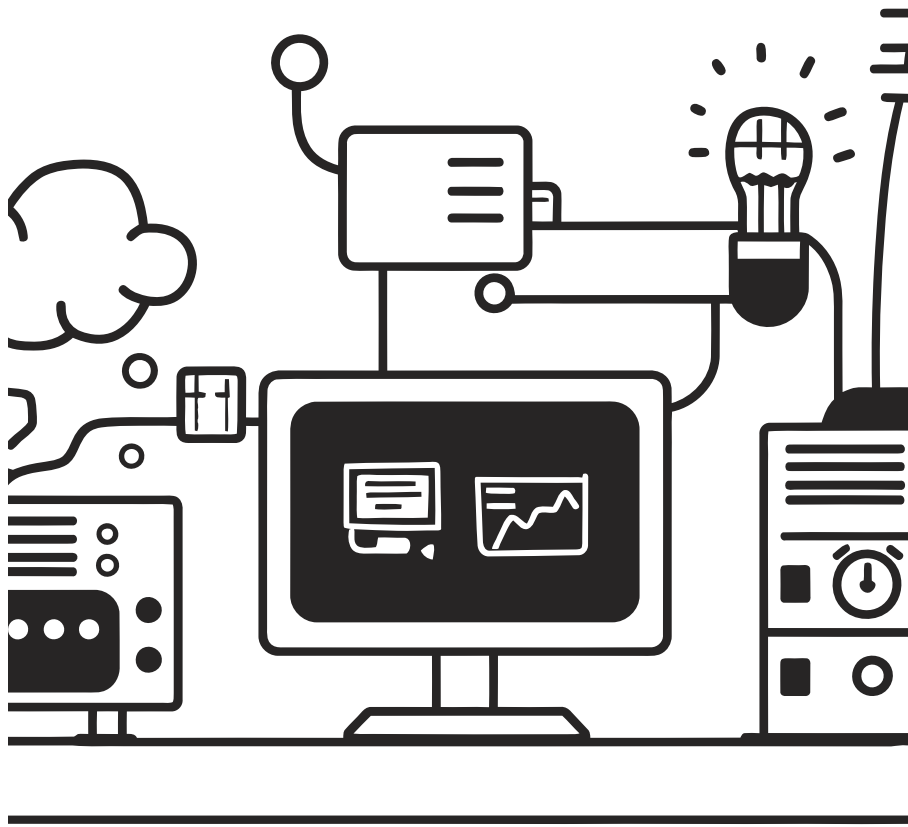
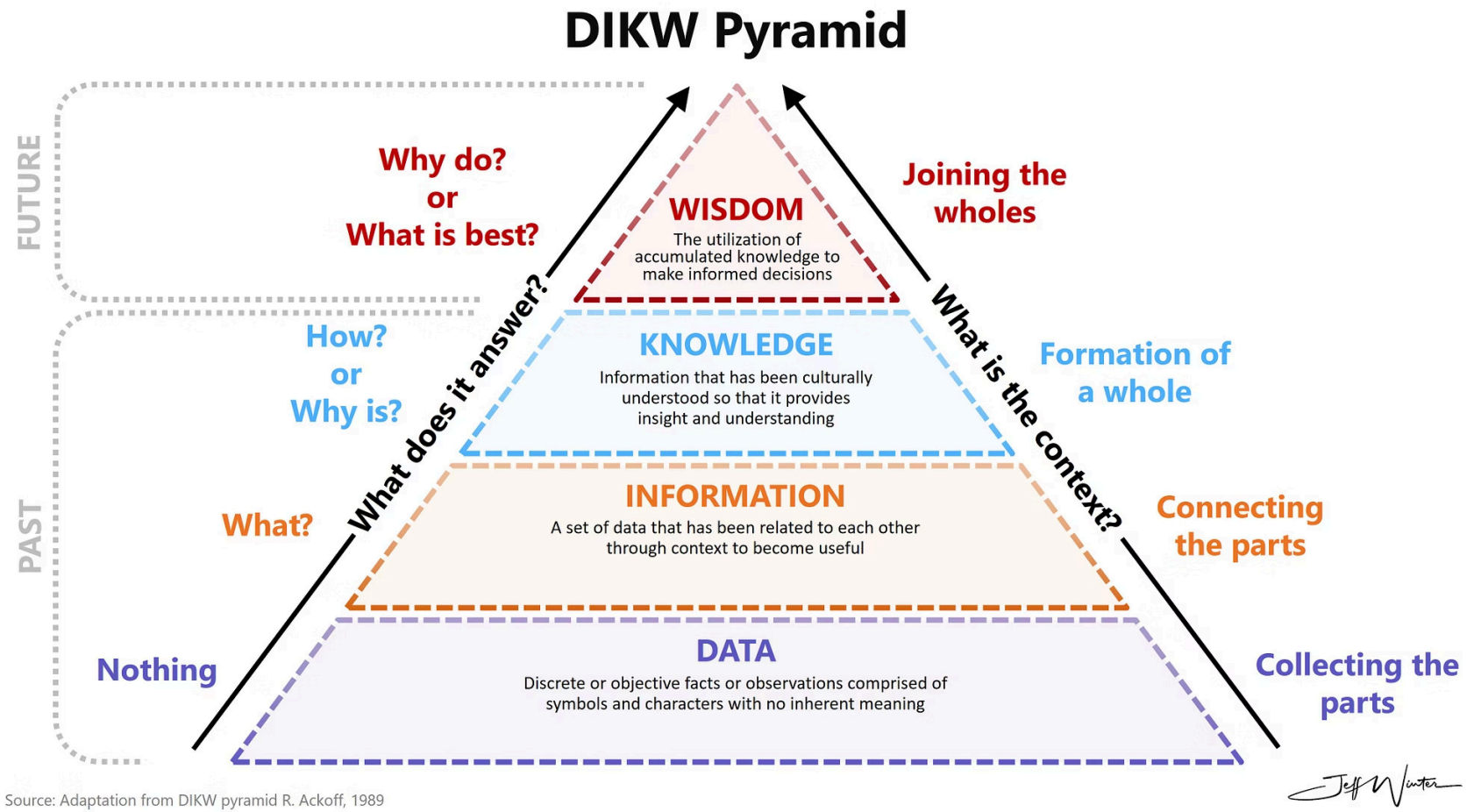
Applying LLM Capabilities For Group Projects

Bernhard Krüpl-Sypien
bernhard@became.ai

05 November 2025



Recap: Knowledge Management Model & Modes



Source: jeffwinterinsights.com / Russell Ackoff 1989

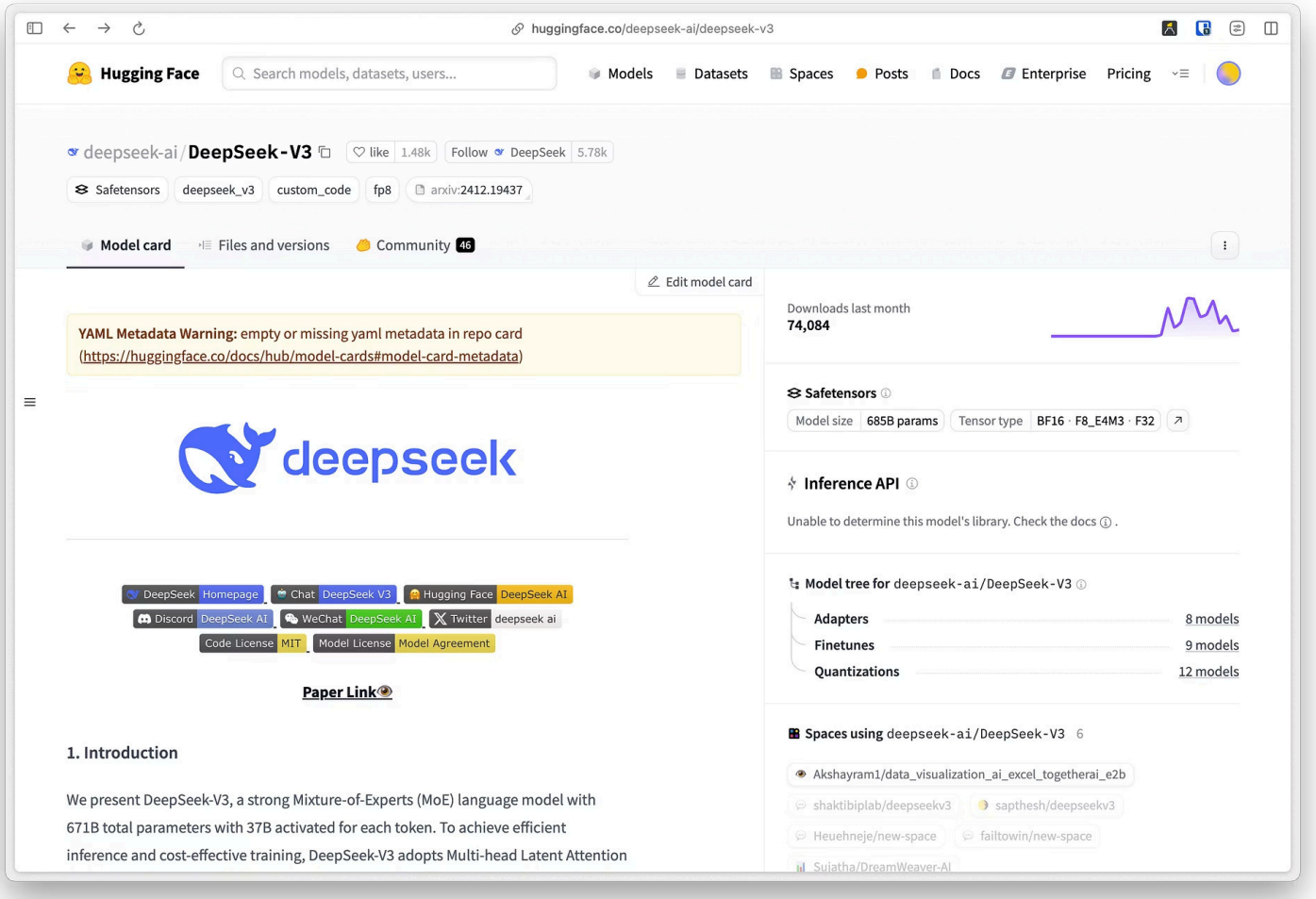
Modes of Personal Knowledge Management

We propose the following mental model of modes when working with PKM:

- **Collect**
  - Collect data / information (/ knowledge?)
  - Traditional challenges: Information overload, inconsistent formats
- **Surface**
  - Find collected thoughts, either active (search) or passive (agentic)
- **Connect**
  - Connect related ideas, consider perspective
- **Synthesize**
  - Synthesize knowledge from collection
- **Reflect**
  - Analyze thinking patterns and knowledge gaps

Accessing LLMs

HuggingFace



Types of Pre-Trained Models

1 Foundation Models (= Base Models)

Usually trained for next token prediction / text completion

2 Fine-Tuned Models

Specialised for specific tasks:

- *Instruct models*
  - follow instructions on a single prompt
  - often work well in conversations
- *Chat models*
  - Optimised for multi-turn dialogues and maintaining context over several interactions
- *Multi-modal models*
  - Combine information from different modalities such as text with images etc.

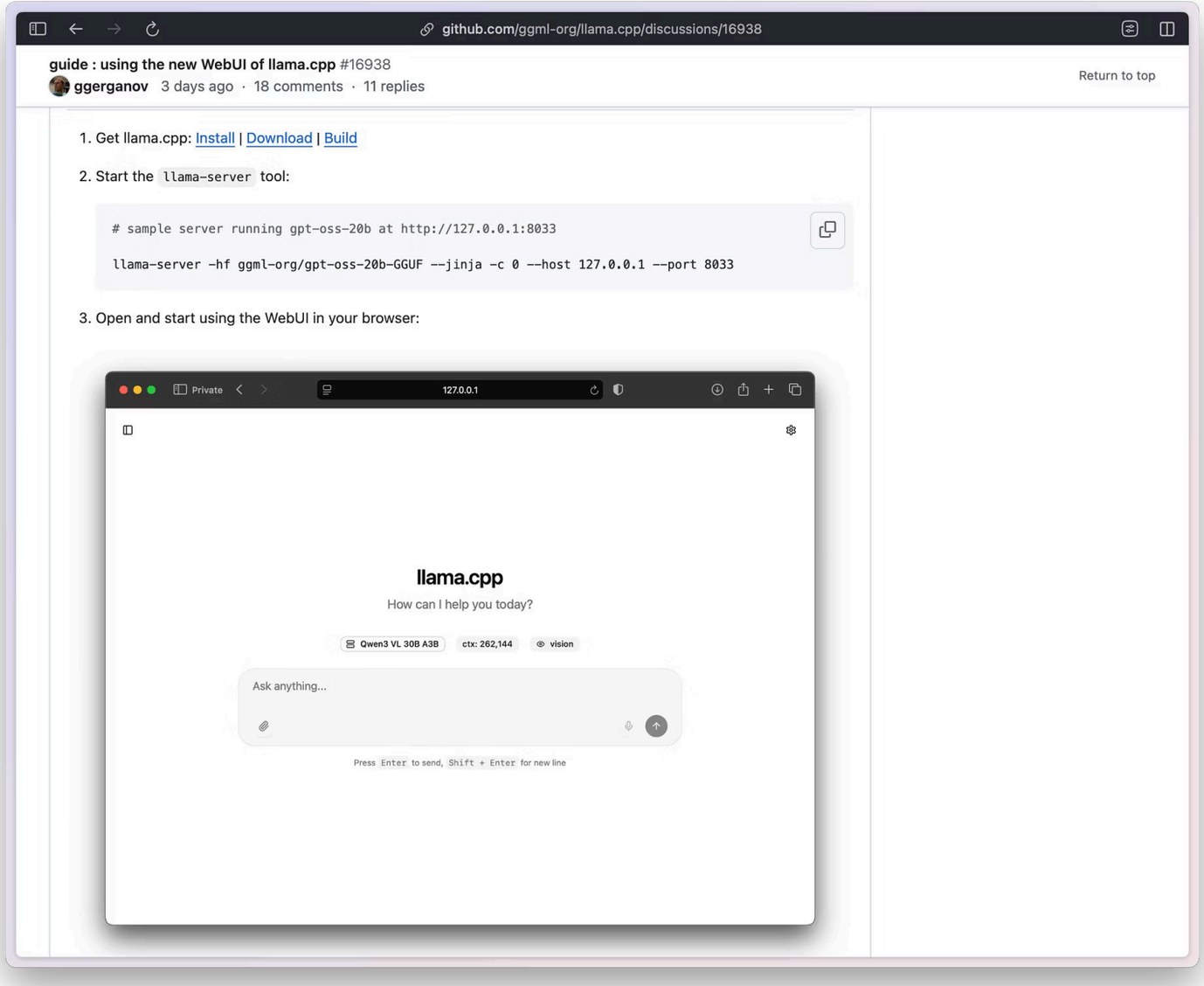
3 Embedding Models

Generate dense vector representations of input data for tasks like similarity search, clustering, or recommendation systems. Often trained with objectives like contrastive learning or triplet loss to enhance semantic relationships in the embedding space.

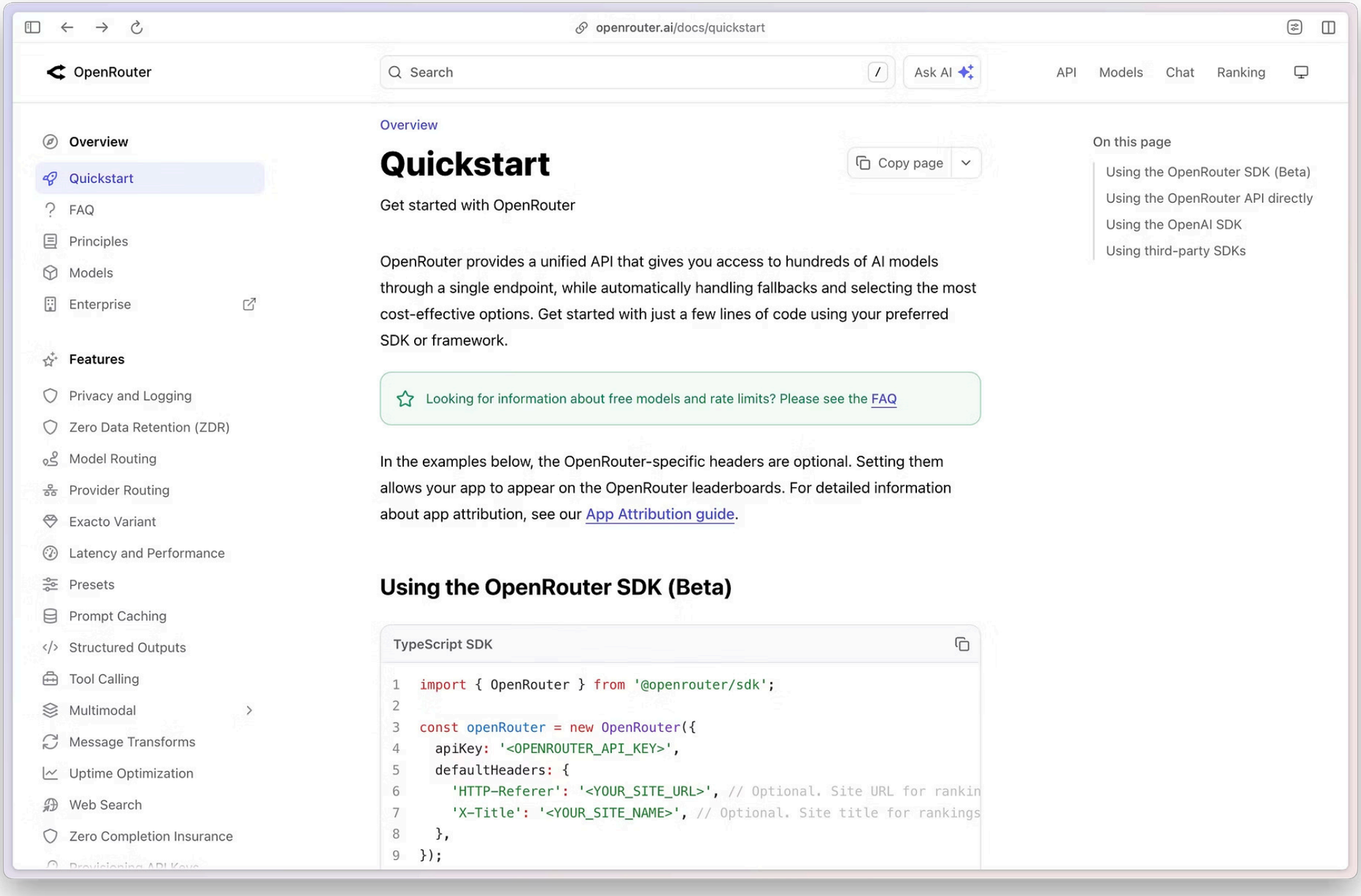
ollama

```
% ollama run phi4
>>> Who are you?
I am Phi, a language model developed by Microsoft. I'm designed to assist
users by providing information and answering questions across a wide range
of topics. My primary function is to understand your queries and generate
helpful responses based on the data and knowledge I was trained on up
until October 2023. If you have any questions or need assistance with
something, feel free to ask!
```

llama.cpp UI


github.com/ggml-org/llama.cpp/discussions/16938
guide : using the new WebUI of llama.cpp #16938
ggerganov 3 days ago · 18 comments · 11 replies
Return to top
1. Get llama.cpp: Install | Download | Build
2. Start the llama-server tool:
sample server running gpt-oss-20b at http://127.0.0.1:8033
llama-server -hf ggml-org/gpt-oss-20b-GGUF --jinja -c 0 --host 127.0.0.1 --port 8033
3. Open and start using the WebUI in your browser:
Private
127.0.0.1
llama.cpp
How can I help you today?
Qwen3 VL 30B A3B
ctx: 262,144
vision
Ask anything...
Press Enter to send, Shift + Enter for new line

OpenRouter

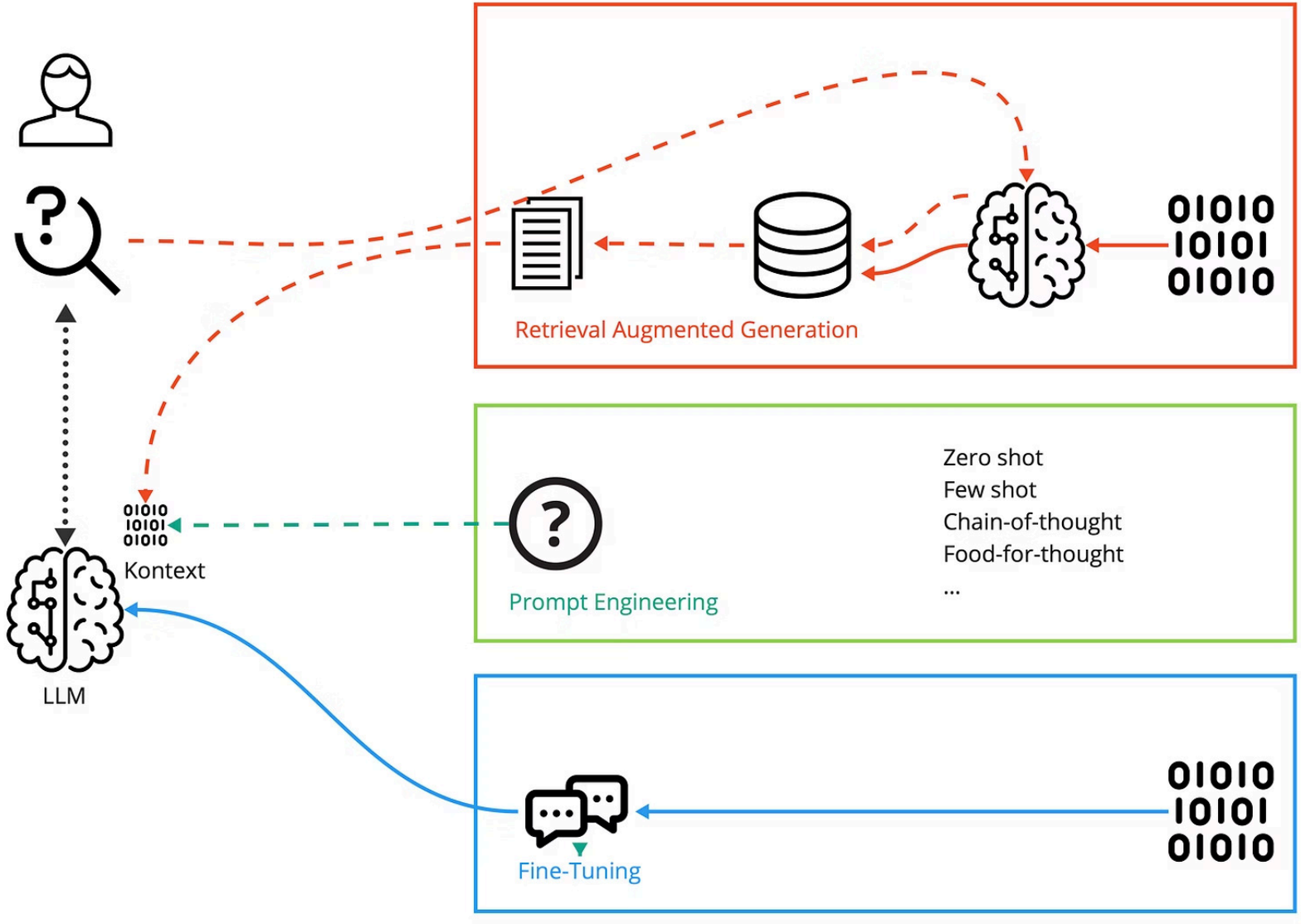

openrouter.ai/docs/quickstart
OpenRouter
Search
/
Ask AI
API
Models
Chat
Ranking
Overview
Quickstart
FAQ
Principles
Models
Enterprise
Features
Privacy and Logging
Zero Data Retention (ZDR)
Model Routing
Provider Routing
Exacto Variant
Latency and Performance
Presets
Prompt Caching
Structured Outputs
Tool Calling
Multimodal
Message Transforms
Uptime Optimization
Web Search
Zero Completion Insurance
Overview
Quickstart
Copy page
Get started with OpenRouter
OpenRouter provides a unified API that gives you access to hundreds of AI models through a single endpoint, while automatically handling fallbacks and selecting the most cost-effective options. Get started with just a few lines of code using your preferred SDK or framework.
Looking for information about free models and rate limits? Please see the FAQ
In the examples below, the OpenRouter-specific headers are optional. Setting them allows your app to appear on the OpenRouter leaderboards. For detailed information about app attribution, see our App Attribution guide.
Using the OpenRouter SDK (Beta)
TypeScript SDK
import { OpenRouter } from '@openrouter/sdk';
const openRouter = new OpenRouter({
apiKey: '<OPENROUTER_API_KEY>',
defaultHeaders: {
'HTTP-Referer': '<YOUR_SITE_URL>', // Optional. Site URL for rankin
'X-Title': '<YOUR_SITE_NAME>', // Optional. Site title for rankings
},
});
On this page
Using the OpenRouter SDK (Beta)
Using the OpenRouter API directly
Using the OpenAI SDK
Using third-party SDKs

Foundation of LLM Behaviour

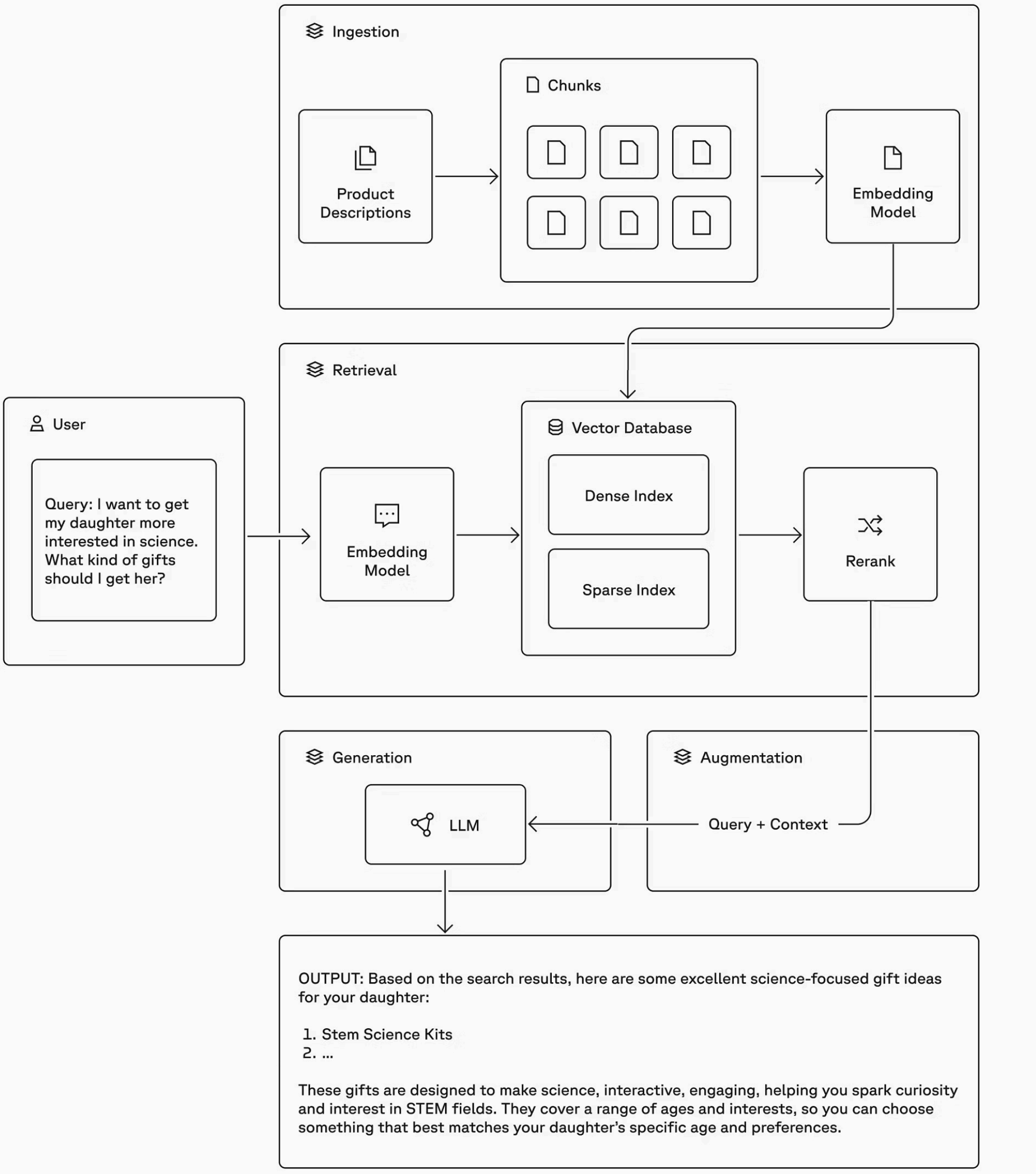
The output of an LLM relies on its *weights* and *context* – and the *program* running it ("inference")

- **Weights (long-term memory)**
  - Learnt by original training of the model
  - Determine how the model processes input and produces output
- **Context (short-term memory)**
  - The text provided to the model at inference time – system prompt, user prompt, chat history, and any additional information as in RAG
  - This short-term memory is filled with previous interactions
  - But is also the room for future interactions
  - Models with large context windows tend to forget the middle of the context

Adapting Pre-trained LLMs – Three Approaches



Retrieval Augmented Generation (RAG)



Source: pinecone.io

Prompting for In-Context Adaptation

Large Language Models (LLMs) can adapt to new tasks from examples provided directly within the prompt, a technique known as In-Context Learning. This method allows for rapid task adaptation without requiring expensive re-training or fine-tuning. We prefer to call it In-Context *Adaptation* instead of *Learning*, because there are no parameters updated or learnt.

Zero-Shot Prompting

No examples are provided in the prompt. The LLM relies solely on its vast pre-trained knowledge to understand the instruction and perform the task.

Query

Classify the sentiment of the following text as positive, negative, or neutral.

Text: I liked the hotel.
Sentiment:

Answer

positive

One-Shot Prompting

A single example is included in the prompt. This provides a clear template, helping the model understand the desired output format or style for a specific task.

Query

Classify the sentiment of the following text as positive, negative, or neutral.

Text: The hotel is terrible.
Sentiment: negative

Text: I think the food was okay. Sentiment:

Answer

neutral

Few-Shot Prompting

Multiple examples are supplied in the prompt. This significantly enhances the model's ability to follow complex instructions and quickly adapt to novel tasks.

Query

Classify the sentiment of the following text as positive, negative, or neutral.

Text: The hotel is terrible.
Sentiment: negative

Text: Excellent service, appreciated it
Sentiment: positive

Text: The bed was dirty and too short.
Sentiment:

Answer

negative

Fine-Tuning

Fine-tuning takes a pre-trained model and trains it further on a smaller, task-specific dataset. Unlike in-context learning, which leverages examples within a prompt, fine-tuning modifies the model's parameters directly.

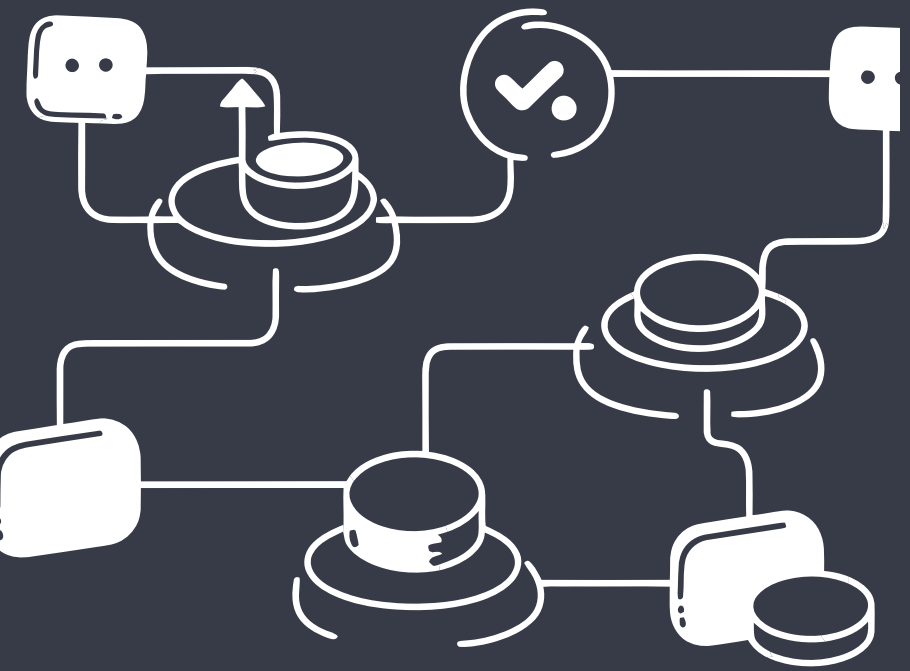
Usually **GOOD** for focusing and formatting.
Significantly improves accuracy and quality on specific target tasks.
Tailors the model's understanding and generation to a particular field.

Usually **BAD** for adding new knowledge due to overfitting and catastrophic forgetting – use RAG instead

Creating LLM Flows

- Chains of operations often work better than zero-shot
- Breaking complex tasks into manageable steps
- Example: Multiple LLM calls for summarise → extract → format instead of one single step that tries to do all at once

Implementation: Multi-step composition in code (cmp. chain mini pattern). Orchestration frameworks like LangChain. Flow engineering tools.



Mini AI Patterns

A showcase of selected architectural patterns to program without a framework

Common Architecture Patterns

Context & History Handling

Managing conversation state, summarizing long contexts, and strategically pruning history to stay within token limits whilst preserving critical information.

Data Formats

Structured input/output handling through JSON schemas, XML templates, or custom formats ensuring reliable parsing and validation of model responses.

Workflows & Chains

Connecting multiple LLM calls in sequence, passing outputs as inputs to subsequent steps, enabling complex multi-stage processing and reasoning.

Tool Usage & Function Calling

Calling external tools and functions to augment context.

Mini-Pattern: Conversation Memory Management

Problem: LLMs are stateless. They don't remember previous interactions.

```
import anthropic

client = anthropic.Anthropic(api_key="your-api-key")

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages = [{"role": "user", "content": "What is the capital of Austria?"}]
)

print(response.content[0].text)
# -> "The capital of Austria is Vienna."

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages = [{"role": "user", "content": "How many people live in it?"}]
)

print(response.content[0].text)
# -> "I don‘t know what you are referring to."
```

(*) The example uses Anthropic Claude, but the pattern works across different platforms and SDKs.

Mini-Pattern: Conversation Memory Management

Solution: The application must maintain conversation history and pass it with each request.

```
import anthropic

client = anthropic.Anthropic(api_key="your-api-key")

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages = [{"role": "user", "content": "What is the capital of Austria?"}]
)

print(response.content[0].text)
# -> "The capital of Austria is Vienna."

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages = [
        {"role": "user", "content": "What is the capital of Austria?"},
        {"role": "assistant", "content": "The capital of Austria is Vienna."},
        {"role": "user", "content": "How many people live in it?"}
    ]
)

print(response.content[0].text)
# -> "Vienna has about 2 million inhabitants."
```

Mini-Pattern: Output Parsing & Structured Data

Check whether the output of the LLM conforms to

```
import json
import re

class OutputParser:
    def __init__(self, pattern: str):
        self.pattern = pattern

    def parse(self, llm_output: str) -> dict:
        match = re.search(self.pattern, llm_output)
        if not match:
            raise ValueError(f"Output doesn't match expected format")
        return json.loads(match.group(0))

# In your LLM call, request structured output
prompt = """Extract this person's info as JSON:
"Sarah is a 28-year-old teacher"
Format: {"name": "...", "age": ..., "occupation": "..."}"""

parser = OutputParser(r'\{"name":\s*"[^"]+",\s*"age":\s*\d+,\s*"occupation":\s*"[^"]+"\}')
person = parser.parse(llm_response)
print(person["name"])  # "Sarah"
print(person["age"])   # 28
```

It

Mini-Pattern: Multi-Step Processing

Initial Processing

First LLM call extracts key information or performs initial analysis on raw input data

Intermediate Steps

Subsequent calls refine, validate, or transform the output from previous steps

Final Generation

Last call synthesizes all intermediate results into final output format

Building Simple Chains

```
def process_chain(text):
    # Step 1: Summarize text
    summary = llm_call(
        f"Summarize: {text}"
    )

    # Step 2: Extract actions
    actions = llm_call(
        f"Extract actions from: {summary}"
    )

    # Step 3: Format output
    return llm_call(
        f"Prioritize: {actions}"
    )
```

Comprehensive LLM Capabilities for the Project

Information Processing Capabilities

Transformation

Converting between different formats and structures

Example: Text to JSON, markdown to structured data

01

Implementation

LLMs can transform content using structured output techniques like function calling (defining output schemas) or response formatting (XML/JSON templates)

02

Technical Detail

Libraries like LangChain's output parsers or OpenAI's function calling API enable reliable format conversion

03

Real-world Example

Notion AI converting meeting transcripts into structured action items with assignees and deadlines

PKM Relevance: Supports Collect by standardising information formats

Extraction

Pulling specific information from larger documents

Example: Extracting key facts, claims, or data points from research papers

01

Implementation

Targeted prompting with specific extraction instructions, often using "Extract all [X] from the following text"

02

Technical Detail

For complex extraction, chain-of-thought prompting significantly improves accuracy by guiding the LLM through a step-by-step reasoning process

03

Real-world Example

Elicit.ai extracting methodology details and findings from scientific papers to compare research approaches

PKM Relevance: Enhances Collect by isolating relevant information

Summarisation (Extractive and Abstractive)

Creating concise versions of longer content

Extractive: Selecting important sentences vs. **Abstractive:** Rewriting in new words

01

Implementation

For long documents, recursive summarisation breaks content into chunks, summarises each, then summarises those summaries

02

Technical Detail

Control parameters like max_tokens and temperature allow fine-tuning summary length and creativity

03

Real-world Example

Otter.ai's Meeting Notes feature that condenses hour-long meetings into key points and action items

PKM Relevance: Supports Collect and Reflect by distilling essential information

Classification & Categorisation

Automatically tagging and categorising content

Assigning documents to predefined categories

01

Implementation

Few-shot learning with examples of correctly categorised content, or zero-shot classification using predefined taxonomies

02

Technical Detail

Comparing embedding similarity scores to predefined category embeddings often outperforms prompt-only approaches

03

Real-world Example

Gmail's automatic email categorisation system that sorts messages into Primary, Social, and Promotional tabs

PKM Relevance: Enhances Collect and Surface by organising information

Search & Retrieval Capabilities

Semantic Search

Finding conceptually related content without exact keyword matches

Using embeddings to understand meaning, not just words

01

Implementation

Converting documents and queries into vector embeddings, then comparing their similarity in vector space

02

Technical Detail

Cosine similarity between normalised vectors is the most common comparison metric, with values closer to 1 indicating greater similarity

03

Real-world Example

GitHub Copilot's code search that finds conceptually similar code snippets even without exact keyword matches

PKM Relevance: Core to Surface by improving information findability

RAG

Responding to specific queries about your knowledge base

Drawing answers from multiple sources

01

Implementation

Retrieval-Augmented Generation (RAG) pipelines that search for relevant context, then generate answers based on retrieved information

02

Technical Detail

Hybrid retrieval combining keyword search (TF-IDF/BM25) with semantic search often outperforms either method alone

03

Real-world Example

NotebookLM which answers questions based on multiple uploaded sources

PKM Relevance: Supports Surface by providing direct answers, not just documents

Knowledge Generation Capabilities

Connection Discovery

Identifying relationships between concepts and documents

Building knowledge graphs automatically

01

Implementation

Entity extraction followed by relationship classification between pairs of entities

02

Technical Detail

Relationships can be stored in knowledge graphs for exploration and analysis

03

Real-world Example

Connected Papers visualisation tool showing conceptual relationships between academic publications

PKM Relevance: Essential for Connect, revealing non-obvious relationships

Text Generation & Expansion

Creating new content based on existing knowledge

Elaborating on ideas with supporting details

01

Implementation

The autoregressive nature of LLMs.

02

Technical Detail

Controlling generation characteristics through temperature and top_p parameters to balance creativity and coherence

03

Real-world Example

Chatbots such as ChatGPT. Jasper AI expanding marketing bullet points into full blog posts whilst maintaining brand voice and style

PKM Relevance: Supports Synthesise by developing ideas further

Knowledge Synthesis

Creating new knowledge from existing information

Combining multiple perspectives or sources

01

Implementation

Multi-step prompting where initial search results are processed and integrated using explicit synthesis instructions

02

Technical Detail

Map-reduce patterns first analyse individual sources (map) then combine insights (reduce) for complex synthesis tasks

03

Real-world Example

Elicit.org creating literature reviews that identify consensus views and areas of disagreement across scientific papers

PKM Relevance: Core to Synthesise mode, creating higher-level insights

Reasoning & Inference

Drawing logical conclusions from existing information

Identifying implications not explicitly stated

01

Implementation

Chain-of-thought prompting with explicit instructions to reason step-by-step

02

Technical Detail

Tree of Thoughts approaches explore multiple reasoning paths simultaneously to find the most robust conclusion

03

Real-world Example

Claude AI analysing complex logical arguments and identifying unstated assumptions or conclusions

PKM Relevance: Enhances Reflect and Synthesise with logical extensions

Other Capabilities

Text Rewriting & Adaptation

Paraphrasing and adjusting content for different purposes

Changing complexity level or tone

01

Implementation

Style transfer techniques that explicitly define target audience and communication parameters

02

Technical Detail

Explicit instructions like "Rewrite this for a [technical/non-technical] audience" with examples of desired style

03

Real-world Example

Grammarly's tone adjustment features that rewrite text to be more formal, confident, or friendly based on the communication context

PKM Relevance: Helps Surface by presenting information appropriately

Translation

Processing multilingual knowledge bases

Translating between languages

01

Implementation

Using multilingual models that can understand and generate content across dozens of languages

02

Technical Detail

Language identification followed by translation, with special handling for technical terminology and domain-specific vocabulary

03

Real-world Example

DeepL's contextual translations that preserve technical terminology and domain-specific meaning

PKM Relevance: Expands Collect and Surface across language barriers

Code Generation & Analysis

Creating code based on natural language descriptions

Explaining existing code functionality

01

Implementation

Specialised models fine-tuned on code repositories, with attention to syntax rules and programming patterns

02

Technical Detail

Abstract Syntax Tree (AST) validation ensures generated code is syntactically correct and follows language-specific rules

03

Real-world Example

Claude Code generating functional code implementations from natural language descriptions of functionality

PKM Relevance: Supports Collect and Synthesise for programming knowledge

Function Calling

Enabling LLMs to interact with external tools and APIs

Manipulating context and performing actions beyond text generation

01

Implementation

The LLM identifies user intent that requires an external action, selects the appropriate function, generates its arguments, and then processes the tool's output

02

Technical Detail

LLMs are provided with tool schemas (e.g., OpenAPI specifications) that describe available functions and their parameters, enabling structured and reliable interaction

03

Real-world Example

A customer service chatbot automatically checking order status by calling an e-commerce API based on a customer's query

PKM Relevance: Extends LLM capabilities to interact dynamically with external systems, supporting Synthesise and Reflect

Agentic Capabilities

Empowering LLMs to act autonomously, plan tasks, and execute actions to achieve defined objectives.

Moving beyond simple query-response to proactive problem-solving and task completion.

01

Implementation

Involves an iterative loop of planning, acting (using tools), observing results, and refining the plan until the objective is met.

02

Technical Detail

Leverages frameworks like CrewAI, LangChain Agents or AutoGen, which integrate LLMs with a suite of tools and a control loop for decision-making and execution.

03

Real-world Example

An LLM agent autonomously researching a topic, accessing databases, summarising findings, and generating a detailed report without continuous human input.

PKM Relevance: Automates complex, multi-step knowledge workflows, transforming information into actionable insights and supporting all PKM modes.

Multi-modal Understanding

Processing text together with images, audio, video etc.

Connecting across different content types

01

Implementation

Models that align different modalities in a shared embedding space, allowing cross-modal search and understanding

02

Technical Detail

CLIP-like architectures train on pairs of images and descriptions to learn associations between visual and textual elements

03

Real-world Example

Extracting the text content from scanned PDF documents, automatically reconstructing reading order and logical structure

PKM Relevance: Enhances Collect and Connect across media types

Perspective

LLMs as Perspective Engines

Changing the POV (point-of-view)

Depending on your current goal, the available data / information / knowledge will look very different to you

01

Implementation

System prompts that establish different viewpoints or conceptual frameworks for analysis, e.g. "Analyse this product strategy from engineering, marketing, and customer perspectives"

02

Technical Detail

Set out different goals using prompts. Role-playing techniques with explicit persona definitions that include background, expertise, and typical concerns. Use different embedding strategies etc.

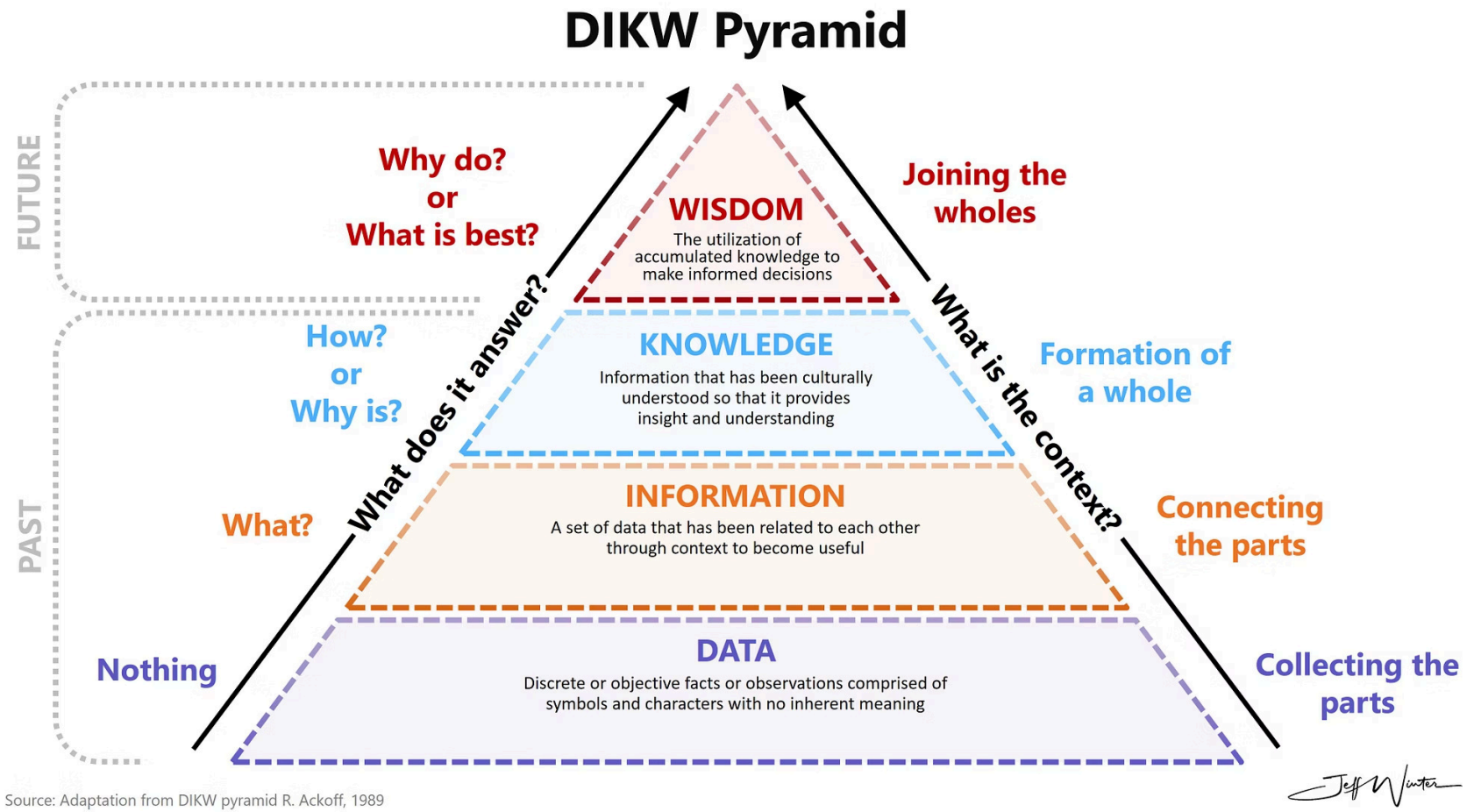
03

Real-world Example

A "Debate" feature that presents multiple viewpoints on controversial topics for an online newspaper

PKM Relevance: All modes.

Recap: Knowledge Management Model & Modes



Source: jeffwinterinsights.com / Russell Ackoff 1989

Modes of Personal Knowledge Management

We propose the following mental model of modes when working with PKM:

- **Collect**
  - Collect data / information (/ knowledge?)
  - Traditional challenges: Information overload, inconsistent formats
- **Surface**
  - Find collected thoughts, either active (search) or passive (agentic)
- **Connect**
  - Connect related ideas, consider perspective
- **Synthesize**
  - Synthesize knowledge from collection
- **Reflect**
  - Analyze thinking patterns and knowledge gaps