Future of LLM Productionization

Deepak Karunanidhi

Future of LLM deployment

Traditional AI models in production

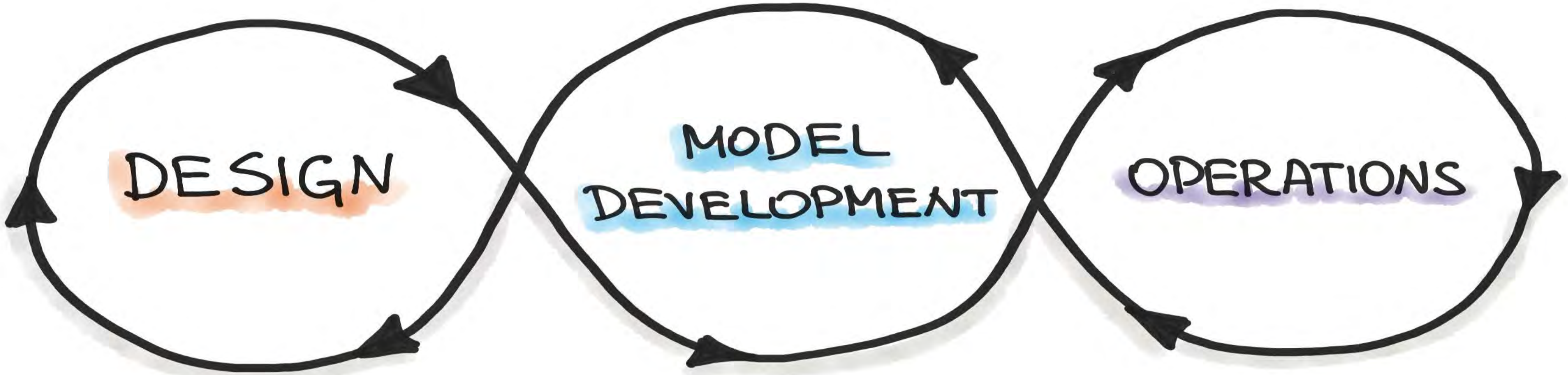
Deploying traditional AI models like BERT (Bidirectional Encoder Representations from Transformers) in production involves several steps to ensure efficiency, scalability, and reliability. Here's a generalized outline of the process:

1. **Model Training**: Initially, you need to train your BERT model on your dataset. This typically requires significant computational resources and may involve specialized hardware like GPUs or TPUs.
2. **Model Optimization**: Once trained, you might optimize your model for deployment. This could involve techniques like quantization (reducing the precision of weights) to decrease the model size and make it more efficient during inference.
3. **Deployment Environment Setup**: Set up the infrastructure for deploying your BERT model. This involves choosing a deployment environment such as cloud services (e.g., AWS, Azure, GCP) or on-premises servers.
4. **Model Serialization**: Serialize your trained BERT model into a format suitable for deployment. Common formats include TensorFlow's SavedModel format or PyTorch's TorchScript.

Cont…

5. **Model Serving**: Deploy the serialized model using a web server or specialized model-serving framework (e.g., TensorFlow Serving, TorchServe). These frameworks provide APIs for loading the model into memory and serving predictions over HTTP or other protocols.
6. **API Design**: Design an API for interacting with your BERT model. This might involve defining input/output formats, handling authentication, and implementing any necessary pre-processing or post-processing logic.
7. **Scalability and Load Balancing**: Ensure that your deployment setup can handle the expected load and scale dynamically as demand fluctuates. This may involve using load balancers and auto-scaling features provided by your deployment environment.
8. **Monitoring and Logging**: Implement monitoring and logging to track the performance and health of your deployed model. This includes metrics like latency, throughput, and error rates, as well as logging of input/output data for debugging and analysis.
9. **Security**: Implement security measures to protect your deployed model from unauthorized access and attacks. This might include encryption of data in transit and at rest, authentication and authorization mechanisms, and regular security audits.
10. **Continuous Integration/Continuous Deployment (CI/CD)**: Set up CI/CD pipelines to automate the process of testing, building, and deploying updates to your BERT model. This helps ensure rapid iteration and deployment of improvements or bug fixes.
11. **Versioning and Rollback**: Implement versioning for your deployed models to track changes over time and facilitate rollback to previous versions if needed.

MLOps

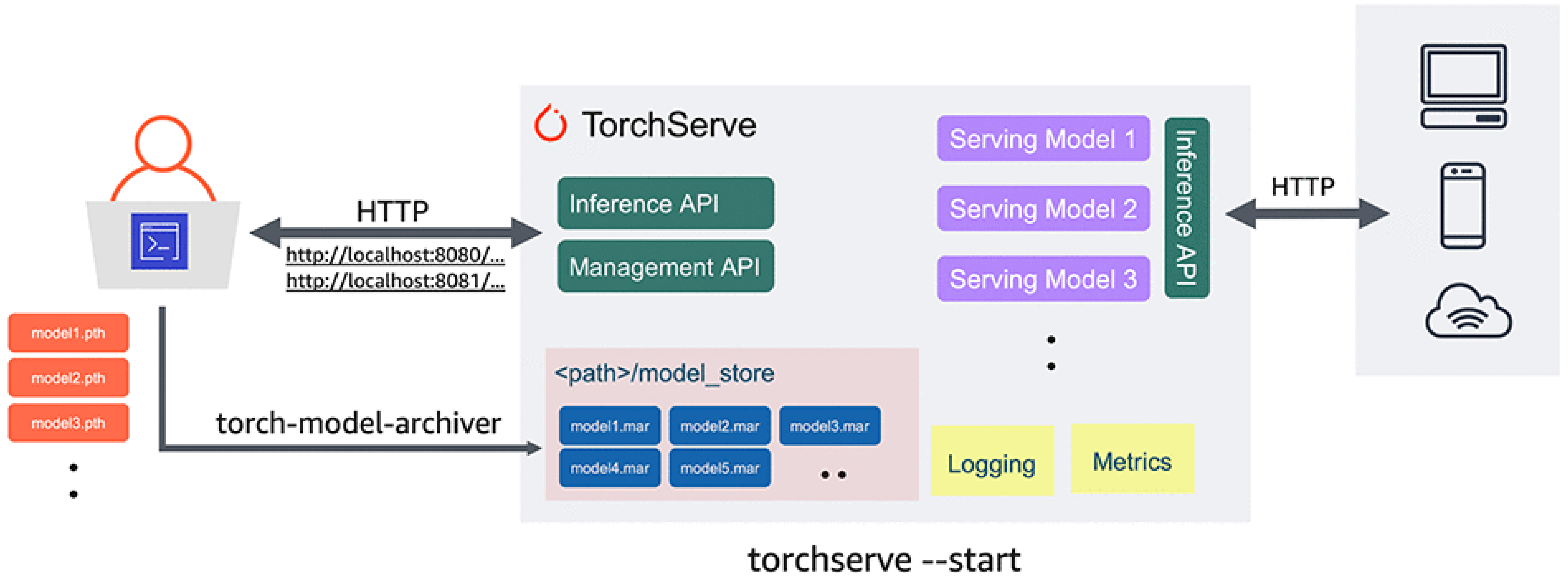


- Requirements Engineering
- ML Use-Cases Priorization
- Data Availability Check

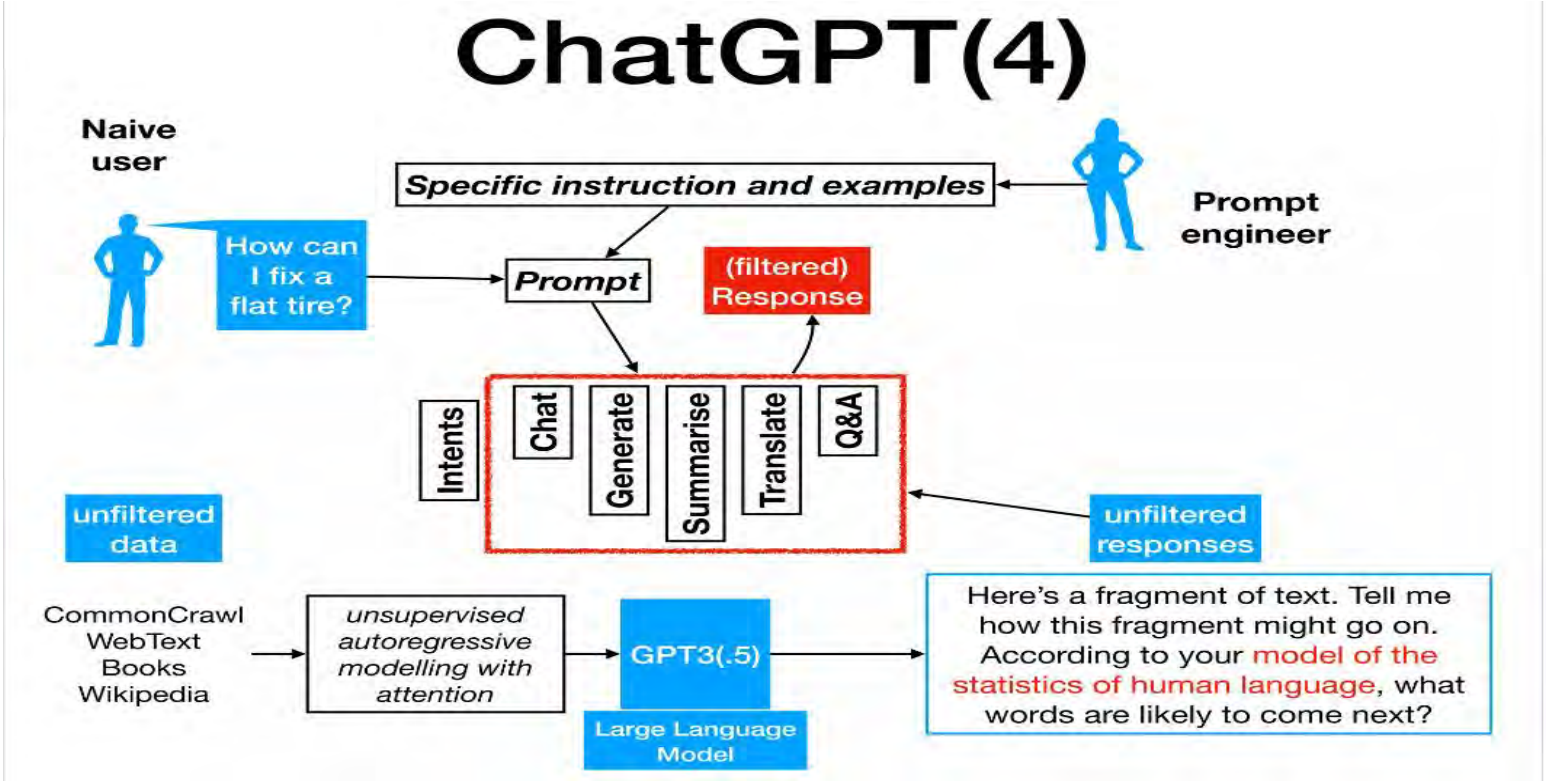
- Data Engineering
- ML Model Engineering
- Model Testing & Validation

- ML Model Deployment
- CI/CD Pipelines
- Monitoring & Triggering

Pytorch Serving – ML models



Large Language Model


ChatGPT(4)
Naive user
How can I fix a flat tire?
Specific instruction and examples
Prompt engineer
Prompt
(filtered) Response
Intents
Chat
Generate
Summarise
Translate
Q&A
unfiltered data
unfiltered responses
CommonCrawl
WebText
Books
Wikipedia
unsupervised autoregressive modelling with attention
GPT3(.5)
Large Language Model
Here's a fragment of text. Tell me how this fragment might go on. According to your model of the statistics of human language, what words are likely to come next?

Evolution of AI Architecture: Traditional ML to Generative AI

Traditional ML

Data Pre-Processing
Cleaning and preparing data for analysis

↓

Feature Engineering
Extracting important features from data

↓

Training & Tuning
Training models on data and adjusting parameters for optimal performance

↓

Deployment & Monitoring
Implementing models in real-world applications and monitoring their performance

Tech Stack for Traditional ML

- **ML Frameworks:** Keras, Theano
- **ML API's & SDK:** IBM Watson
- **Database:** SQL Server, Oracle
- **ML Ops:** Docker, Jenkins

Generative AI

Data Pre-Processing
Cleaning and preparing data for analysis

↓

Prompt Engineering/Fine Tuning
Designing effective prompts to guide AI in generating desired outputs

↓

Foundational/Fine-Tuned LLM
Using foundation and Fine-tuned language models for sophisticated content generation

↓

Deployment & Monitoring
Implementing models in real-world applications and monitoring their performance

Tech Stack for Generative AI

- **Gen AI Orchestration:** Langchain, llamaindex
- **LLM Models:** OpenAI, Anthropic
- **Vector Database:** Pinecone, Weaviate
- **LLM Ops:** Prompt Layer, Helicone

LangChain

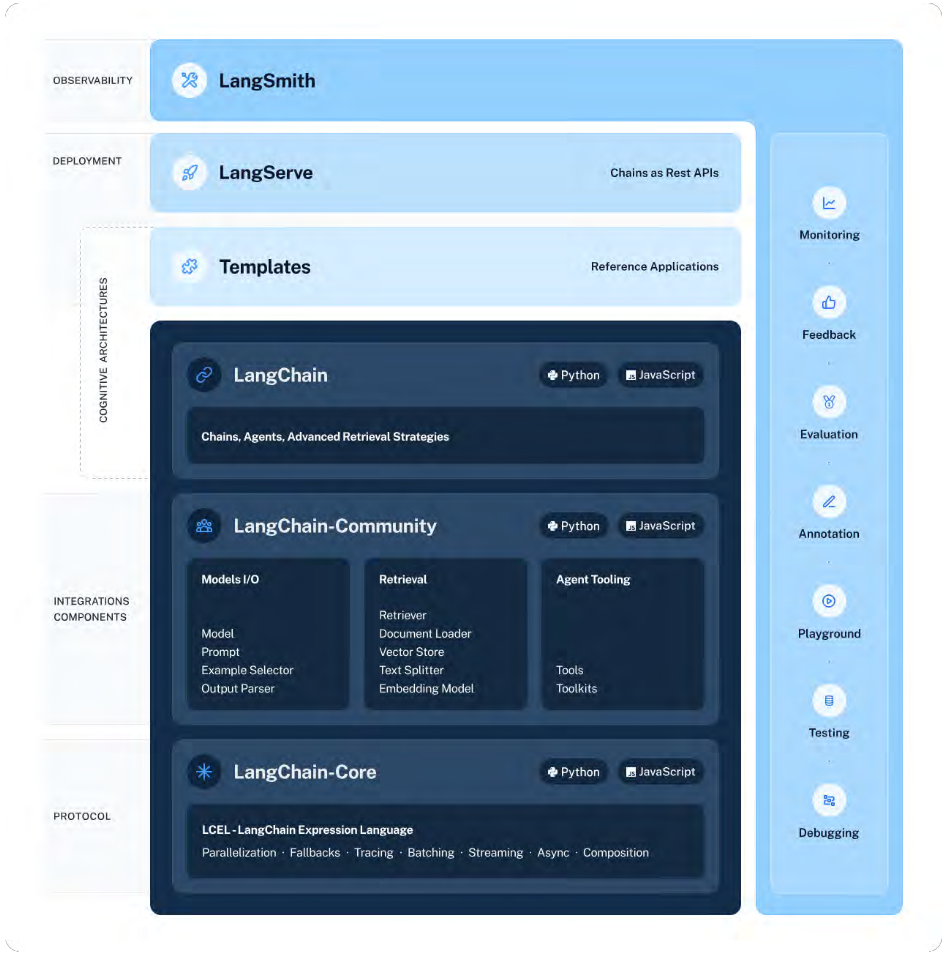
Purpose:

LangChain is designed as a comprehensive toolkit for developers working with large language models (LLMs). It aims to facilitate the creation of applications that are **context-aware** and capable of **reasoning**, thereby enhancing the practical utility of LLMs in various scenarios.

Value Proposition:

LangChain simplifies the transition from prototype to production, offering a suite of tools for debugging, testing, evaluation, and monitoring.

Parts of LangChain Framework



https://www.langchain.com/

•**Libraries:** Available in Python and JavaScript, these libraries offer interfaces and integrations for various components, a runtime for creating chains and agents, and ready-made chain and agent implementations.

•**Templates:** This is a set of deployable reference architectures for diverse tasks, facilitating ease of deployment.

•**LangServe:** A specialized library for converting LangChain chains into a REST API, enhancing accessibility and integration.

•**LangSmith:** A comprehensive developer platform designed for debugging, testing, evaluating, and monitoring chains created with any LLM framework, fully compatible with LangChain.

GenAI Application Development with LangChain

Develop

- **Streamlined Prototyping:** Simplifies the process of creating prototypes with large language models.
- **Context-Aware Systems:** Facilitates the building of applications that understand and utilize context effectively.
- **Integration Support:** Offers tools for integrating various data sources and components.
- **Production Readiness:** Provides resources for debugging, testing, evaluating, and monitoring applications.
- **Collaborative Development:** Encourages and supports collaborative efforts in the developer community.
- **Diverse Applications:** Suitable for a wide range of applications, from chatbots to document analysis.

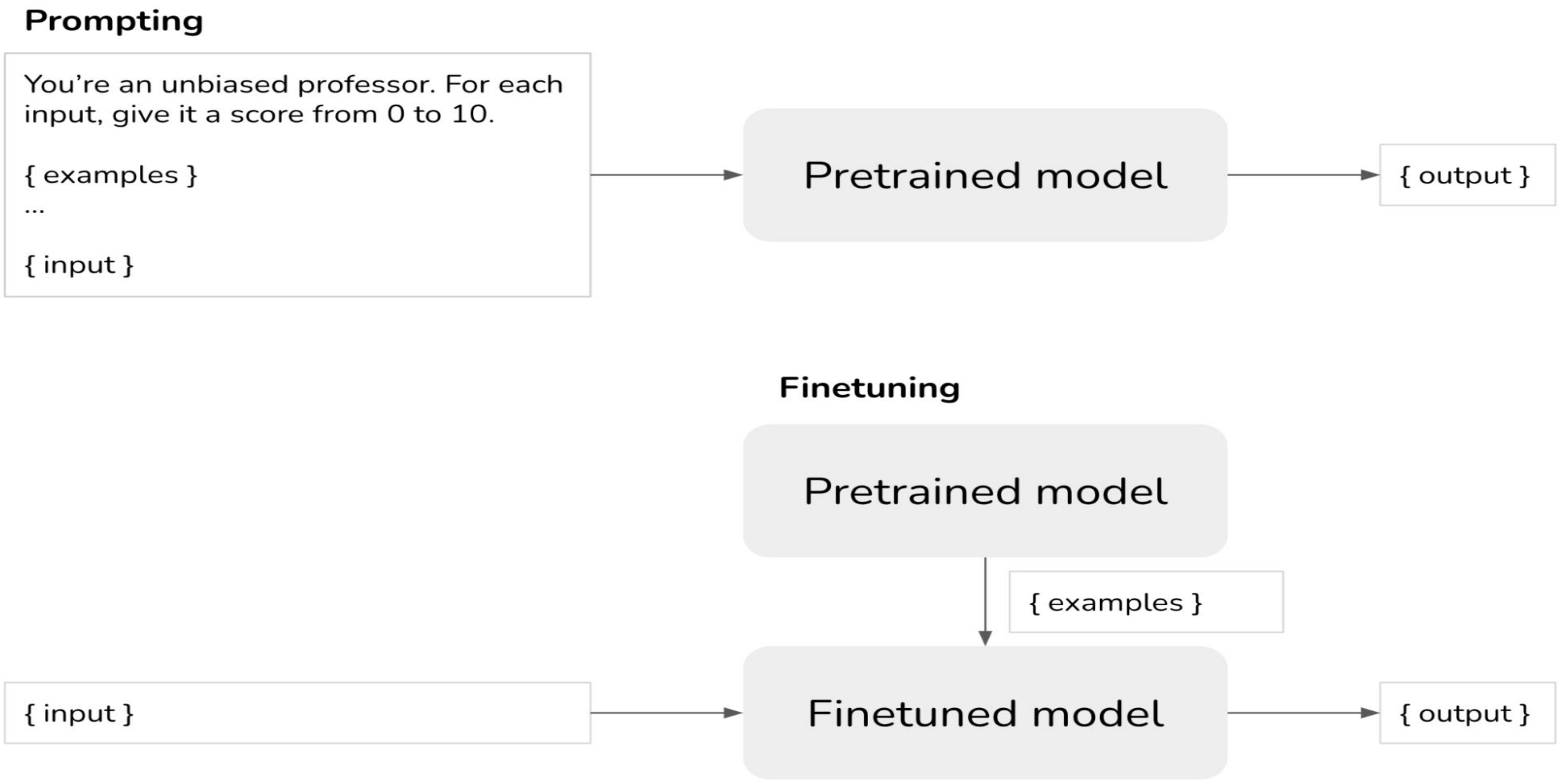
Turn into product

- **Scalability:** Provides tools to scale applications from small prototypes to larger, production-level systems.
- **Robust Testing:** Offers robust testing frameworks to ensure application reliability.
- **Monitoring Tools:** Includes monitoring capabilities to track performance and user interactions.
- **Deployment Ease:** Simplifies the deployment process, making it easier to launch applications.
- **Continuous Improvement:** Supports ongoing development and refinement of applications post-launch.

Deploy

- **LangServe:** A library that allows for the deployment of LangChain chains as REST APIs, making applications easily accessible and integrable.
- **Deployment Templates:** Ready-to-use reference architectures that streamline the deployment process for various tasks.
- **Scalability Tools:** Supports the scaling of applications from development to production level.
- **Ease of Integration:** Ensures seamless integration with existing systems and workflows.
- **Production-Grade Support:** Offers features for ensuring stability and performance in production environments.

Prompting vs. finetuning vs. alternatives



LangChain Demo

Installation and CONFIGURATION

Install LangChain and OpenAI Model:

```
pip install langchain
pip install langchain-openai
```

Set up API KEY for OpenAI:

$$\begin{tikzcd}
\text{OpenAI} \arrow[d] \\
\text{OPENAI\_API\_KEY} \arrow[d] \\
\text{sudo vi /etc/launchd.conf}
\end{tikzcd}$$

```
sudo vi /etc/launchd.conf
export  OPENAI_API_KEY = “KEY”
```

Hello OPENAI LangChain

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI()

r = llm.invoke("how can langsmith help with testing?")
print(r)
```

PROMPT TEMPLATE

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI()


prompt = ChatPromptTemplate.from_messages([
    ("system", "You are world class technical documentation writer."),
    ("user", "{input}")
])

chain = prompt | llm

r = chain.invoke({"input": "how can langsmith help with testing?"})
print(r)
```

OUTPUT PARSER

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI()


prompt = ChatPromptTemplate.from_messages([
    ("system", "You are world class technical documentation writer."),
    ("user", "{input}")
])

output_parser = StrOutputParser()

chain = prompt | llm | output_parser

r = chain.invoke({"input": "how can langsmith help with testing?"})
print(r)
```

THANK YOU