

What Is Generative AI?

Generative AI refers to models that can **generate** new content like:

- Text (e.g., ChatGPT, GPT-4)
- Images (e.g., DALL·E, Stable Diffusion)
- Code (e.g., CodeGen, Codex)
- Audio (e.g., MusicLM)
- Video (e.g., Sora, Runway)

Open Source Frameworks for Generative AI

1. Text Generation

Framework	Description	Language
<input type="checkbox"/> Hugging Face Transformers	The leading library for NLP and generative LLMs	Python
<input type="checkbox"/> GPT-NeoX / GPT-J / GPT-2	Open-source transformer models from EleutherAI	Python
<input type="checkbox"/> LangChain	Framework to build LLM-based agents, chains, and apps	Python, JS
<input type="checkbox"/> LLamaIndex	Connects LLMs to data for retrieval-augmented generation (RAG)	Python
<input type="checkbox"/> Haystack	RAG and NLP workflows with Transformers	Python

2. Image Generation

Framework	Description	Notes
<input type="checkbox"/> Stable Diffusion	Text-to-image model by Stability AI	Hugely popular
<input type="checkbox"/> Diffusers (by Hugging Face)	Toolkit for image/audio/video diffusion models	Plug-and-play
<input type="checkbox"/> ComfyUI	Node-based interface for building diffusion workflows	Visual programming
<input type="checkbox"/> InvokeAI	Stable Diffusion web UI with powerful workflows	Python

□ □ 3. Code Generation

Framework	Description
□ CodeGen by Salesforce	Trained on code for multi-language generation
□ PolyCoder	Lightweight code generation model
□ StarCoder	Open LLM trained on GitHub code by BigCode
□ TabbyML	Open-source GitHub Copilot alternative

□ 4. Multimodal & General AI Agents

Framework	Description	Supports
□ AutoGen (by Microsoft)	Framework for multi-agent LLM apps	Agents, Tools
□ CrewAI	Multi-agent workflow framework	Task delegation
□ LangGraph	Graph-based workflows on LangChain	Loops, branches
□ MetaGPT	Builds software using multi-agent LLMs	End-to-end agent system
□ Semantic Kernel	Orchestration for AI skills and planning	Microsoft ecosystem

□ 5. Audio/Video Generation (Early-Stage)

Framework	Description
□ MusicGen (Meta)	Generate music from text
□ Bark (Suno AI)	Multilingual text-to-speech
□ AnimateDiff	Diffusion-based motion generation
□ OpenVoice	Real-time voice cloning (by MyShell)

□ Bonus: Supporting Tools

Tool	Purpose
Weights & Biases	Experiment tracking and model monitoring
Gradio / Streamlit	Build UIs for generative apps
Docker	Containerize your AI projects
MLflow	Model versioning and deployment
ONNX	Model export for inference acceleration

□ Starter Idea: Build Your Own Chatbot

Using:

- Hugging Face model (e.g., Mistral or GPT-Neo)
- LangChain or AutoGen for agent orchestration
- FAISS for memory
- Streamlit for UI

If you'd like a **starter project** or **repo template**, just say on ChatGpt.com or Gemini.google.com, copilot.microsoft.com:

“Give me a LangChain chatbot starter”

or

“Set up a generative image app with Stable Diffusion”

How to Develop Generative AI using open source Platforms

Developing **generative AI** using **open-source platforms** involves several key steps, from choosing the right models and tools to training or fine-tuning and deploying them. Here's a structured guide to get you started:-

□ 1. Understand What Kind of Generative AI You Want to Build

Generative AI can be applied in various domains:

- **Text:** Chatbots, story generation, code generation (e.g., GPT, LLaMA, Mistral)
 - **Images:** Art, photo generation (e.g., Stable Diffusion, DALL·E)
 - **Audio:** Music, speech synthesis (e.g., Bark, Tortoise TTS)
 - **Video:** Video generation from text (more advanced, e.g., Sora by OpenAI)
 - **Multimodal:** Combines text, image, etc. (e.g., LLaVA, CLIP)
-

□ 2. Choose Your Open-Source Tools & Frameworks

Popular Model Libraries:

- □ **Hugging Face Transformers** – Text & vision models (GPT, BERT, T5, etc.)
- □ **Diffusers (Hugging Face)** – Image and audio generation
- □ **PyTorch** – Most generative models use this framework
- □ **TensorFlow** – Also used in some cases, especially in production
- □ **LLaMA, Mistral, Mixtral** – Open LLMs
- □ **LangChain** – For building LLM-powered applications with memory, tools, etc.

Datasets:

- □ **Hugging Face Datasets**
 - □ **The Pile, LAION-5B, Common Crawl, COCO**
-

□ 3. Build or Fine-tune a Model

Options:

- **Use a pre-trained model** (most common)
- **Fine-tune an existing model** on your data
- **Train from scratch** (only for research or very large-scale projects)

Tools for Training/Fine-tuning:

- Hugging Face's `Trainer` API
- LoRA / QLoRA (for lightweight fine-tuning)
- DeepSpeed, FSDP (for large-scale training)
- PEFT (parameter-efficient fine-tuning)

Example for fine-tuning GPT-2 using Hugging Face:

Python:-

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel, Trainer,
TrainingArguments

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Prepare your dataset
train_data = tokenizer(["your text data here"], return_tensors='pt',
truncation=True, padding=True)

training_args = TrainingArguments(
    output_dir="./gpt2-finetuned",
    per_device_train_batch_size=2,
    num_train_epochs=3,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
)

trainer.train()
```

□ 4. Deploy the Model

Options:

- **Locally** via Gradio or Streamlit
- **Cloud:** AWS, GCP, or Hugging Face Spaces
- **Docker:** Containerize your app

Gradio example:

Python:-

```
import gradio as gr

def generate(prompt):
    inputs = tokenizer(prompt, return_tensors="pt")
    output = model.generate(**inputs, max_new_tokens=50)
    return tokenizer.decode(output[0])

gr.Interface(fn=generate, inputs="text", outputs="text").launch()
```

□ 5. Considerations

- **Compute:** GPU/TPU is needed for training; inference can be CPU/GPU
 - **Model Size:** Smaller models for local use (e.g., TinyLLaMA, Phi-2)
 - **Ethics:** Ensure safe and unbiased generation
 - **License:** Check if the model is commercially usable (many are not!)
-

□ Suggested Open-Source Tools by Domain

Task	Tools / Models
Text	GPT-2, LLaMA, Mistral, BLOOM, Falcon, Phi-2
Image	Stable Diffusion, Kandinsky, VQGAN+CLIP
Audio	Bark, MusicGen, Tortoise TTS
Video	ModelScope (basic), Sora (not open-source yet)
Multimodal	LLaVA, MiniGPT-4, CLIP

□ Want a Real-World Project Idea?

- Chatbot trained on your company's documents (use LangChain + LLaMA)
 - Image generation app using Stable Diffusion with a web UI
 - Music generator fine-tuned on a genre using MusicGen
-

LLM-powered applications

LLM-powered applications are tools, platforms, or products that use **Large Language Models** (LLMs) like GPT, LLaMA, or Mistral to understand and generate human-like language. These applications span a wide range of industries and use cases, from chatbots to code generation and even creative writing or tutoring systems.

□ Key Components of an LLM-Powered Application

1. **LLM Backbone**
 - Open-source: LLaMA, Mistral, Falcon, GPT-J, BLOOM, etc.
 - API-based: OpenAI (GPT-4), Anthropic (Claude), Cohere, etc.
 2. **Interface Layer**
 - How users interact with the model: Chat UI, API, voice, etc.
 3. **Prompt Engineering / Templates**
 - Carefully crafted inputs to guide the model's behavior and output.
 4. **Context Management**
 - Memory, conversation history, or document embedding using vector databases.
 5. **Orchestration Framework**
 - Tools like **LangChain**, **LlamaIndex**, or **Haystack** for chaining tasks, document retrieval, or tool use.
 6. **Retrieval-Augmented Generation (RAG)**
 - Augment the LLM with external knowledge (e.g., private docs, websites).
 - Uses vector stores like **FAISS**, **Pinecone**, **Weaviate**, **Chroma**.
-

□ Example Architectures

1. RAG-Based Q&A Bot

- **Use Case:** Company FAQ bot using internal documents.
- **Tech Stack:**
 - LLM: Mistral or GPT-4
 - Embedding: sentence-transformers/all-MiniLM-L6-v2
 - Vector DB: FAISS or Chroma
 - Retrieval: LangChain retriever
 - Frontend: Gradio or React

2. Autonomous Agent

- **Use Case:** Automate workflows (e.g. AutoGPT, BabyAGI)
- **Core Idea:** LLM acts as a planner and executor
- **Tools:** LangChain Agents + Tools (e.g. web search, file tools)

3. Coding Assistant

- **Use Case:** Auto-complete and refactor code
- **Tech Stack:**
 - LLM: StarCoder, Code LLaMA
 - Embedding search for context
 - VSCode or Jupyter plugin as frontend

Tooling to Build LLM Applications

Tool	Purpose
<input type="checkbox"/> LangChain	Chain LLM prompts, tools, and memory
<input type="checkbox"/> LlamaIndex	Index and query document sources
<input type="checkbox"/> Haystack	NLP pipelines with LLM + retrievers
<input type="checkbox"/> FAISS / Chroma	Vector search and document similarity
<input type="checkbox"/> Pinecone / Weaviate	Cloud vector DBs with scalable RAG support
<input type="checkbox"/> Gradio / Streamlit	Build interactive UIs easily
<input type="checkbox"/> FastAPI	Backend server for LLM inference

Simple Example: Q&A over Custom Docs (LangChain + LLaMA)

Python:-

```
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.llms import HuggingFaceHub
from langchain.chains import RetrievalQA
from langchain.document_loaders import TextLoader

# Load documents and create vector store
loader = TextLoader("docs.txt")
docs = loader.load()
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
db = FAISS.from_documents(docs, embeddings)

# Connect to LLM
llm = HuggingFaceHub(repo_id="mistralai/Mistral-7B-Instruct-v0.1")

# Create retrieval-based QA chain
qa = RetrievalQA.from_chain_type(llm=llm, retriever=db.as_retriever())
print(qa.run("What is the company refund policy?"))
```

□ Popular LLM Application Ideas

Use Case	Description
Chatbots	Customer support, healthcare assistants, tutors
Code Assistants	Auto-complete, code explanation
Document Q&A	Ask questions over PDFs, websites, manuals
Email Generators	Summarize or write personalized replies
Workflow Agents	Plan tasks, search online, generate reports
Knowledge Base	Intranet-powered assistant for teams