful socks
- hair pins
- furniture
- mops
- car window shields
- liquid detergent
- harmonica
- winter gloves

Start multiple requests

Made with Streamlit

Inflight Batching

Maximizing GPU Utilization during LLM Serving

- Replaces completed requests in the batch
- Improves throughput, time to first token, & GPU utilization
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible through the TensorRT-LLM Batch Manager

		Iteration	1	2	3	4	5	6	7	8	9	...
Batch Elements		R ₁					END				R ₅	...
		R ₂		END							R ₆	...
Batch Elements		R ₃				END					R ₇	...
		R ₄								END	R ₈	...

Static Batching

		Iteration	1	2	3	4	5	6	7	8	9	...
Batch Elements		R ₁					END	R ₅			...	
		R ₂		END	R ₆						...	
Batch Elements		R ₃				END	R ₇		END	R ₉	...	
		R ₄							END	R ₈	...	

Inflight Batching

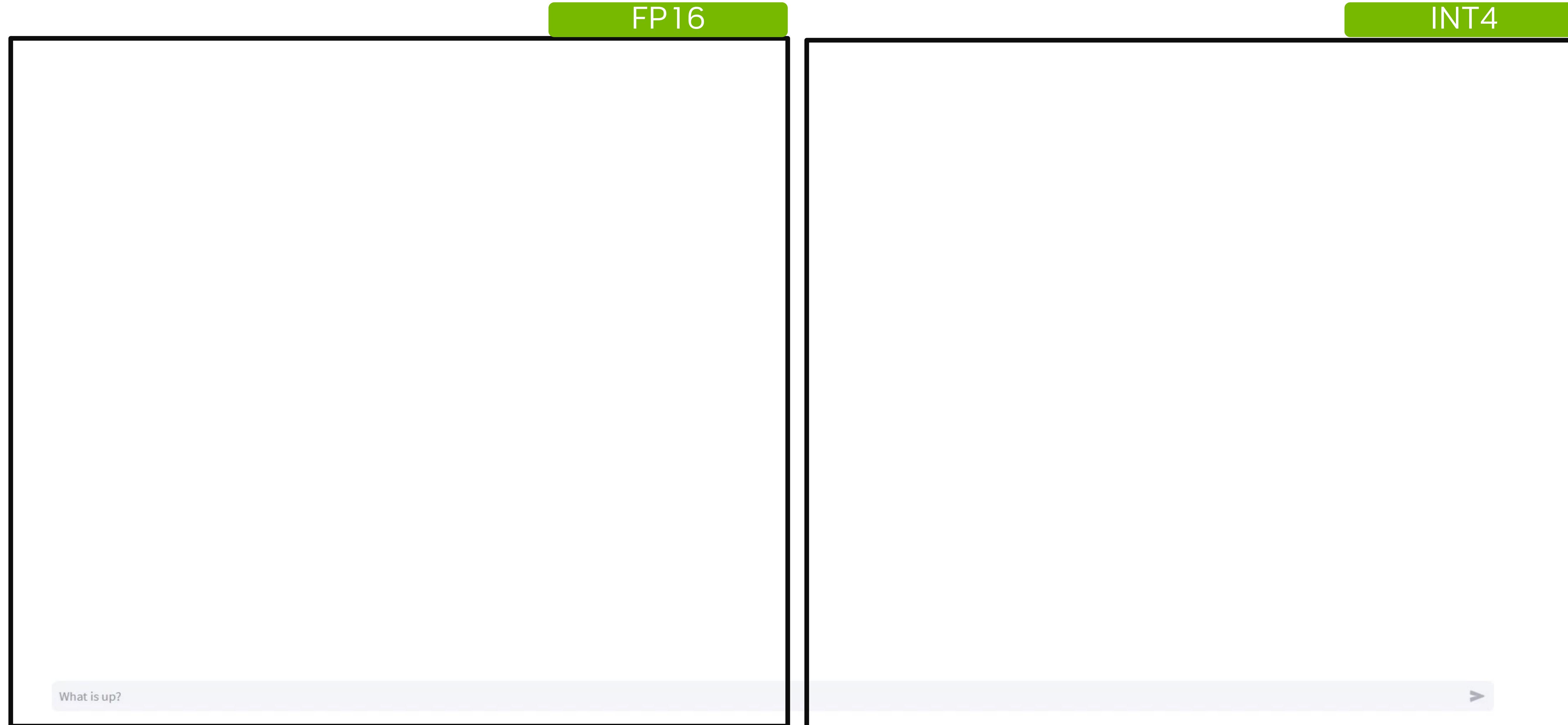
Context | Gen | EoS | NoOp

Optimization in TRT-LLM

Quantization for Better Throughput, Latency, Portability, and Scale

⋮

Llama-2 Quantization Comparison: FP16 v.s. INT4 Weights



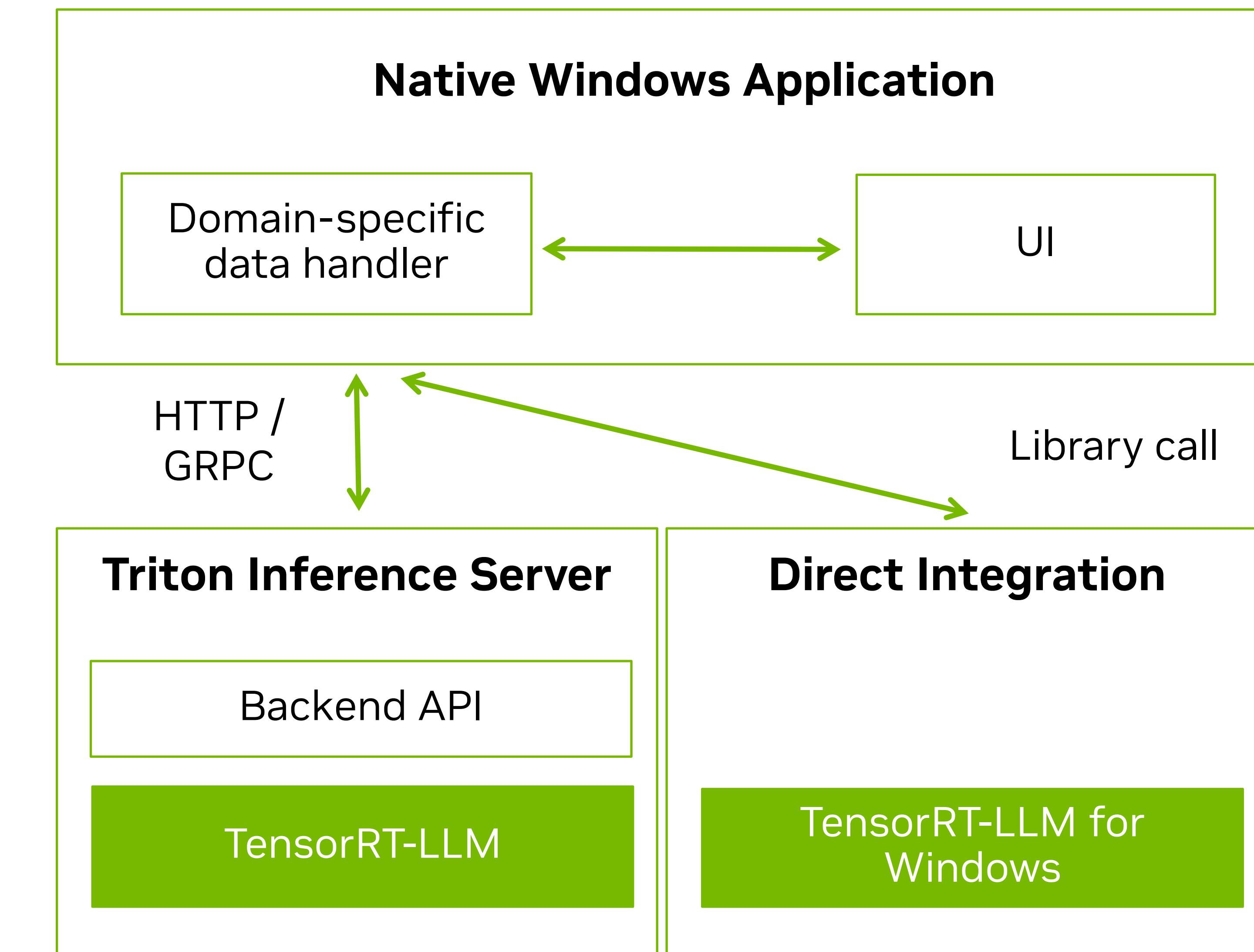
LLMs on Windows with TensorRT-LLM and RTX

Benefits of LLMs on Windows Clients

- Latency – low latency tolerances for some use cases (e.g. gaming, digital assistants)
- Availability – still need to write docs on an airplane
- Data privacy/locality – Keep private/proprietary data on-device

Development and Deployment

- Develop/train models with NeMo Framework, deploy on client
- TensorRT-LLM for Windows - simple direct integration
- Triton Inference Server on WSL - portability
- Compatibility with ChatGPT API



NVIDIA AI Foundry



NVIDIA AI Enterprise OFF

Tag: 23.08-pyt-python-py3 | Architecture: arm64

Scan Details

Scan Result: C	Policy Bundle: NGC Internal Policy	Scan Status: Scan Complete	Last Scanned: 10/23/2023 3:14 PM
--	--	----------------------------	----------------------------------

Image Digest: sha256:bef9ceff265329087128030208bbb03a22f9bba0d9559f78f2738321f4cfa0

Vulnerabilities

All: 208	Critical: 0	High: 9	Medium: 197	Low: 0
----------	-------------	---------	-------------	--------

Scanned By: anchor

Search vulnerabilities...

SEVERITY	IDENTIFIER	OUTPUT	MORE INFO
HIGH	CVE-2018-20225	Vulnerability found in non-os package type (python) - /usr/local/lib/python...	>
HIGH	CVE-2023-4911	Vulnerability found in os package type (dpkg) - libc-dev-bin (fixed in: 2.35-...	>
HIGH	CVE-2022-29501	Vulnerability found in os package type (dpkg) - libslurm37 (CVE-2022-295...	>
HIGH	CVE-2023-4911	Vulnerability found in os package type (dpkg) - libc6 (fixed in: 2.35-Oubun...	>

NVIDIA AI Enterprise ON

Tag: 23.08.02-pyt-python-py3 | Architecture: arm64

Scan Details

Scan Result: A	Policy Bundle: NGC Internal Policy	Scan Status: Scan Complete	Last Scanned: 10/26/2023 5:21 PM
--	--	----------------------------	----------------------------------

Image Digest: sha256:8b28b511bc1157738782108acf8110f54c34d953e7e55107ee126833867e20b8

Vulnerabilities

All: 134	Critical: 0	High: 0	Medium: 134	Low: 0
----------	-------------	---------	-------------	--------

Scanned By: anchor

Search vulnerabilities... Include no-risk vulnerabilities

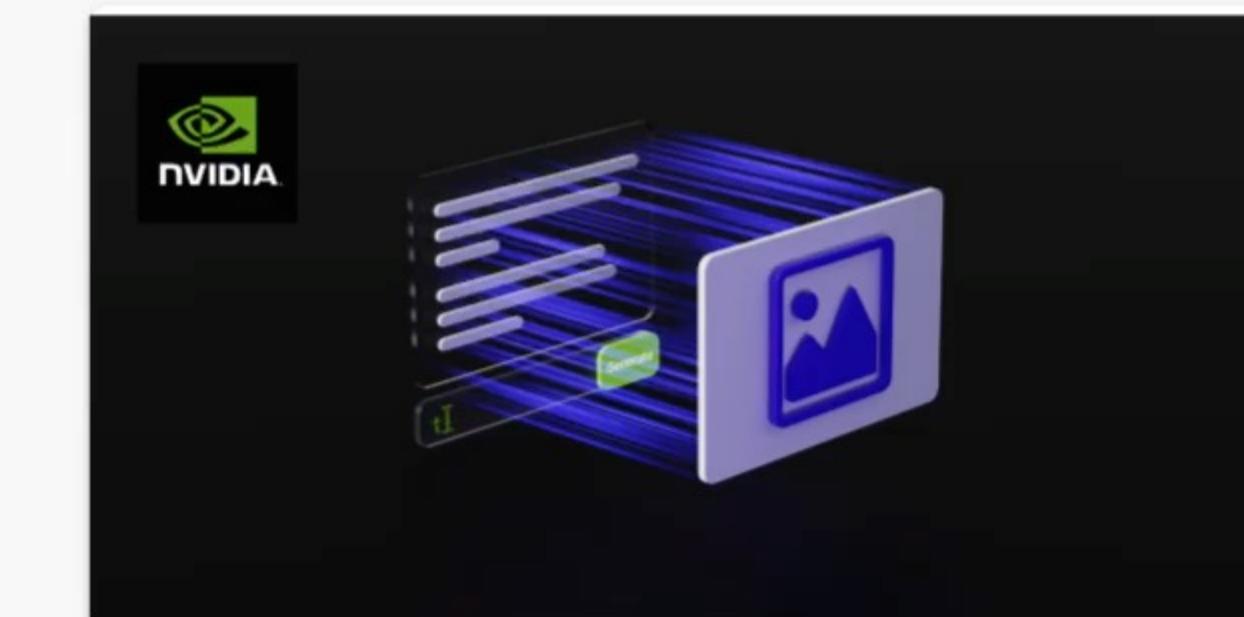
SEVERITY	IDENTIFIER	OUTPUT	MORE INFO
MEDIUM	CVE-2023-32681	Vulnerability found in os package type (dpkg) - python3-pip (CVE-2023-3...	>
MEDIUM	CVE-2023-4781	Vulnerability found in os package type (dpkg) - xxd (fixed in: 2:8.2.3995-1...	>
MEDIUM	CVE-2023-5535	Vulnerability found in os package type (dpkg) - vim (fixed in: 2:8.2.3995-1...	>
MEDIUM	CVE-2022-35205	Vulnerability found in os package type (dpkg) - libctf0 (CVE-2022-35205 -...	>

Emerging LLM Applications



Catalog > Models > NeVA: NeMo Vision and Language Assistant

NeVA: NeMo Vision and Language Assistant

[Fine Tune Model](#)[Get API Access](#)

Playground

Overview

Model Card++

Playground Mode: [User Interface](#)[Run on Windows with RTX](#)

Description

NeVA is a multi-modal vision-language model that understands text and images and generates informative responses.

Publisher

NVIDIA

Modified

October 13, 2023

[Language Generation](#)[Large Language Model](#)[Vision Assistant](#)[Visual Question Answering](#)[Object Detection](#)[NeMo](#)[Computer Vision](#)

AI models generate responses and outputs based on complex algorithms and machine learning techniques, and those responses or outputs may be inaccurate or indecent. By using this Playground, you assume the risk of any harm caused by any response or output of the model. X

By using this demo, you acknowledge that you have read and agreed to the [terms & conditions](#).

Model Description

NeVA is a multi-modal vision-language model that understands text and images and generates informative responses. This demo shows NeVA's ability to generate responses based on the image or text as an input.

Model Input Instructions

Upload an image and/or enter a prompt to receive an AI-generated response.

NeVA-43b

[Image](#)

Drop Image Here
- or -
Click to Upload

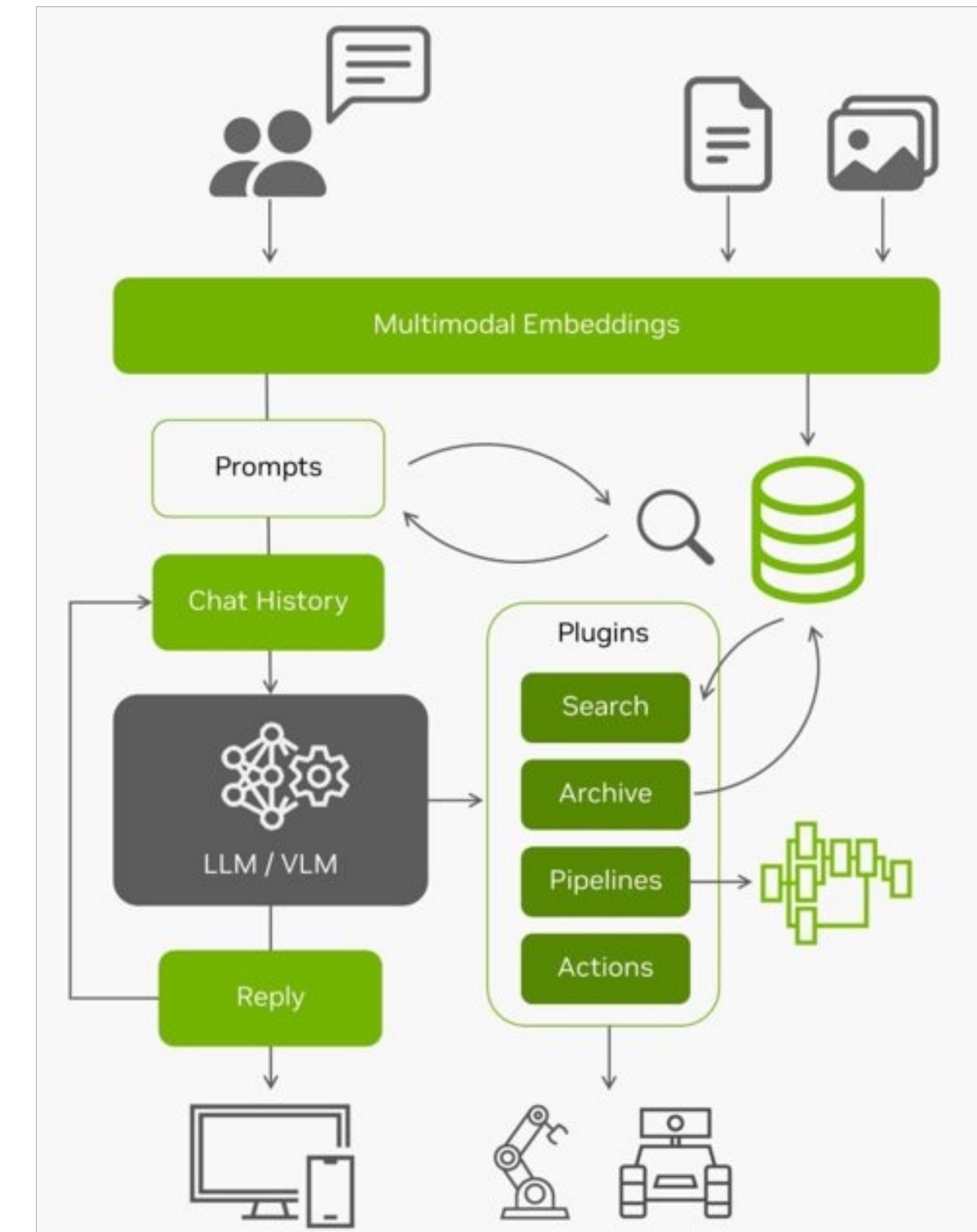
Preprocess for non-square image

Crop Resize Pad

NeVA Chatbot

LLM Agents

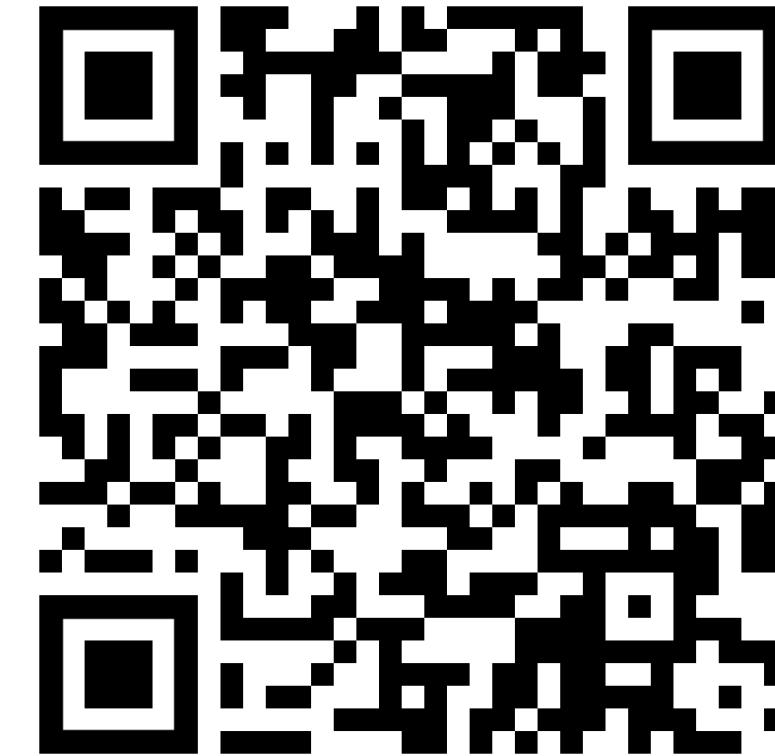
Beyond deterministic prompt chaining



Q & A

Apply to NVIDIA Inception for startups:

NVIDIA.com/startups



Join the NVIDIA Developer community to get access to technical training, technology, AI models and 600+ SDKs:

Developer.nvidia.com/join



Explore More Gen AI/LLM Training:

25% off Workshops for LLM Day registrants*

USE CODE: TRAIN-LLM

Instructor-Led Workshops

- > [Generative AI With Diffusion Models](#)
- > [Rapid Application Development Using LLMs](#)
- > [Efficient Large Language Model \(LLM\) Customizations](#)

Self-Paced Courses

- > [Generative AI Explained \(Free\)](#)
- > [Generative AI With Diffusion Models](#)

View our comprehensive Gen AI/LLM learning path, covering fundamental to advanced topics

- > [Gen AI/LLM Learning Path](#)

*Offer valid for any of the [DLI public workshops](#) scheduled through March 01, 2024.

