Group 3:

GPT Models

Members

- Mithunan
- Thibakar
- Abinaya

OpenAI Coding / Software Models

Evaluation

1) Early OpenAI Code Models (Pre-2023) (Codex & Code-DAVinci)

These were the first coding-focused models released by OpenAI.

- Code-davinci-002 – OpenAI's most capable early Codex-style model
- Code-davinci-001 – Older Davinci variant for code
- Code-cushman-002 – Faster/cheaper coding variant
- Code-cushman-001 – Earlier small coding model

Note: All of the above were **deprecated in March 2023** and replaced by newer general models, with **GPT-4o** recommended as replacement.

2) GPT-3 / GPT-3.5 Era (Transitional Coding Support)

These models were *general LLMs* but widely used for coding before dedicated successors emerged.

- **GPT-3.5-turbo-** variants — general model family used for code generation (Contextually used because newer Codex deprecated)
- **GPT-4 / GPT-4.0 era** — early OpenAI flagship that also generated code before dedicated Codex revival

Note: GPT-4 family was not "Codex-branded" but widely used for code tasks pre-Codex revival.

3) Codex Revival and Native OpenAI Coding Agents (2024–2025)

In 2025, OpenAI re-introduced **Codex as a first-class coding agent** — distinct from general LLMs — with models optimized for developer workflows.

Codex Core Models

Model	Release / Era	What & Why	Notes
codex-1	2025 (Codex launch)	Core coding model powering Codex agent	Underlies Codex product in early 2025
codex-mini-latest	2025	Cost-efficient coding model for CLI / fast workflows	Available via API & Codex CLI

4) GPT-5-Series Coding Specializations (2025)

As general LLMs evolved, OpenAI produced **specialized GPT-5 coding variants** optimized for real software engineering.

GPT-5-Series Coding Models

Model	Release	Purpose
gpt-5-codex	Sep 15, 2025	GPT-5 variant optimized specifically for coding tasks in Codex environments (code refactor, tests, reviews)
gpt-5-codex-mini	2025	Smaller, cost-effective variant of GPT-5-Codex

5) GPT-5.1 Coding Models (Latest — Late 2025)

Latest and most capable coding-oriented models from OpenAI.

GPT-5.1 Codex Models

Model	Release	What it Does
Gpt-5.1-codex	Nov 13, 2025	Successor to GPT-5-Codex : optimized for **longer, more complex software engineering tasks**
Gpt-5.1-codex-mini	Nov 2025	Smaller, cheaper version for iterative coding / fast tasks
Gpt-5.1-codex-max	Nov 19, 2025	**Most advanced coding model** — designed for sustained multi-step coding workflows (refactors, tests, etc.) and integration with IDE/CLI/Cloud

TimeLine:

Legacy / Deprecated

- code-davinci-002, code-davinci-001
- code-cushman-002, code-cushman-001
  (Deprecated March 2023) OpenAI Platform

General LLM Support (Pre-2025)

- GPT-3 / 3.5 Turbo (widely used for code)
- GPT-4 family (strong coding support, not Codex branded)

2025 Codex Revitalization

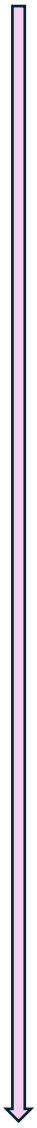
- codex-1
- codex-mini-latest

GPT-5 Coding Specializations

- gpt-5-codex
- gpt-5-codex-mini

GPT-5.1 Modern Coding Models

- gpt-5.1-codex
- gpt-5.1-codex-mini
- gpt-5.1-codex-max



Speech and Audio domains

Speech Recognition (Audio → Text)

1) Original Whisper Speech Recognition Models (Open-Source)

Whisper (initial)

- open-source Whisper model
- Transcribes spoken audio to text, supports many languages.

Whisper Large V2 & Whisper Large V3

- These models were widely used *before* API-level GPT-transcribe models replaced them as the new generation.

2) GPT-4o-Based Transcription Models (Next Generation)

gpt-4o-transcribe

- Official speech-to-text transcription model powered by GPT-4o architecture.
- Converts spoken audio into text with improved accuracy vs older Whisper.

gpt-4o-mini-transcribe

- Smaller, cheaper transcription model powered by GPT-4o mini.
- Fast, cost-efficient speech-to-text with lower error rates vs Whisper

gpt-4o-transcribe-diarize

- Adds speaker diarization (speaker labeling + timestamps).
- Transcribes audio and marks which speaker spoke when.

Text-to-Speech (Text → Audio)

tts-1

- Basic text-to-speech model on the OpenAI Audio API.
- Takes text input and outputs spoken audio via API.

tts-1-hd

- Higher quality, more natural speech output.
- Produces higher-fidelity spoken audio output.

gpt-4o-mini-tts

- Text-to-speech model built on GPT-4o mini
- Generates natural speech from text with controllable style/intonation.

Real-Time Audio Interaction

gpt-realtime

- Latest real-time multimodal model that supports audio input and audio output in one model.
- Performs native speech-to-speech, audio understanding and generation in realtime with natural language capability

gpt-4o-audio-preview

- A preview model enabling GPT-4o to accept audio input and produce audio/text output.
- Adds audio I/O to the Chat Completions API before full realtime.

Year	Model	Purpose
2022	Whisper (initial)	Speech-to-text transcription (open-source)
2022 Dec	Whisper Large V2	Higher-accuracy transcription
2023 Nov	Whisper Large V3	Improved transcription quality
2024	gpt-4o-audio-preview	Early audio I/O preview in Chat Completions API
2024-2025	gpt-realtime	Real-time speech-to-speech + multimodal interaction
2025	gpt-4o-transcribe	Next-gen high-accuracy speech-to-text
2025	gpt-4o-mini-transcribe	Lightweight transcription
2025	gpt-4o-transcribe-diarize	Transcription + diarization
2025	tts-1	API text-to-speech
2025	tts-1-hd	High-quality TTS
2025	gpt-4o-mini-tts	GPT-4o-based advanced TTS

Pricing

Model	Free?	Price	Notes
Whisper (open-source)	Yes	(free offline) API: $0.006/min	Speech→text, supports Approximatly99 languages (API charges $0.006/min)
gpt-4o-transcribe	NO	$2.50/1M input $10.00/1M output + $0.006/min	Speech→text via API; GPT-4o-based transcription (no open-source version).
gpt-4o-transcribe-diarize	No	same as transcribe above	Adds speaker labeling (diarization).
gpt-4o-mini-transcribe	No	$1.25/1M input $5.00/1M output + $0.003/min	Lower-cost version of transcription.
tts-1 (API)	No	$15.00 per 1M output characters	Standard text-to-speech (API).
tts-1-hd (API)	No	$30.00 per 1M output characters	High-quality TTS.
gpt-4o-mini-tts	No	$0.60/1M input + $0.015/min output	GPT-4o-based TTS with controllable style.
gpt-realtime (text)	No	$4.00/1M input, $0.40/1M cache, $16.00/1M output	Real-time low-latency text API.
gpt-realtime (audio)	No	$32.00/1M audio input, $64.00/1M output	Real-time speech↔speech.
gpt-realtime-mini (text)	No	$0.60/1M input, $0.06/1M cache, $2.40/1M output	Lightweight real-time text.
gpt-realtime-mini (audio)	No	$10.00/1M audio input, $20.00/1M output	Lightweight audio S2S.

gpt-4o-audio-preview (2024)	No (preview)	$2.50–$10.00/1M text + $40–$80/1M audio	Early multimodal preview (transcript + audio output). Higher rates (shown) apply to text/audio tokens respectively.

Model	Free?	Price (Prompt/ Completion)	Notes
code-davinci-002, code-davinci-001 code-cushman-002, code-cushman-001 (Codex)	No (paid, deprecated)	Replaced by GPT-4o (no new API access)	Discontinued (Mar 2023)
gpt-3.5-turbo	No	$0.002 per 1K input tokens / ~$0.004 per 1K output	Widely used for code (cheap, 16K max context for turbo-16k)
GPT-4 (8K)	No	$0.03/$0.06 per 1K (input/output)	General-purpose model (8K context); strong coder
GPT-4 (32K)	No	$0.06/$0.12 per 1K (input/output) (prior rates)	Higher context (32K tokens) for long docs
codex-1 (2025)	No	GPT-4.1 series pricing	Core of new Codex product (2025) high-capacity code model
codex-mini-latest (2025)	No	$1.50 / $6.00 per 1M (prompt/output)	Cost-efficient CLI-focused model
gpt-5.1-codex-max	No	$1.25 / $10.00 per 1M	Top-end coding model (Nov 2025)
gpt-5.1-codex	No	$1.25 / $10.00 per 1M	Successor to GPT-5-Codex
gpt-5.1-codex-mini	No	$0.25 / $2.00 per 1M	Smaller, cheaper Codex (iterative coding)

gpt-5-codex	No	$1.25 / $10.00 per 1M	GPT-5 variant for coding (Sept 2025 release).
gpt-5-codex-mini	No	(similar ~$0.25/$2.00 per 1M)	

OPENAI MODEL ECOSYSTEM — TEXT → IMAGE & TEXT → VIDEO

OpenAI's model ecosystem provides specialized solutions for text-to-image and text-to-video generation. These models have been optimized for:

- Semantic understanding of prompts
- High fidelity visual output
- Multi-modal integration (text + image/video + audio)
- Production readiness (commercial or research)

This document provides a comprehensive, detailed technical reference, covering capabilities, API usage, pricing, limitations, and practical best practices.

TEXT → IMAGE MODELS

GPT-Image-1

Domain

- Primary model for image generation and image editing in OpenAI's ecosystem.
- Supports high-fidelity image generation from textual prompts.

Proficiency

- Generates images from detailed natural language prompts.
- Excels in instruction following, scene composition, and text rendering within images.
- Supports inpainting and background manipulation.

Key Features

- Multi-aspect ratio image generation (e.g., 1024x1024, 1792x1024).
- Image editing via input + mask.
- Supports transparent background parameter.
- High alignment with prompt instructions.
- Produces photorealistic or stylized artistic outputs.

Technical Details

- Transformer-based multimodal architecture.
- Combines diffusion and GPT-based semantic understanding.
- Optimized for instruction fidelity rather than purely generative artistic variance.

API Information

Endpoints:

- POST /v1/images/generations
- POST /v1/images/edits

Parameters:

- model: "gpt-image-1"
- prompt: Textual instruction
- image: Optional base image for edits
- mask: Optional mask for inpainting
- size: "1024x1024", "1792x1024", etc.
- background: "transparent", "opaque", "auto"

Example Request (Curl):

```
curl https://api.openai.com/v1/images/generations \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model":"gpt-image-1",
    "prompt":"Cinematic wide-angle photo of a red fox on a misty log, golden hour lighting",
    "size":"1024x1024"
  }'
```

Pricing

- Based on output image size and token usage.
- Example pricing (representative, confirm current at OpenAI docs):
    - o Input text tokens: ~$5 / 1M tokens
    - o Input image tokens: ~$10 / 1M tokens
    - o Output image tokens: ~$40 / 1M tokens

Limitations

- Small text inside images may be distorted.
- Complex symbolic instructions may fail.
- High-resolution images increase cost substantially.

Recommended Use Cases

- Product visuals and marketing images
- Storyboarding and educational diagrams
- Concept art and creative ideation

GPT-Image-1-Mini

Domain

- Cost-optimized and faster variant of GPT-Image-1.

Proficiency

- Lower-cost, high-speed generation.
- Suitable for high-volume outputs and prototypes.

Key Features

- Fast generation for large-scale testing.
- Supports same API as GPT-Image-1.

Limitations

- Reduced detail and fidelity.
- Not recommended for production-grade commercial visuals.

Recommended Use Cases

- Thumbnails and social media content
- Rapid iteration of visual concepts

DALL·E 2 (Historical)

Proficiency

- Legacy diffusion-based image generator.
- Capable of inpainting and style-based image generation.

Limitations

- Outperformed by GPT-Image-1 in instruction following.
- Lower resolution and weaker text handling.

DALL·E 3 (Historical)

Proficiency

- Improved over DALL·E 2 with richer prompt understanding.
- Generates more coherent artistic outputs.

Limitations

- Superseded by GPT-Image-1 for most production uses.

GPT-4o Image Generation

Domain

- Image generation within multimodal conversational workflows.

Proficiency

- Interactive generation with iterative refinement in chat.
- Useful for brainstorming or ideation tasks.

Limitations

- Cannot be directly invoked as a standalone /images API model.
- Limited control over resolution or aspect ratio compared to GPT-Image-1.

TEXT → VIDEO MODELS

Sora (Original)

Domain

- Early text-to-video research model.

Proficiency

- Generates short videos (10–60s).
- Demonstrates basic object motion and scene transitions.

Limitations

- Visual artifacts common.
- Limited API accessibility.

Sora-2

Proficiency

- State-of-the-art text → video model with synchronized audio.
- Improved temporal consistency and object fidelity.

Key Features

- Video generation from text or image input.
- Video extension and editing.
- Synchronized audio tracks.
- Style, aspect ratio, FPS, and duration configuration.

API Information

Primary Endpoints:

- POST /v1/videos
- POST /v1/videos/edits

Parameters:

- model = "sora-2"
- prompt
- duration_seconds
- fps
- aspect
- audio_options

Example Request (pseudo-curl):

```
curl https://api.openai.com/v1/videos \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model":"sora-2",
    "prompt":"10s cinematic shot: small wooden boat at sunrise, misty lake, gentle camera pan",
    "duration_seconds":10,
    "fps":24,
    "aspect":"landscape"
  }'
```

Pricing

- Charged per second; example: ~$0.10/sec (standard Sora-2).
- Cost scales with duration and complexity.

Limitations

- Artifacts are possible in complex or crowded scenes.
- Audio quality may need post-processing.

Recommended Use Cases

- Content creation and marketing videos
- Educational or storytelling clips
- Rapid prototyping of cinematic shots

Sora-2-Pro

Proficiency

- Premium cinematic video generation model.
- Highest temporal and visual fidelity with advanced audio.

Key Features

- Higher resolution and frame count.
- Advanced camera and motion controls.
- Professional-grade output suitable for studio production.

API Information

- Same as Sora-2, but with model="sora-2-pro".

Pricing

- ~3× the standard Sora-2 rate per second.

Limitations

- Expensive for long-duration or high-resolution content.
- Requires careful prompt engineering.

Recommended Use Cases

- Film previsualization
- Advertising campaigns
- Professional animation and cinematic projects

API INTEGRATION & BEST PRACTICES

- Image and video generation follow a unified API pattern (prompt → model → output).
- Iterative workflows: generate low-res / mini versions first, then upscale.
- For Sora models, use seed parameter for reproducible results.
- Image edits: use masks to precisely control modifications.
- Video: define camera motion and duration in prompt to reduce artifacts.

PRICING & CREDIT USAGE

Model	Pricing Type	Example Rate
GPT-Image-1	Per image / per token	Input tokens $5/1M, Output image $40/1M
GPT-Image-1-Mini	Per image / per token	Approx. 50% cheaper
DALL·E 3	Per image	Low/Med/High: $0.01/$0.04/$0.17
Sora-2	Per second	~$0.10/sec
Sora-2-Pro	Per second	~3× Sora-2

LIMITATIONS & SAFETY CONSIDERATIONS

- Models enforce content policy restrictions.
- Avoid generating illegal or unsafe content.
- Deep-fake restrictions apply to identifiable individuals.
- Commercial use may require IP clearance for copyrighted content.
- Video models may introduce motion or audio artifacts in edge cases.

COMPARISON TABLES

Text → Image Model Comparison

Model	Strengths	Limitations	Use Case
GPT-Image-1	High fidelity, instruction-aligned	Higher cost	Production, concept art
GPT-Image-1-Mini	Low cost, fast	Lower detail	Prototype, bulk images
DALL·E 3	Creative art	Legacy	Art, stylized content
DALL·E 2	Stable	Outdated	Education, testing
GPT-4o	Interactive	Less control	Chat-based workflows

Text → Video Model Comparison

Model	Strengths	Limitations	Use Case
Sora	Early text→video	Artifacts, short duration	Research, demos
Sora-2	Audio/video, coherent motion	Medium cost	Content creation, prototypes
Sora-2-Pro	Cinematic, high fidelity	High cost	Studio, professional media

PROMPT ENGINEERING & TESTING GUIDELINES

- Image prompts: include lens, lighting, composition, style.
- Video prompts: define shot motion, duration, camera angle, audio cues.
- Test across multiple models for quality and cost evaluation.
- Use mini models for rapid iterations.
- Iteratively refine prompts and optionally use masking or reference images.

OpenAI's visual generation models provide a full spectrum of image and video production capabilities, from high-fidelity GPT-Image outputs to professional-grade Sora-2-Pro cinematic videos. Proper model selection, API usage, prompt engineering, and cost management are critical for achieving optimal results.

Natural Language Understanding (NLU)

What it means ?

- NLU is the model's ability to interpret human language understanding meaning, context, intent, tone, and relationships between words.

How GPT achieves NLU

1. **Transformer architecture**

GPT uses the Transformer neural network design to process text. Instead of reading a sentence word by word (like older RNNs), a Transformer can look at all words in parallel. It stacks many identical "Transformer blocks" on top of each other, where each block contains attention and feed-forward sub-layers. In GPT's case, the model uses only the decoder half of a Transformer (an auto-regressive design) which is ideal for generating text. This architecture allows GPT to capture complex patterns: for example, it can maintain context over long sentences or documents without forgetting earlier words.

2. **Self-attention mechanism → identifies relationships between words**

The key innovation in Transformers is self-attention. In simple terms, every word (token) "looks at" all the other words in the sentence and computes attention scores. This tells the model which words are most relevant to each other. You can imagine each word asking "how much should I focus on this other word to understand my meaning?". For example, in "She saw a bat by the tree," the word "bat" can use attention to check nearby words: if "ball" or "hit" appeared before, it might pick the baseball meaning, but with "cave" or "night", it'd pick the animal meaning. GPT actually uses multi-head self attention, meaning it does this focus-calculation in several different ways at once, capturing different patterns in the data. Altogether, self-attention lets GPT weigh context dynamically: important related words get more influence on each token's understanding.

3. Pretraining on diverse datasets

Before doing any specific task, GPT is “pre-trained” on massive amounts of raw text from diverse sources (web pages, books, articles, etc.). During this phase, the model’s only job is to predict the next word in each sequence of text. This simple objective forces it to learn grammar, vocabulary, and factual knowledge. For example, by reading billions of sentences, GPT learns that “Paris is the capital of ?” is likely to end with “France.” The diversity of the data means GPT sees many topics and writing styles, so it acquires broad knowledge. After pretraining, GPT has already internalized a great deal of language structure, so it can be fine-tuned or prompted for specific tasks with little extra training.

4. Token-level contextual understanding

A crucial result of the above components is that GPT builds contextual embeddings for each token. In other words, the meaning of a word in GPT depends on its surrounding words, not just a fixed dictionary definition. Each token’s vector representation is computed by combining information from nearby tokens via the attention layers. For example, the word “bank” will have different internal representations in “river bank” vs. “bank account,” because self-attention signals which context is relevant. In practice, after all Transformer layers, each token’s final hidden vector encodes the token in context. GPT can thus interpret words and phrases in light of their sentence, allowing it to handle ambiguity and understand nuance.

Capability	Explanation	Example
Intent Recognition	Understanding what the user wants	“Book me a flight” → Action intent
Entity Extraction	Identify names, places, products	“Apple launched Vision Pro in 2024”
Sentiment Analysis	Recognize emotions	“I’m disappointed” → negative
Context Tracking	Maintains multi-turn conversation	Remembers earlier parts of chat

Why it matters

- NLU is the foundation for chatbots, virtual assistants, sentiment systems, classifiers, search engines, and summarization tools.

Natural Language Generation (NLG)

What it means?

- NLG is GPT's ability to produce human-like, coherent, context-aware language.

How GPT performs NLG

- Predicts next tokens with probability distributions
- Uses contextual embeddings
- Balances coherence + creativity with decoding methods (sampling, temperature, top-p)

Key Capabilities

Capability	Description
Text generation	Stories, answers, essays
Summarization	Condense long text
Paraphrasing	Rewrite in different styles
Translation	Multi-language translation
Code generation	Write & fix code

Example:

Input: "Explain transformers in simple words."

Output: GPT generates a structured, easy explanation.

NLU / NLG MODEL COMPARISON TABLE

Model	Strengths	Limitations	Best Use Case
GPT-5.1	Best reasoning, highest accuracy	Higher cost	Research, enterprise AI
GPT-5	Strong reasoning	Slightly less optimized	Advanced chatbots
GPT-4.5	High-quality text generation	Slower than GPT-4o	Content & analysis
GPT-4.1	Multimodal understanding	Higher latency	Vision + text tasks
GPT-4o	Fast, multimodal, low latency	Less deep reasoning	Real-time assistants

Embeddings

What embeddings are:

- Embeddings are numerical vector representations of text (words, sentences, documents).
- They help the model measure semantic similarity.

Example:

- "car" and "vehicle" will have closer vectors
- "car" and "banana" will be far apart

What GPT uses embeddings for

1. **Semantic Search → more accurate than keyword search**

- Embeddings convert queries and documents into high-dimensional vectors capturing their meaning, not just matching exact words. Semantic search then finds documents whose vectors are close to the query vector, so it returns conceptually relevant results.

- This handles synonyms and context. For example, a semantic search for "how to cool a room without AC" would retrieve advice on fans or ventilation – understanding the cooling intent – rather than simply matching the words "cool," "room," and "AC".

2. **Recommendation Systems - Embeddings represent users and items (movies, products, etc.) as vectors capturing**

- their features and preferences. The system then recommends items whose vectors are close to the user's vector, indicating similar content or interests.

- This goes beyond keyword tags: if a user likes an action–comedy movie, embedding based recommendations will find other action-comedy films even if their descriptions use different words

3. Clustering & Classification

- **Clustering:** Embeddings transform data (text, images, etc.) into vectors, allowing standard clustering algorithms (like k-means) to group similar items. For instance, product reviews can be clustered by topic (e.g. "positive" vs "negative" comments) by embedding each review and grouping neighbors

- **Classification:** Embeddings can serve as features for classifiers. For text classification, one can embed each document and then use a model (or nearest-neighbor in embedding space) to assign labels. Even without retraining, "zero-shot" classification is possible: a text is labeled by comparing its embedding to prototype embeddings of each class

4. Context Retrieval (RAG)

- RAG augments a language model by first retrieving relevant context from an external knowledge base using embeddings. The query is embedded and matched against document embeddings. The top relevant snippets are then given to the LLM as extra input (context), allowing it to generate more accurate, up-to-date answers.

- This means the model can answer questions using current or domain-specific data it wasn't explicitly trained on, reducing "hallucinations."

5. Similarity Matching (resumes, products, documents)

- Similarity Search: Embeddings let you find items that "match" by meaning. For example, to match resumes to a job description, both the resume text and job posting are embedded; then the system finds the resumes whose vectors are closest to the job's vector.

- This applies to products (find similar products), documents (find duplicate or related articles), or any items: just compare embedding distances.

6. Knowledge base augmentation

- Embeddings can incorporate structured knowledge into a model. For example, a knowledge graph of entities can be embedded so that related concepts are close in vector space. These knowledge-rich embeddings help the system reason about facts.

- New methods (like KBLAM) inject knowledge base entries directly into a model's context by converting facts into embedding tokens, so the model has access to up-todate domain-specific information during generation. Why embeddings matter They allow GPT models to work with large external data sources and perform advanced tasks that require understanding how similar two pieces of text are.

Tools in the GPT/OpenAI Ecosystem

- Tools expand GPT's native abilities. Tools are external functions, APIs, or modules that the model can call.

1. Function Calling

- GPT can call a predefined function with structured JSON output.

  Example:

  User : "Book me a hotel in Tokyo."

  GPT → Calls function: search_hotels(city = "Tokyo")

2. Code Interpreter (Python)

Allows:

- Data analysis
- Visualizations
- File conversion
- Running Python programs inside a sandbox

3. Image Generation Tools (DALL·E)

GPT can:

- Generate images
- Edit images
- Create variations

4. Retrieval Tools

Connect GPT to external knowledge bases:

- Company documents
- PDFs
- Databases

The model fetches relevant data using embeddings.

5. Browsing Tool

- Allows live internet search (optional depending on plan).

EMBEDDINGS & TOOLS COMPARISON

Component	Purpose	Example
Text Embeddings	Semantic similarity	Resume matching
Vector DB	Fast retrieval	Knowledge base search
RAG	Grounded responses	Enterprise chatbot
Function Calling	External actions	Booking systems
Code Interpreter	Computation	Data analysis

How These Components Connect Together

Component	**Role**	**Example**
NLU	Understand input	"Summarize this PDF"
Embeddings	Retrieve related knowledge	Convert PDF text → vectors
Tools	Access external functions or files	Code Interpreter reads the PDF
NLG	Generate final answer	"Here is a summary…"