COMPLETE PROMPT: AI Agents Tutorial Application

Application Overview

Create a comprehensive, fully-functional HTML tutorial application for **AI Agents Tutorial** with 8 main tabs, vibrant color scheme, all interactive features, and complete content based on 36 reference graphics.

VISUAL DESIGN SPECIFICATIONS

Color Scheme: Vibrant & Modern

- **Primary Colors**:
  - Electric Blue (#3B82F6)
  - Vibrant Purple (#A855F7)
  - Energetic Orange (#F97316)
  - Fresh Green (#10B981)
  - Hot Pink (#EC4899)
- **Gradients**: Use smooth gradients between colors (e.g., blue-to-purple, orange-to-pink)
- **Backgrounds**:
  - Main: White (#FFFFFF) with subtle gradient overlay
  - Cards: Light backgrounds (#F9FAFB) with hover effects
  - Headers: Gradient backgrounds with animated transitions
- **Accents**: Bright, high-contrast colors for CTAs and important elements
- **Text**: Dark gray (#1F2937) for readability, white on dark backgrounds

Layout & Spacing

- **Container**: Max-width 1400px, centered
- **Cards**: Rounded corners (12px), subtle shadows, hover lift effects
- **Spacing**: Generous padding (24px cards, 16px elements)
- **Grid**: Responsive 3-column grid for content (2-col tablet, 1-col mobile)

Typography

- **Headers**: Bold, large (H1: 36px, H2: 28px, H3: 20px)
- **Body**: 16px, line-height 1.6 for readability
- **Code**: Monospace font, dark background with syntax highlighting

Interactive Elements

- **Buttons**: Rounded, gradient backgrounds, hover scale effects
- **Tabs**: Horizontal tabs with active indicator, smooth transitions
- **Cards**: Expandable/collapsible with smooth animations
- **Search**: Prominent search bar with icon, instant filtering

---

TAB STRUCTURE & CONTENT

Tab 1: Fundamentals

Purpose: Introduction to AI Agents - what they are, types, and core components

Content Structure:

1. **What is an AI Agent?** (Expandable Section)
    - Definition: An AI Agent is a smart system that uses a language model to achieve a specific goal by understanding instructions, planning steps, and taking actions—often with help from tools or APIs
    - These agents don't just answer questions—they can perform real-world tasks like booking appointments, managing emails, or gathering data from apps
    - Think of it like a robot with a brain and a to-do list
2. **Core Components of an AI Agent** (4 Cards with Icons) **Card 1: Language Model (The "Thinking Brain")**
    - Description: The AI that understands your request and makes decisions for the agent
    - Examples: GPT-4, Claude, Gemini 2.5, DeepSeek, LLaMA 4
    - Functions: Text generation, reasoning, planning, decision-making

    Card 2: Tools (Add-ons that allow action)
    - Description: Extensions that let agents act in the real world—API calls, polling data, sending messages

- Examples: Web search, database queries, email APIs, code interpreters
- Why Important: LLMs alone can't fetch real-time info or update systems

Card 3: Orchestration Layer (The Logic System)

- Description: The coordinator that decides how and when to use the model and tools, in what order
- Key Patterns: Chain-of-Thought (CoT), Tree-of-Thoughts (ToT), ReAct, Reflection
- Functions: Task decomposition, planning strategies, memory management, tool delegation

Card 4: Memory

- Description: Agents retain information across turns—either short-term (in-context) or long-term (external storage)
- Types:
  - In-context memory (short-term): Within the LLM's context window
  - Out-of-context memory (long-term): Stored in vector/graph databases
- Use Cases: Personalization, continuity, learning from feedback

3. **3 Ways to Use AI** (Comparison Cards) **Non-Agentic AI**
   - Definition: Basic prompt-response AI that delivers one-off outputs without memory, reasoning, or tool integration
   - How It Works: You enter a prompt, the AI responds instantly. No context carried forward
   - Use Cases: Summarizing reports, rewriting emails, generating headline ideas
   - Pros: Quick, easy, no setup required
   - Cons: No reasoning or memory, can't handle multi-step workflows, output quality depends on prompt clarity
   - Tools: ChatGPT, Claude, Gemini in basic mode

   AI Agent

   - Definition: A focused AI worker designed to automate one repetitive task with consistency and accuracy
   - How It Works: You define a single task (e.g., "compile weekly reports"). The agent connects to tools and executes without human oversight. Results delivered automatically
   - Use Cases: Syncing CRM data, automating email replies, extracting financial data
   - Pros: Saves time, easy to configure, reliable for narrow responsibilities
   - Cons: Rigid, task-bound, breaks with missing/malformed data, doesn't adapt to broader goals
   - Tools: Zapier, LangChain, n8n, OpenAI Assistants API

   Agentic AI

- Definition: A self-managing AI system that can plan, execute, and refine across multiple steps and tools
- How It Works: You provide a broad goal (e.g., "launch a product"). The system breaks it into sub-tasks, connects to tools, and executes. Uses memory and feedback loops to improve results
- Use Cases: Market research & strategy, multi-platform ad campaigns, building/testing onboarding flows
- Pros: Handles complex multi-step projects, integrates external tools and memory, produces consistent adaptive results
- Cons: Slower, more resource-intensive, requires human oversight/safety checks, can be overkill for simple tasks
- Tools: CrewAI, LangGraph, AutoGen, LlamaIndex workflows

4. **AI Agent vs Traditional AI vs AI Workflow** (Comparison Table)

Aspect	Automation	AI Workflow	AI Agent
Definition	Rule-based execution	LLM via API for tasks	Non-deterministic autonomous tasks
Core Logic	Boolean Logic	Boolean & Fuzzy Logic	Fuzzy Logic & Autonomy
Tasks	Deterministic, predefined	Deterministic tasks with flexibility	Non-deterministic, adaptive
Strengths	Reliable, fast	Handles complex rules, pattern recognition	Highly adaptive, simulates human-like reasoning
Weaknesses	Limited to explicit tasks, can't adapt	Requires data training, harder to debug	Less reliable, may produce unpredictable outcomes, slower
Example	Slack notification on new lead	Analyze/score/route leads with ChatGPT	Full internet search on leads, update records

5. **Types of AI Agents** (Expandable Grid - 7 Types) **UI Interaction Agent**
   - Description: Streamlines your web browsing experience
   - Examples: Browser automation tools, form fillers
   - Icon: 👩‍💼 Receptionist persona

   Workflow Automation Agent
   - Description: Performs repetitive automated tasks
   - Examples: Data entry, report generation

- Icon: 👷 Factory Supervisor persona

Knowledge Retrieval Agent

- Description: Helps you maximize productivity with specialized tools
- Examples: Document search, Q&A systems
- Icon: 👨💼 Knowledge worker personas

Coding Agent

- Description: Writes and maintains code
- Examples: GitHub Copilot, Cursor, Windsurf
- Icon: 👨‍💻 Developer persona

Voice Agent

- Description: Executes specific commands through speech
- Examples: Voice assistants, call centers
- Icon: 🎧 Call center persona

Tool-Specific Agent

- Description: Helps you maximize productivity with specialized tools
- Examples: Salesforce agents, Slack bots
- Icon: 🛠️ Specialist persona

Data & Analytics Agent

- Description: Analyzes data and generates insights
- Examples: Business intelligence, predictive analytics
- Icon: 📊 Analyst persona

6. **How Agentic AI Actually Works** (7-Step Visual Process) **Step 1**: Understand What Agentic AI Means
   - Agentic AI is like a super-smart helper that can think, decide, plan, and do tasks on its own
   - It's not just answering questions—it can take actions like booking a ticket or sending an email
   - Think of it like a robot with a brain and a to-do list

   Step 2: Learn the 3 Powers of Agentic AI
   - 🧠 Memory: Remembers what happened before and uses it
   - 💭 Thinking: Plans steps and thinks about choices
   - 🎯 Doing: Takes real actions, not just words
   - Example: "Plan my trip to Japan" → AI finds flights, checks weather, suggests places, books hotels —all by itself

Step 3: How Agentic AI "Thinks" & "Acts"

- It breaks your request into steps
- Thinks: "What tools do I need?" (search, maps, emails)
- Picks tools like search, maps, emails
- Then does the task—just like a personal assistant

Step 4: Know the Two Main Agent Types

- Task Agents: Focused on one goal (e.g., "Summarize this file")
- Autonomous Agents: Can handle multiple steps, adapt, make smart decisions over time

Step 5: How Agentic AI Uses Tools

- Think of it like a superhero with gadgets
- It knows when to use: Maps, Email, Excel, Google Search
- It chooses the right tool for the job—like Batman picking gadgets from his belt

Step 6: How Agentic AI Learns

- It tries, fails, learns, and improves (just like us!)
- Uses feedback: "Did the user like my plan?"
- Learns better ways to solve problems next time

Step 7: See Agentic AI in Real Life

- Auto-booking meetings, managing your calendar
- Writing and sending follow-up emails
- Creating full presentations, running a business workflow—with almost no help!

Additional Graphics Reference:

- Include clickable thumbnail gallery at bottom of each section linking to full-size reference images

---

Tab 2: Key Concepts

Purpose: Core concepts - RAG, Agentic RAG, Memory, Orchestration, Protocols

Content Structure:

1. **RAG (Retrieval-Augmented Generation)** (Major Section) **What is RAG?**
    - Definition: RAG combines external information retrieval with generative AI to produce accurate, grounded, up-to-date responses

- How it works:
    1. User query → Retrieval Algorithm → External Knowledge
    2. Query + Retrieved Documents → Large Language Model
    3. LLM → Response
- Benefits: Improved accuracy, reduced hallucinations, up-to-date information
- Applications: General-purpose QA systems, initial RAG implementations

Standard RAG Process (5 Stages):

1. **Loading**: Get data from text files, PDFs, websites, databases, APIs into your workflow
2. **Indexing**: Process and structure data for efficient queries (generate embeddings, store in vector DB)
3. **Storing**: Save indexed data in vector databases for fast retrieval
4. **Querying**: User query filters data to most relevant context
5. **Evaluation**: Assess quality and accuracy of retrieved context and generated responses

16 Types of RAG (Collapsible/Expandable Table)

Type	Key Features	Benefits	Applications	Examples
Standard RAG	Basic retrieval + generation, RAG-Sequence/Token	Improved accuracy, reduced hallucinations	General-purpose QA, initial implementations	Hugging Face Transformers, Facebook RAG
Agentic RAG	Autonomous agents, tool use, dynamic retrieval	Handles complex tasks, proactive AI	Personal assistants, research aids, dynamic service bots	LangChain Agents, OpenAI GPT-4 with Plugins, Semantic Kernel
Graph RAG	Knowledge graphs, relational reasoning	Rich information, context handling	Expert systems, medical/legal/engineering, semantic search	Neo4j Graph Database, Apache Jena, Stardog
Modular RAG	Independent modules for retrieval/reasoning/generation	Flexibility, scalability	Large collaborative projects, frequent updates	Docker & Kubernetes, Apache Kafka
Memory-Augmented RAG	External memory storage/retrieval	Continuity, personalization	Long-term context chatbots, personalized recommendations	Redis, Amazon Dynamo DB, Pinecone Vector Database
Multi-Modal RAG	Cross-modal retrieval (text/images/audio)	Richer response, accessibility	Image captioning, video summarization, multi-lingual assistants	OpenAI CLIP, TensorFlow Hub Models, PyTorch Multi-Modal Libraries
Federated RAG	Decentralized data sources, privacy-preserving	Data security, compliance	Healthcare sensitive data, collaborative organizations, federated learning	TensorFlow Federated, PySyft by OpenMinded, Federated Learning Libraries

Type	Key Features	Benefits	Applications	Examples
Streaming RAG	Real-time data retrieval	Up-to-date information, low latency	Live reporting, financial tickers, social media monitoring	Apache Kafka Streams, Amazon Kinesis, Stark Streaming
ODQA RAG	Broad knowledge base, dynamic retrieval	Broad applicability, dynamic responses	Search engines, virtual assistants, diverse queries	Elasticsearch, Haystack by Deepset, Hugging Face Transformers
Contextual Retrieval RAG	Context-aware retrieval using conversation history	Personalization, coherence	Conversational AI, customer support chatbots, context sessions	Dialogflow by Google, Rasa Open Source, Microsoft Bot Framework
Knowledge-Enhanced RAG	Integration of structured knowledge bases	Factual accuracy, domain expertise	Educational tools, professional domain applications	Knowledge Graph Embeddings Libraries, OWL API, Apache Jena
Domain-Specific RAG	Customized for industries/fields	Relevance, compliance, trustworthiness	Legal research, medical diagnosis, financial analysis	LexPredict Contract Analytics, Watson Health, Financial NLP Tools
Hybrid RAG	Combining multiple retrieval approaches	Improved recall, enhanced relevance	Complex QA systems, search engines, lexical+semantic matching	Elasticsearch with kNN Plugin, FAISS by Facebook AI, Hybrid Retrieval Strategies

Type	Key Features	Benefits	Applications	Examples
Self-RAG	Self-reflection mechanisms, iterative refinement	Enhanced accuracy, improved coherence	Content creation tools, educational platforms requiring high accuracy	OpenAI GPT Models with Fine-Tuning, Human-in-the-Loop Platforms
HyDE RAG	Hypothetical document embeddings for guided retrieval	Better recall, improved answer quality	Complex queries with implicit meanings, research assistance	Custom Implementations with Haystack Pipelines
Recursive/Multi-Step RAG	Multiple rounds of retrieval and generation	Enhanced reasoning, greater understanding	Analytical problem-solving, dialogue systems, multi-turn interactions	LangChain's Chains and Agents, DeepSet's AlphaCode Framework

2. **Agentic RAG** (Deep Dive Section) **Definition**: An agent workflow that involves real-time data retrieval and generation. It uses autonomous agents to handle retrieval, planning, and tool execution **How It Differs from Standard RAG**:

   - **Standard RAG**: User → Query → Embedding → Vector DB → Retrieved Info + Query + System Prompt → Augmented → LLM → Output
   - **Agentic RAG**: User → Query → Agent (with Memory: Short Term, Long Term; Planning: ReAct, CoT; Tools: Vector Search X, Vector Search Y, Search, Mail) → LLM → Output

   Key Components:

   1. **Memory**: Short-term and long-term memory for agent context
   2. **Planning**: ReAct (Reason + Act), Chain-of-Thought (CoT)
   3. **Tools**: Multiple vector searches, web search, email, custom APIs
   4. **Agent Loop**: Think → Act → Learn → Repeat

   Benefits:

   - Handles complex tasks requiring multiple reasoning steps
   - Proactive AI that can adapt strategies
   - Integrates multiple tools and data sources

   Use Cases:

- Personal assistants needing dynamic interaction
- Research aids combining multiple sources
- Customer service bots with complex queries

3. **Agent Memory Types** (Visual Diagram) **Context Window**:
    - System prompt
    - Tool list
    - In-context memory (info from past conversations)
    - Message history (info from current conversation)
    - User message

    In-context Memory (Short-term):

    - Refers to information available in the context window of the LLM
    - Can be both information from current conversation and pulled in from past
    - Stored temporarily during session
    - **Tools**: Redis, Postgres, in-memory storage

    Out-of-context Memory (Long-term):

    - Refers to information stored in external storage, such as vector or graph database
    - Persists across sessions
    - Enables personalization and learning
    - **Tools**: Pinecone, Weaviate, Chroma vector databases

    Popular Memory Tools:

    - FAISS: Finds similar data fast using AI math (embeddings)
    - Redis/Postgres: Stores what agents learn or remember
    - Pinecone/Weaviate/Chroma: AI's memory banks for recalling info
    - Llamaindex: Connects AI to external files and notes

4. **Agent Protocols** (2 Major Protocols) **MCP (Model Context Protocol)**
    - **Created by**: Anthropic
    - **Purpose**: Helps connect LLMs and agents to securely interact with external sources, data sources, and services
    - **How it works**: An open standard that allows LLMs/agents to fetch the latest state from your board, summarize directly in the channel
    - **Key Features**:

    - Ensures LLMs are context-aware
    - Always updated with latest state
    - Seamless handoffs between tools
- **Example**: An agent in Slack uses MCP to fetch latest state from Asana board, summarize in channel
- **Use Cases**:
    - Database query interface
    - Tool orchestration layer
    - File system access
    - Service integration bridge

A2A (Agent2Agent) Protocol

- **Created by**: Google
- **Purpose**: Enables direct communication between agents. Allows AI agents to securely communicate, coordinate tasks, and delegate across systems regardless of vendor
- **How it works**: After retrieving data update, one agent uses A2A to pass result to different "report-generating agent" responsible for summary/sending to user
- **Key Features**:
    - Direct agent-to-agent communication
    - Lets one agent delegate to another specialized agent based on skills
    - Works regardless of vendor/platform
- **Example**: After retrieving data, agent uses A2A to pass to specialized "reporting agent"
- **Use Cases**:
    - Task delegation orchestration
    - Agent ecosystem collaboration
    - Cross-platform coordination
    - Vendor-agnostic workflows

Combined Power: When MCP and A2A are combined, they enable a powerful ecosystem of intelligent agents that can collaborate across tasks and platforms

5. **Orchestration Patterns** (6 Key Patterns - Cards) **Chain-of-Thought (CoT)**
    - Description: Breaks complex problems into a series of logical steps, like solving a math problem line by line

- When to use: Problem-solving tasks requiring step-by-step reasoning
- Example: "Calculate compound interest" → Break into steps

Tree-of-Thoughts (ToT)

- Description: Extends CoT by branching into multiple possible reasoning paths and comparing them to pick the best one
- When to use: Problems with multiple valid approaches
- Example: Strategic planning, exploring alternative solutions

ReAct (Reason + Act)

- Description: Lets the agent reflect on each action, take a step, then reflect again
- When to use: Tasks requiring tool use and iterative refinement
- Example: Research task with web searches, document retrieval

Reflection Pattern

- Description: Agent evaluates its own output, flags issues, then revises
- When to use: Quality-critical tasks requiring self-correction
- Example: Code review, content editing

Planning Pattern

- Description: Agent creates a plan first, then executes tasks sequentially
- When to use: Multi-step workflows with dependencies
- Example: Project management, complex automation

Tool Use Pattern

- Description: Agent selects appropriate tools based on task requirements
- When to use: Tasks requiring external resources (APIs, databases, search)
- Example: Data analysis requiring multiple data sources

6. **AI Agent 2025 Trends** (Circular Infographic Content) **Central Hub: 2025 AI Agent Trends**
   Surrounding sections:
   - **Voice Agents**: Intelligent agents that interact via natural language, TTS
   - **Agentic RAG**: Agent workflows based on real-time data retrieval
   - **CUA (Computer Using Agents)**: Agents that can interact with computers like humans
   - **Coding Agents**: Agents for building and debugging applications 10x faster
   - **AI Agent Protocols**: Streamlining multi-agent communication
   - **Deep Research Agents**: Agents with internet access to build extensively researched reports

Tab 3: Frameworks & Tools

Purpose: Overview of frameworks, tools, tech stack for building AI agents

Content Structure:

1. **Agentic AI Tech Stack** (8 Layer Hierarchy) **Layer 1: Deployment and Infrastructure**
   - Groq, AWS, Together.ai, Baseten, Modal, Fireworks AI, Replicate, Google Cloud

   Layer 2: Evaluation and Monitoring
   - LangSmith, MLflow, Weights & Biases, Ragas, Deepchecks, Fairlearn, Holistic AI, HuggingFace, W&B

   Layer 3: Foundation Models
   - Claude 3.7 Sonnet, Claude 4, Mistral AI, Cohere, Gemini 2.5 Pro, LLaMA 4, GPT-4O

   Layer 4: Orchestration Frameworks
   - LangChain, DSPy, Microsoft AutoGen, Adalflow, LlamaIndex, Haystack, LiteLLM, Dify, Semantic Kernel, RAY

   Layer 5: Vector Databases
   - Milvus, Qdrant, Redis, pgvector, Chroma, Pinecone, Elasticsearch, Vespa, Vald, Weaviate

   Layer 6: Embedding Models
   - Voyage AI, Nomic, FastText, HuggingFace, OpenAI, spaCy, VectorFlow, Cohere

   Layer 7: Data Ingestion and Extraction
   - Scrapy, Docling, Beautifulsoup, DIFFBOT, Firecrawl, LlamaParse, Amazon Textract, Apache Tika

   Layer 8: Memory and Context Management
   - Letta, Mem0, Zep, Chroma, Cognee, LangChain, LlamaIndex

2. **Major Frameworks** (Detailed Cards - 6 Frameworks) **LangChain**
   - **Type**: General-purpose agent framework
   - **Description**: Platform for agent engineering. Modular abstractions for components necessary to work with language models
   - **Best For**: Rapid prototyping, diverse use cases, comprehensive tooling
   - **Key Features**:
     - Modular components (LLMs, retrievers, chains, agents)
     - LangSmith for tracing and debugging
     - LangGraph for stateful, multi-actor applications

    - Extensive integrations with LLM providers, vector stores, tools
- **Pros**: Mature ecosystem, extensive documentation, active community
- **Cons**: Can be complex for simple use cases, learning curve
- **Languages**: Python, JavaScript/TypeScript
- **Links**:
    - Documentation: https://docs.langchain.com/
    - GitHub: https://github.com/langchain-ai/langchain
    - Python API: https://python.langchain.com/
- **Code Example** (Python):

```python
from langchain.agents import AgentExecutor, create_react_agent
from langchain.llms import OpenAI
from langchain.tools import Tool

# Define tools
tools = [
    Tool(name="Search", func=search_func, description="Search the web"),
    Tool(name="Calculator", func=calc_func, description="Do math")
]

# Create agent
llm = OpenAI(temperature=0)
agent = create_react_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# Run
result = agent_executor.invoke({"input": "What's the weather in SF?"})
```

CrewAI

- **Type**: Multi-agent orchestration framework
- **Description**: Lean, lightning-fast Python framework for creating role-playing autonomous AI agents. Designed for collaborative intelligence
- **Best For**: Multi-agent teams, role-based workflows, autonomous collaboration
- **Key Features**:

    - Role-based agent architecture
    - Crews (autonomous teams) and Flows (deterministic control)
    - Built from scratch (no LangChain dependency)
    - Over 100,000 certified developers
- **Pros**: Simple APIs, fast execution, reliable results, great for team simulations
- **Cons**: Relatively newer than LangChain, Python-only
- **Languages**: Python
- **Links**:
    - Documentation: https://docs.crewai.com/
    - GitHub: https://github.com/crewAIInc/crewAI
    - Website: https://www.crewai.com/
- **Code Example** (Python):

```python
```

```
from crewai import Agent, Task, Crew

# Define agents with specific roles
researcher = Agent(
    role="Senior Researcher",
    goal="Find and summarize latest AI news",
    backstory="Expert in AI research",
    tools=[search_tool]
)

writer = Agent(
    role="Content Writer",
    goal="Write engaging blog posts",
    backstory="Skilled writer",
    tools=[write_tool]
)

# Define tasks
research_task = Task(
    description="Research latest AI developments",
    agent=researcher
)

write_task = Task(
    description="Write blog post from research",
    agent=writer
)

# Create and run crew
crew = Crew(agents=[researcher, writer], tasks=[research_task, write_task])
result = crew.kickoff()
```

AutoGen (Microsoft)

- **Type**: Multi-agent conversation framework
- **Description**: Framework for building AI agents and multi-agent systems. Event-driven, asynchronous architecture
- **Best For**: Complex multi-agent conversations, enterprise applications, research
- **Key Features**:
    - ConversableAgent base class for all agents

    - GroupChat for multi-agent orchestration
    - AutoGen Studio (low-code GUI)
    - Cross-language support (Python, .NET)
- **Pros**: Backed by Microsoft Research, robust architecture, great for research
- **Cons**: More complex than alternatives, still evolving (v0.4 redesign)
- **Languages**: Python, C#/.NET
- **Links**:
    - Documentation: https://microsoft.github.io/autogen/
    - GitHub: https://github.com/microsoft/autogen
    - Research: https://www.microsoft.com/en-us/research/project/autogen/
- **Code Example** (Python):

```python
from autogen import AssistantAgent, UserProxyAgent

# Create assistant agent
assistant = AssistantAgent(
    name="assistant",
    llm_config={"model": "gpt-4", "api_key": api_key}
)

# Create user proxy agent
user_proxy = UserProxyAgent(
    name="user_proxy",
    human_input_mode="NEVER",
    code_execution_config={"work_dir": "coding"}
)

# Start conversation
user_proxy.initiate_chat(
    assistant,
    message="Write Python code to analyze this data"
)
```

LlamaIndex

- **Type**: Data framework for RAG applications

- **Description**: Framework for connecting custom data sources to LLMs. Specialized in ingestion, indexing, and querying
- **Best For**: RAG applications, document QA, knowledge base queries
- **Key Features**:
  - Data connectors for multiple sources
  - Indexing strategies (vector, graph, keyword)
  - Query engines and chat engines
  - Agent capabilities built on top of RAG
- **Pros**: Best-in-class for RAG, excellent documentation, flexible
- **Cons**: Focused primarily on RAG (not general agents)
- **Languages**: Python, TypeScript
- **Links**:
  - Documentation: https://docs.llamaindex.ai/
  - GitHub: https://github.com/run-llama/llama_index
  - Website: https://www.llamaindex.ai/
- **Code Example** (Python):

```python
from llama_index import VectorStoreIndex, SimpleDirectoryReader

# Load documents
documents = SimpleDirectoryReader('data').load_data()

# Create index
index = VectorStoreIndex.from_documents(documents)

# Create query engine
query_engine = index.as_query_engine()

# Query
response = query_engine.query("What are the main topics in these docs?")
print(response)
```

LangGraph

- **Type**: Framework for stateful, multi-actor agents

- **Description**: Part of LangChain ecosystem. Build agents as graphs with cycles and state management
- **Best For**: Complex workflows with state, cyclic flows, human-in-the-loop
- **Key Features**:
  - Graph-based workflow definition
  - Built-in persistence and state management
  - Cycles and conditional edges
  - Human-in-the-loop support
- **Pros**: Powerful for complex workflows, visual graph representation
- **Cons**: Steeper learning curve, part of LangChain ecosystem
- **Languages**: Python, JavaScript
- **Links**:
  - Documentation: https://langchain-ai.github.io/langgraph/
  - GitHub: https://github.com/langchain-ai/langgraph
- **Code Example** (Python):

```python
```

```python
from langgraph.graph import StateGraph, END

# Define state
class AgentState(TypedDict):
    messages: List[str]
    next_action: str

# Define graph
workflow = StateGraph(AgentState)

# Add nodes
workflow.add_node("research", research_node)
workflow.add_node("write", write_node)

# Add edges
workflow.add_edge("research", "write")
workflow.add_edge("write", END)

# Set entry point
workflow.set_entry_point("research")

# Compile and run
app = workflow.compile()
result = app.invoke({"messages": ["Start research"]})
```

Haystack

- **Type**: NLP framework for search and QA
- **Description**: Open-source framework by Deepset. Focuses on building production-ready search and QA systems
- **Best For**: Enterprise search, production RAG, document QA
- **Key Features**:
  - Pipeline-based architecture
  - Pre-built components for RAG
  - REST API generation
  - Evaluation tools built-in
- **Pros**: Production-ready, modular, great for search use cases
- **Cons**: More specialized, smaller community than LangChain

- **Languages**: Python
- **Links**:
    - Documentation: https://docs.haystack.deepset.ai/
    - GitHub: https://github.com/deepset-ai/haystack
    - Website: https://haystack.deepset.ai/

3. **Open Source RAG Stack** (Recommended Stack) **Complete Stack**:
    - **Ingest/Data Processing**: Kubeflow, Apache Airflow, Apache Nifi, LangChain Document Loaders, Haystack Pipelines, Open Search
    - **Embedding Model**: HuggingFace Transformers, LLMWare, Nomic, Sentence Transformers, JinaAI, Cognita
    - **Retrieval & Ranking**: FAISS, Haystack Retrievers, Weaviate, Elasticsearch KNN, JinaAI Rerankers
    - **Vector Database**: Milvus (built for scale), Weaviate, PgVector, Chroma, Qdrant
    - **LLMs**: LLaMA, Mistral, Gemma, Phi-2, Deepseek, Qwen
    - **LLM Frameworks**: LangChain, CrewAI, LlamaIndex, HuggingFace
    - **Frontend Frameworks**: NextJS, SvelteKit, StreamLit, VueJS
4. **Vector Databases** (Comparison Table)

Database	Type	Best For	Key Features	Pricing
Pinecone	Managed	Production, scale	Serverless, fast, easy setup	Paid (free tier available)
Weaviate	Open-source/Managed	Flexibility, hybrid search	GraphQL, hybrid search	Open-source + paid cloud
Chroma	Open-source	Local dev, simple use cases	Lightweight, easy setup	Free (open-source)
Qdrant	Open-source/Managed	Performance, Rust-based	Fast, filtering, on-prem option	Open-source + paid cloud
Milvus	Open-source	Large-scale, enterprise	Scalable, built for big data	Free (open-source)
PgVector	PostgreSQL extension	Existing Postgres users	Native Postgres integration	Free (open-source)
Elasticsearch	Open-source/Managed	Search + vectors	Mature search features	Open-source + paid cloud
Redis	In-memory	Fast retrieval, caching	In-memory speed	Open-source + paid cloud

5. **Deployment & Infrastructure Tools** (6 Cards) **Groq**
    - Fast LLM inference hardware
    - Ultra-low latency
    - https://groq.com/

AWS (Amazon Web Services)

- Cloud infrastructure
- SageMaker for ML deployment
- https://aws.amazon.com/

Modal

- Serverless compute for ML
- Easy GPU access
- https://modal.com/

Together.ai

- Open-source model hosting
- Fine-tuning platform
- https://together.ai/

Baseten

- ML model deployment
- Autoscaling infrastructure
- https://baseten.co/

Fireworks AI

- Fast inference for open models
- Production-ready
- https://fireworks.ai/

Tab 4: Design Patterns

Purpose: Common architectural patterns for building AI agents

Content Structure:

1. **Core Agent Architectures** (6 Major Patterns - Detailed Cards) **ReAct Agent (Reason + Act)**
    - **Definition**: A reasoning and acting framework that combines thought and action in interleaved manner
    - **How It Works**:
        1. Query → Reasoning → Action → Tool → Result
        2. Agent alternates between thinking and acting
        3. Each action informs next reasoning step
    - **Pattern Flow**:
        - User Query
        - Agent: "I need to search for current information"
        - Action: Use search tool
        - Observation: Get search results
        - Agent: "Now I have the info, I can answer"
        - Response to user
    - **Used By**: Most AI agent products, LangChain, AutoGen

- **Best For**: Tasks requiring tools and iterative problem-solving
- **Pros**: Interpretable, works well with tools, handles complexity
- **Cons**: Can be slow with many iterations
- **Code Example** (Python - LangChain):

```python
from langchain.agents import create_react_agent, AgentExecutor
from langchain.llms import OpenAI
from langchain.tools import Tool

# Define tools
tools = [
    Tool(name="Search", func=search, description="Search the web"),
    Tool(name="Calculator", func=calculate, description="Do math")
]

# Create ReAct agent
llm = OpenAI(temperature=0)
agent = create_react_agent(llm, tools, react_prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

# Run
result = agent_executor.invoke({
    "input": "What's 25% of the population of Tokyo?"
})
```

CodeAct Agent

- **Definition**: Architecture by Marcus AI where agents write and execute code to accomplish tasks
- **How It Works**:
    1. User Query → Agent decides to write code
    2. Agent writes Python/JavaScript using tools as libraries
    3. Code executes in sandbox environment
    4. Agent observes outcome and iterates
- **Pattern Flow**:
    - User Query
    - Agent: Write code to solve

    - Execute code in environment
    - Observe results
    - Agent: Iterate or respond
- **Used By**: Manus, coding agents
- **Best For**: Data analysis, automation, computational tasks
- **Pros**: Powerful for technical tasks, deterministic execution
- **Cons**: Requires secure sandbox, debugging can be complex
- **Code Example** (JavaScript):

```javascript
// CodeAct pattern with code execution
const agent = {
  query: async (userInput) => {
    // Agent generates code
    const code = await llm.generateCode(userInput);

    // Execute in sandbox
    const result = await sandbox.execute(code);

    // Agent observes and decides
    if (result.success) {
      return result.output;
    } else {
      return agent.query(`Fix error: ${result.error}`);
    }
  }
};
```

Planning Pattern (Plan-and-Execute)

- **Definition**: Agent creates comprehensive plan first, then executes tasks sequentially
- **How It Works**:
    1. User → Planner creates detailed plan
    2. Planner writes steps
    3. Executor runs each step
    4. Results aggregate

- **Pattern Flow**:
  - User Goal
  - Planner: Break into steps (Step 1, 2, 3...)
  - Executor: Execute Step 1 → Execute Step 2 → Execute Step 3
  - Memory: Track state between steps
  - Final output
- **Used By**: Complex workflows, project management agents
- **Best For**: Multi-step projects with dependencies
- **Pros**: Organized, clear structure, good for complex tasks
- **Cons**: Less flexible, upfront planning overhead
- **Code Example** (Python):

```python
from langchain.agents import PlanAndExecute

# Define planner and executor
planner = create_planner(llm)
executor = create_executor(llm, tools)

# Create plan-and-execute agent
agent = PlanAndExecute(planner=planner, executor=executor)

# Run
result = agent.run("Plan a trip to Japan including flights, hotels, activities")
```

Reflection Pattern (Self-Reflection)

- **Definition**: Agent evaluates its own output using feedback loops to improve
- **How It Works**:
  1. Agent generates first draft
  2. Agent critiques its own work
  3. Agent revises based on critique
  4. Repeat until quality threshold met
- **Pattern Flow**:
  - User Query

    - Generator: Create first draft
    - Results → Reflect: Evaluate quality, flag issues
    - Response ok? → Yes: Final Response
    - Response ok? → No: Generate again (loop back)
    - Reflected Response
- **Used By**: Open Source AI, writing assistants, code review
- **Best For**: Quality-critical tasks, content creation
- **Pros**: Higher quality output, self-correcting
- **Cons**: Slower, requires additional LLM calls
- **Code Example** (Python):

```python
def reflection_agent(task):
    # Generate initial output
    output = llm.generate(task)

    # Reflection loop
    for _ in range(3):  # Max 3 iterations
        critique = llm.generate(f"Critique this: {output}")

        if "acceptable" in critique.lower():
            break

        # Revise based on critique
        output = llm.generate(
            f"Original: {output}\\nCritique: {critique}\\nRevise:"
        )

    return output
```

Multi-Agent Workflow

- **Definition**: Multiple specialized agents work together to accomplish complex task
- **How It Works**:
    1. User query goes to aggregator/orchestrator
    2. Orchestrator delegates to specialized agents

3. Each agent handles specific sub-task
4. Results aggregate back to user

- **Pattern Flow**:
  - User Query
  - Aggregator LLM
  - Distribute to: Agent 1, Agent 2, Agent N (with tools)
  - Aggregate results
  - Output
- **Used By**: Gemini Deep Research, CrewAI, AutoGen
- **Best For**: Complex tasks requiring different expertise
- **Pros**: Specialization, parallel processing, modular
- **Cons**: Complex orchestration, coordination overhead
- **Code Example** (Python - CrewAI):

```python
from crewai import Agent, Task, Crew

# Create specialized agents
researcher = Agent(role="Researcher", goal="Find information")
analyst = Agent(role="Analyst", goal="Analyze data")
writer = Agent(role="Writer", goal="Write report")

# Define tasks
tasks = [
    Task(description="Research topic", agent=researcher),
    Task(description="Analyze findings", agent=analyst),
    Task(description="Write report", agent=writer)
]

# Create crew
crew = Crew(agents=[researcher, analyst, writer], tasks=tasks)
result = crew.kickoff()
```

Agentic RAG Pattern

- **Definition**: RAG enhanced with agentic capabilities - dynamic retrieval, tool use, planning

- **How It Works**:
    1. Query → Agent decides what information needed
    2. Agent uses tools to retrieve from multiple sources
    3. Agent reasons about retrieved info
    4. Agent generates response with citations
- **Pattern Flow**:
    - User Query
    - Agent (with Memory + Planning + Tools)
    - Tools: Vector Search X, Vector Search Y, Web Search, Email
    - Agent decides: Which tool? How many searches?
    - Retrieve and synthesize
    - LLM generates final output
- **Used By**: Perplexity, enterprise RAG systems
- **Best For**: Complex queries requiring multiple data sources
- **Pros**: More intelligent retrieval, handles complex queries
- **Cons**: More complex than standard RAG, slower
- **Code Example** (Python):

```python
```

```
from langchain.agents import create_openai_functions_agent
from langchain.tools.retriever import create_retriever_tool

# Create retriever tools
vector_tool = create_retriever_tool(
    vector_retriever,
    "vector_search",
    "Search internal documents"
)

web_tool = create_retriever_tool(
    web_retriever,
    "web_search",
    "Search the web"
)

# Create agentic RAG
tools = [vector_tool, web_tool]
agent = create_openai_functions_agent(llm, tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=tools)

result = agent_executor.invoke({
    "input": "Compare our Q3 results to industry benchmarks"
})
```

2. **Orchestration Techniques** (6 Techniques - Cards) **Chain-of-Thought (CoT)**
    - Breaks complex problems into step-by-step reasoning
    - Agent "thinks aloud" through problem
    - Improves accuracy on complex tasks
    - Example prompt: "Let's think step by step..."

    Tree-of-Thoughts (ToT)
    - Explores multiple reasoning paths simultaneously
    - Evaluates and backtracks if needed
    - Best for problems with multiple solutions
    - More computationally expensive than CoT

    Self-Consistency
    - Generate multiple reasoning paths
    - Vote on most consistent answer

- Improves reliability
- Trades compute for accuracy

Agent Loop (Think-Act-Observe)

- Continuous cycle of reasoning and action
- Observe results, adjust strategy
- Core of most agent frameworks
- Enables adaptive behavior

Human-in-the-Loop

- Agent requests human input at decision points
- Critical for high-stakes tasks
- Ensures oversight and safety
- Slows execution but increases trust

Hierarchical Planning

- Break task into high-level goals
- Each goal broken into sub-goals
- Tree structure of tasks
- Good for complex, multi-layered problems

3. **Agent Maturity Levels** (5 Levels) **Level 0: Direct Tool Call**
    - No reasoning, just direct API calls
    - Deterministic, rule-based
    - Example: "Send email to John"

    Level 1: ReAct Loop

    - Single-step helper with basic reasoning
    - Can use one tool at a time
    - Example: "Search and summarize"

    Level 2: Planner + Executor

    - Creates plan, then executes with short-term memory
    - Can handle multi-step workflows
    - Example: "Book flight, hotel, create itinerary"

    Level 3: Multi-Agent Crew

    - Multiple agents with specialized roles

- Agents debate and critique each other
- Example: Research team with researcher, analyst, writer

Level 4: Autonomous System

- Long-term memory and self-healing capabilities
- Can adapt plans based on failures
- Requires minimal human intervention
- Example: Fully autonomous business operations

4. **Tool Use Patterns** (How Agents Select and Use Tools) **Pattern**: User Query → Agent (decides tool) → Select Tool(s) → Use Tool(s) → Generate Response → Results **Tool Selection Strategies**:

    1. **Function Calling**
        - LLM returns structured tool call
        - Most common pattern
        - Example: `{"tool": "search", "query": "weather SF"}`
    2. **Prompt-Based Selection**
        - Tool descriptions in prompt
        - Agent chooses via text generation
        - More flexible but less structured
    3. **Embedding-Based Selection**
        - Tools embedded in vector space
        - Retrieve relevant tools via similarity
        - Scales to many tools

    Tool Invocation Patterns:

    - **Sequential**: One tool after another
    - **Parallel**: Multiple tools simultaneously
    - **Nested**: Tool calls tool calls tool
    - **Conditional**: If-then tool selection

5. **Workflows and Patterns Comparison** (Visual Diagram Content) **Automated Workflow (rule-based, non-AI)**:

    - User query → Defined step 1 → Defined step N → Response
    - Deterministic, fast, limited flexibility

    AI Workflow (non-agentic):

- User query → Ask AI over query → Response
- Adds intelligence but still single-pass

Agentic Workflow:

- User query → Planning (plan + sub-tasks) → Execute plan with tools → Reflect on results of actions → Response
- Full agent loop with memory and tools
- Most flexible and powerful

Tab 5: Coding Agents

Purpose: Specialized focus on AI coding assistants and development tools

Content Structure:

1. **Top 10 Coding Agents (IDEs to ease Dev Workflow in 2025)** Each agent gets a detailed card with:
   - Logo/Icon
   - What it does
   - Prompt example
   - Best for
   - Pricing
   - Link

1. LOVABLE

- **What it does**: Converts plain English into working React code
- **Prompt example**: "Build a landing page with login"
- **Best for**: Frontend developers, React beginners
- **Pricing**: Free tier + paid plans
- **Link**: https://lovable.dev/

2. REPLIT

- **What it does**: Real-time AI programmer that writes & debugs code
- **Prompt example**: "Get instant help on JavaScript syntax issues"
- **Best for**: Live coding, debugging, learning
- **Pricing**: Free + paid tiers

- **Link**: https://replit.com/

3. CURSOR

- **What it does**: AI-powered code editor with built-in chat
- **Prompt example**: "Refactor this code to be more efficient"
- **Best for**: Code reviews, pair programming
- **Pricing**: Free + Pro plan
- **Link**: https://cursor.sh/

4. WINDSURF

- **What it does**: Autonomous coding agent that builds full-stack apps
- **Prompt example**: "Build a full-stack app with authentication"
- **Best for**: Solo devs, startup founders
- **Pricing**: Free trial + subscription
- **Link**: https://codeium.com/windsurf

5. TRAE AI

- **What it does**: For code generation, refactoring, debugging, and more
- **Prompt example**: "Portfolio site with contact form"
- **Best for**: Devs, AI Engineers
- **Pricing**: Free tier available
- **Link**: https://trae.ai/

6. EMERGENT.SH

- **What it does**: Creates full-stack dashboards from natural language
- **Prompt example**: "Build a dashboard that tracks users and revenue"
- **Best for**: Data engineers, product teams
- **Pricing**: Open beta
- **Link**: https://emergent.sh/

7. MANUS AI

- **What it does**: Create code, voice scripts, or video content from a prompt
- **Prompt example**: "Create a video script for a new product"
- **Best for**: Content creators, dev-marketing teams
- **Pricing**: Freemium
- **Link**: https://manus.ai/

8. ANTIGRAVITY

- **What it does**: Agentic IDE at a higher, task-oriented level
- **Prompt example**: "Create a new React app with Tailwind"
- **Best for**: Hackathons, MVP builders
- **Pricing**: Beta access
- **Link**: https://antigravity.dev/

9. ROCKET..NEW

- **What it does**: Generates fullstack apps
- **Prompt example**: "Generate code from a screenshot of a login page"
- **Best for**: websites, Webapps, Games
- **Pricing**: Free tier
- **Link**: https://rocket.new/

10. BOLT..NEW

- **What it does**: Generates code from natural language
- **Prompt example**: "Create a Python script that scrapes data"
- **Best for**: Automation, scripting, data tasks
- **Pricing**: Free + Pro
- **Link**: https://bolt.new/

2. **Coding Agent Capabilities** (What They Can Do) **Code Generation**
    - Write entire functions/classes from descriptions
    - Scaffold projects and boilerplate
    - Generate unit tests automatically
    - Create documentation from code

Code Explanation

- Explain complex code in plain English
- Break down algorithms step-by-step
- Identify code smells and anti-patterns
- Suggest improvements

Debugging & Refactoring

- Find and fix bugs
- Optimize performance

- Improve code readability
- Update deprecated syntax

Integration & Deployment

- Set up CI/CD pipelines
- Configure cloud deployments
- Manage dependencies
- Handle environment variables

3. **Coding Agent Workflow** (How to Use Effectively) **Step 1: Define Clear Requirements**
    - Be specific about what you want
    - Include tech stack preferences
    - Mention constraints (performance, security)
    - Example: "Build a REST API with Flask, PostgreSQL, JWT auth"

    Step 2: Iterative Development

    - Start with basic structure
    - Test and verify each component
    - Refine with follow-up prompts
    - "Now add error handling to that function"

    Step 3: Code Review

    - Ask agent to review its own code
    - Request security audit
    - Check for best practices
    - "Review this code for security vulnerabilities"

    Step 4: Documentation

    - Generate README files
    - Create API documentation
    - Add inline comments
    - "Write documentation for this API"

    Step 5: Testing

    - Generate unit tests
    - Create integration tests
    - Test edge cases

- "Write pytest tests for this function"

4. **Best Practices for Coding Agents Prompting Tips**:
    - Be specific about language and framework
    - Provide context about the project
    - Include example inputs/outputs
    - Specify code style preferences

    Security Considerations:
    - Never share sensitive API keys in prompts
    - Review generated code for vulnerabilities
    - Use code scanning tools
    - Don't blindly trust generated security code

    Quality Assurance:
    - Always test generated code
    - Verify edge cases
    - Check for performance issues
    - Ensure code follows project conventions

    Collaboration:
    - Use version control (Git)
    - Document agent-generated code
    - Review with human developers
    - Keep humans in the loop for critical decisions

5. **Comparison: Coding Agents vs Traditional Development**

Aspect	Traditional	With Coding Agent
Speed	Hours to days	Minutes to hours
Boilerplate	Manual typing	Auto-generated
Documentation	Often neglected	Auto-generated
Testing	Manual creation	Auto-generated tests
Learning Curve	Steep for new tech	Flatter with AI guidance
Debugging	Manual search	AI-assisted suggestions
Code Review	Human-only	AI + human
Best For	Complex architecture decisions	Repetitive tasks, boilerplate

---

Tab 6: Learning Path

Purpose: Step-by-step guide from beginner to advanced AI agent development

Content Structure:

1. **Learning Roadmap Overview** (Visual Path) **Level 1: Basics of GenAI and RAG**
    - GenAI Introduction
    - Basics of LLMs
    - Basics of Prompt Engineering
    - Data Handling and Processing
    - Introduction to API Wrappers
    - RAG Essentials

    Level 2: AI Agent Specials
    - Introduction to AI Agents
    - Building a Simple AI Agent
    - Learning About Agentic Memory
    - Basics of Multi-Agent Collaboration
    - Learn Agentic Frameworks

- Basics of Agentic Evaluation
- Basics of Agentic Workflow
- Learning Agentic RAG

2. **7 Stages of AI Agent Mastery** (Detailed Breakdown) **BEGINNER LEVEL Level 1: Understand What an AI Agent Is**
    - **Goal**: For beginners exploring what AI agents are and why they matter
    - **Topics**:
        - Performance vs. rate
        - Function calling basics
        - Prompt engineering fundamentals
        - Agent vs. skillful vs. agentic
        - Single agent vs. multi-agent
    - **Skills to Learn**:
        - Understand AI agent basics
        - Learn LLM fundamentals
        - Grasp tool-calling concepts
    - **Projects**:
        - ChatGPT Clone
        - Simple Q&A Bot
        - Agent demo (ReAct, Qwen, Anthropic)
    - **Resources**:
        - OpenAI documentation
        - Anthropic Claude docs
        - LangChain tutorials
    - **Time**: 1-2 weeks

    Level 2: Learn Prompt Engineering & Role Design

    - **Goal**: Design agent behavior with advanced prompts and predefined roles
    - **Topics**:
        - System vs. user prompts
        - Prompt patterns

    - Role definition
    - Few-shot learning
    - Instruction following
- **Skills to Learn**:
    - Master prompting techniques
    - Design effective agent personas
    - Control agent behavior
- **Projects**:
    - OpenAI Playground experiments
    - Anthropic Prompt Library
    - PromptPerfect testing
    - Saytonic chatbots
- **Resources**:
    - OpenAI Prompt Engineering Guide
    - Anthropic Prompt Library
    - Learn Prompting course
- **Time**: 1-2 weeks

Level 3: Add Memory & Context Handling

- **Goal**: Enable agents to store, recall, and use memory across conversations
- **Topics**:
    - Short-term memory (in-context)
    - Long-term memory (external storage)
    - Persistent conversations
    - Structured data storage
    - Context window management
    - Generation of facts / embeddings
- **Skills to Learn**:
    - Implement memory systems
    - Use vector databases
    - Manage conversation state

- **Projects**:
    - Long-term memory bots (Redis, Pinecone)
    - Memory Modules (FAISS)
    - LangChain memory integration
    - Personal coach with memory
- **Resources**:
    - Pinecone docs
    - Weaviate tutorials
    - LangChain memory guide
- **Time**: 2-3 weeks

INTERMEDIATE LEVEL Level 4: Enable Tool Use and Action Execution

- **Goal**: Let agents execute real actions using tools and API calls
- **Topics**:
    - Tool/function definition
    - Function calling
    - API integration
    - External data sources
    - Tool selection logic
- **Skills to Learn**:
    - Define tool schemas
    - Implement function calling
    - Integrate external APIs
    - Handle tool errors
- **Projects**:
    - Weather Tool Agent (OpenAI API)
    - Email Tool (SendGrid)
    - Database Tool (SQL)
    - Zapier API Executes
    - LangChain Tools
    - Serper API Tool

- **Resources**:
    - OpenAI Function Calling docs
    - LangChain Tools documentation
    - API integration guides
- **Time**: 2-3 weeks

Level 5: Design Multi-Step Reasoning and Planning

- **Goal**: Build agents that plan, reason, and self-correct to achieve complex goals
- **Topics**:
    - Task decomposition
    - Chain-of-Thought
    - Tree-of-Thoughts
    - Planning loops and steps
    - Self-correction
    - Multi-step workflows
- **Skills to Learn**:
    - Implement CoT/ToT
    - Create planning loops
    - Build self-reflection mechanisms
- **Projects**:
    - Research Agent (multi-step)
    - Planner-Executor agents
    - AutoGPT-style agents
    - Business planning AI
    - Reflection Agents (R&A/test)
    - Plan-Execute Agents
- **Resources**:
    - ReAct paper
    - Tree-of-Thoughts paper
    - LangGraph tutorials
- **Time**: 3-4 weeks

ADVANCED LEVEL Level 6: Deploy Multi-Agent Systems

- **Goal**: Create groups of specialized agents that collaborate, delegate, and specialize
- **Topics**:
  - Role-based agents
  - Shared memory
  - Inter-agent communication
  - Task delegation
  - Orchestration strategies
  - Multi-agent debate
- **Skills to Learn**:
  - Design agent teams
  - Implement role specialization
  - Build orchestration systems
- **Projects**:
  - CrewAI role-based crews
  - Agent lifecycle coordination
  - Multi-agent (AutoGen)
  - Research team workflow
  - Startup simulation
- **Resources**:
  - CrewAI documentation
  - AutoGen tutorials
  - Multi-agent papers
- **Time**: 4-6 weeks

Level 7: Build Agentic Ecosystems with Real Automation

- **Goal**: Launch enterprise-grade agent systems that integrate with real-world workflows
- **Topics**:
  - Event-based agents
  - Agent lifecycle (Cost, routing, fallback)
  - System monitoring

        - Testing & Eval
        - Business automation
        - Governance & safety
    - **Skills to Learn**:
        - Deploy production systems
        - Monitor agent performance
        - Implement safety guardrails
        - Build business integrations
    - **Projects**:
        - Business workflows with AI (Cost tracking)
        - SLA/alerting/reliability
        - Agents (Zapier)
        - Customer support automation
        - LLM observability (LangSmith)
        - Agent fallback patterns
    - **Resources**:
        - LangSmith for monitoring
        - Production deployment guides
        - Enterprise AI best practices
    - **Time**: 6-8 weeks ongoing
3. **Recommended Learning Resources** (Curated List) **Courses**:
    - DeepLearning.AI: AI Agents in LangGraph
    - Coursera: Generative AI with LLMs
    - Udemy: Complete LangChain Bootcamp
    - YouTube: Playlist on Agentic AI

    Documentation:
    - LangChain: https://docs.langchain.com/
    - CrewAI: https://docs.crewai.com/
    - AutoGen: https://microsoft.github.io/autogen/
    - LlamaIndex: https://docs.llamaindex.ai/

Papers & Research:

- ReAct: Synergizing Reasoning and Acting in Language Models
- Tree of Thoughts: Deliberate Problem Solving with LLMs
- Reflexion: Language Agents with Verbal Reinforcement Learning
- Generative Agents: Interactive Simulacra of Human Behavior

Communities:

- LangChain Discord
- r/LangChain subreddit
- AI Agent Dev Twitter/X community
- GitHub discussions

4. **Skills Matrix** (Track Your Progress) Interactive checklist with skill categories: **Foundation Skills**:
   - ☐ Prompt Engineering
   - ☐ LLM Integration
   - ☐ API Wrappers
   - ☐ Memory Management
   - ☐ Semantic Search
   - ☐ Context Awareness
   - ☐ Fine-Tuning
   - ☐ Tokenization

   LLM Integration:
   - ☐ Model Selection
   - ☐ Task Automation
   - ☐ User Interaction Design

   Advanced Skills:
   - ☐ API Integration
   - ☐ Latency Optimization
   - ☐ Monitoring & Logging
   - ☐ Data Embeddings
   - ☐ Knowledge Retrieval
   - ☐ Scalability Planning
   - ☐ Adaptive Learning
   - ☐ Reinforcement Learning
   - ☐ Natural Language Understanding (NLU)
   - ☐ Agent Frameworks
   - ☐ Vector Databases

- [ ] Evaluation Metrics
- [ ] Generative Quality
- [ ] Iterative Development
- [ ] Bias Mitigation
- [ ] Ethical AI Design
- [ ] Chain-of-Thought Prompting
- [ ] Zero-shot/Few-shot Learning
- [ ] Dialogue Management
- [ ] Human-in-the-loop
- [ ] Testing & Validation
- [ ] Deployment Strategies
- [ ] Multi-Agent Systems
- [ ] Dynamic Prompting
- [ ] Performance Optimization
- [ ] Conversational UX Design
- [ ] Problem-solving Frameworks
- [ ] Workflow Automation
- [ ] Agent Cooperation
- [ ] Self-Consistency Prompting
- [ ] Cloud Deployment
- [ ] Security & Privacy
- [ ] Documentation & Clarity
- [ ] User Feedback Loops
- [ ] Continuous Learning
- [ ] Real-World Use Cases

Tab 7: Use Cases & Projects

Purpose: Real-world applications and implementation examples

Content Structure:

1. **AI Use Cases to 10x Your Productivity** (Table)

AI Use Case	Tool
Content Creation	Jasper, Copy.ai
Data Analysis	Julius AI, Code Interpreter
Customer Support	Retell AI, Vectorshift AI
Lead Generation	Clay, Phantombuster
Document Processing	ChatGPT, Grok
Email Automation	Instantly, Saleshandy
Social Media	Buffer AI, Taplio

2. **10 Infrastructure Tasks AI Can Now Automate** (Pulumi Neo capabilities) **1. Explain & Diagram**
    - Turns complex resource relationships into conversational insights & diagrams
    - Instant answers to infrastructure queries
    - Visualizes live state

    2. Generate Infrastructure Code
    - Creates production-ready Pulumi code from natural language
    - Eliminates blank page problem
    - Specify cloud, resources, architecture

    3. Migrate from Terraform
    - Analyzes .tfstate/HCL & generates equivalent Pulumi (Python/TS/Go)
    - Preserves patterns during migration
    - Modernizes legacy code

    4. Automate CI/CD
    - Generates GitHub Actions workflows tailored to Pulumi stacks
    - Includes preview & production stages
    - Manages secrets/gates

    5. Spot Inefficiencies
    - Identifies redundant resources & patterns
    - Recommends consolidation (e.g., ASGs)
    - Reduces overhead & waste

6. Upgrade Kubernetes

- Plans complex EKS/K8s upgrades (e.g., ver 1.27 → 1.28)
- Maps dependencies & validates compatibility
- Safer "Day 2" ops

7. Rapid Security Response

- Instantly maps resources affected by CVEs
- Generates targeted remediation PRs
- Speed of machine response

8. Remediate Policy Violations

- Detects non-compliant resources (e.g., unencrypted volumes)
- Auto-generates fix PRs
- Automated governance

9. Update Lambda Runtimes

- Scans for outdated runtimes across accounts
- Manages bulk upgrade rollouts
- Removes manual maintenance toil

10. Enforce Multi-Account Policy

- Applies organizational policies uniformly
- Tags, encryption, security across AWS Orgs
- Governance at scale

3. **9 MCP, AI Agents, Context Engineering, and RAG Projects** (Project Cards) **Project 1: Content Creation Workflow Using Meltia**

- **Description**: Multi-agent workflow using FirecrawlI, K-Agent, LinkedIn Agent, X Agent
- **Components**: Orchestrated web scraping, Meltia agents, social media posting
- **Tech Stack**: Meltia, Firecrawl, LinkedIn API, X (Twitter) API
- **Use Case**: Automated content generation and distribution
- **Link**: https://mcp.DailyDoseOfDS.com

Project 2: Fastest RAG Stack using Binary Quantization

- **Description**: Developer in Streamlit app, RAG pipeline with binary quantization for speed
- **Components**: 1. Source, 2. Embed, 3. Binary quantization, 4. Vector database powered by Milvus
- **Tech Stack**: Streamlit, Milvus, Binary Quantization, Embedding models

- **Use Case**: Ultra-fast document retrieval
- **Link**: https://mcp.DailyDoseOfDS.com

Project 3: MCP-Powered Deep Research Workflow

- **Description**: Bright web search agent (RAG-7, Rearch, Tools) with Tavily, Exa, Brave APIs
- **Components**: Research Agent, MCP Server, multiple search APIs, Social Media Agents
- **Tech Stack**: MCP, Tavily, Exa, Brave, Social platforms
- **Use Case**: Comprehensive research automation
- **Link**: https://mcp.DailyDoseOfDS.com

Project 4: Multi-Agent Browser Automation Workflow

- **Description**: Planner Agent → Browser Tool → Response Generator → Scraper Agent + Browser Agent
- **Components**: Remote workspace, automation flow, scraped data, indexed URLs, metadata
- **Tech Stack**: Browser automation tools, MCP, agent orchestration
- **Use Case**: Automated web research and data collection
- **Link**: https://mcp.DailyDoseOfDS.com

Project 5: MCP-Powered Agentic RAG Workflow

- **Description**: Vector database, Milvus Vector Store, self-hosted vector database with LLM context
- **Components**: Relevant Data, contextual query, knowledge retrieval, chat interface
- **Tech Stack**: Milvus, RAG pipeline, LLM integration
- **Use Case**: Knowledge base with intelligent retrieval
- **Link**: https://mcp.DailyDoseOfDS.com

Project 6: Context Engineering Workflow

- **Description**: Orchestrated with VOCODE, user input, possible execution, augmented memory, conversational agent admin, user text, speech + system
- **Components**: Data filters, data enrichment, augmented context
- **Tech Stack**: VOCODE, Context engineering, Memory systems
- **Use Case**: Enhanced conversational AI
- **Link**: https://mcp.DailyDoseOfDS.com

Project 7: MCP-Powered Financial Analyst

- **Description**: Investor query, Task Stack (context, instructions, Pixel plot, EOD data), Final report with charts, Python Backend (EOD CSV dataset)
- **Components**: Data pipeline, analysis agent, visualization

- **Tech Stack**: Python, EOD data APIs, MCP, Visualization libraries
- **Use Case**: Automated financial analysis
- **Link**: https://mcp.DailyDoseOfDS.com

Project 8: MCP-Powered Voice Agent Workflow with Qwen3

- **Description**: Voice command, convert speech to text (AssemblyAI), transcribe, Qwen3, synthesize response, convert text to speech, return audio
- **Components**: STT, Qwen3 LLM, TTS pipeline
- **Tech Stack**: AssemblyAI, Qwen3, TTS engine, Audio processing
- **Use Case**: Voice-controlled AI assistant
- **Link**: https://mcp.DailyDoseOfDS.com

Project 9: Unified MCP Server with MindsDB

- **Description**: MindsDB MCP Server integrates with external data (Slack, Gmail, Google Drive, etc.) and external systems
- **Components**: Multiple API integrations, unified interface
- **Tech Stack**: MindsDB, MCP, API connectors
- **Use Case**: Central hub for multi-platform agent
- **Link**: https://mcp.DailyDoseOfDS.com

Project 10: Build Backend-ready Agents with Xpander

- **Description**: Xpander is framework-agnostic, integrates SDK Query, Xpander Response, Agent
- **Components**: Framework integration, backend APIs
- **Tech Stack**: Xpander, Backend frameworks
- **Use Case**: Production-ready agent backends
- **Link**: https://mcp.DailyDoseOfDS.com

Project 11: Corrective RAG Agentic Workflow

- **Description**: User Query → Reference Data (Knowledge Context, template) → Validate → Web-search → Query → Merge context → Response
- **Components**: Milvus Vector Search, relevance check, fallback mechanism
- **Tech Stack**: Milvus, Web Search APIs, RAG pipeline
- **Use Case**: Self-correcting RAG with web fallback
- **Link**: https://mcp.DailyDoseOfDS.com

Project 12: Build Reasoning LLMs using GRPO

- **Description**: Data sourcing → tokenization (Tokenize Prompt, Tokenize Response) → GLIM Sampling Engine → Response Generator → Reward Server → Update Model
- **Components**: Less calculation, reasoning steps, reward calculation
- **Tech Stack**: GRPO, Custom LLM training
- **Use Case**: Training specialized reasoning models
- **Link**: https://mcp.DailyDoseOfDS.com

Project 13: Text-to-SQL + RAG Hybrid Agentic Workflow

- **Description**: Streamlit App → Agent (Clennlab) → Text2SQL + RAG → Vector Database (Milvus Vectorstore) + Tool Operations → Query Results
- **Components**: Database, text2SQL conversion, hybrid retrieval
- **Tech Stack**: Streamlit, Cleunlab, Milvus, SQL databases
- **Use Case**: Natural language database queries
- **Link**: https://mcp.DailyDoseOfDS.com

4. **Real-World Agent Applications** (By Industry) **Customer Support**:
    - Auto-booking meetings
    - Managing your calendar
    - Writing and sending follow-up emails
    - Creating full presentations

    Business Operations:
    - Running a business workflow—with almost no help!
    - Traffic flow control
    - Distributed robotics coordination
    - Market simulation modeling
    - Disaster response planning

    Development & Engineering:
    - Code generation assistant
    - Content summarization tool
    - Conversational AI chatbot
    - Data-driven insights generator
    - Automated trading bots
    - Smart home management

- Supply chain optimization
- Virtual research assistant

Research & Analysis:

- Autonomous research assistant
- Workflow automation agent
- Personalized recommendation engine
- Intelligent customer support
- Legal document summarization
- Scientific literature synthesis
- Customer support chatbot

Healthcare:

- Medical diagnosis support
- Patient record analysis
- Treatment recommendation

Finance:

- Risk assessment
- Financial risk navigation
- Dynamic resource allocation
- Smart manufacturing control

Education:

- Personalized learning assistant
- Long-term customer support
- Research continuity agent
- Adaptive workflow planner

5. **5-Step Process to Get Started** (Action Items) **Step 1: Identify AI Use Case**
    - When to automate: Repetitive tasks, data processing, content creation, customer service
    - Choose the right AI: ChatGPT for text & image, Claude for analysis & code, Gemini for multimodal

    Step 2: Select AI Model/Tool

    - Choose based on your use case from comparison tables

    Step 3: Design Agent Workflow

- Map the process: Define inputs/outputs & decision logic

Step 4: Connect APIs & Data

- Integration Setup: Link tools, databases & platforms

Step 5: Deploy & Monitor

- Launch: Track performance, human evals, optimize results

6. **Agent Prompt Templates** (Ready-to-Use) **Lead Enrichment Prompt**:

```
Analyze [company_name] and provide:
• Industry classification and niche
• Tech stack they likely use
• Pain points for AI/product/service offering
• Estimated employee count (need for automation)
• AI readiness score (1-10)

Output format: JSON with these fields
```

Content Generation Prompt:

```
Create [content_type] for [target_audience]:

Context: [your_product/service]
Tone: Professional but conversational
Length: [specify]
Include: Pain point, solution, social proof
CTA: [desired_action]

Add personalization tokens: [first_name], [company]
```

AI Agent Prompt:

```
You are an AI assistant that helps with [task].

Your capabilities:
- [capability_1]
- [capability_2]
- [capability_3]

Constraints:
Always: [rules_to_follow]
Never: [things_to_avoid]

Process:
1. Analyze user input for [criteria]
2. Apply logic: [decision_tree]
3. Execute output in [format]
```

Tab 8: Agent Directory

Purpose: Comprehensive directory of 100+ AI agents by category

Content Structure:

Display Format: Grid of expandable cards organized by category. Each category shows count, and expands to reveal all agents with icons, names, and links.

Categories (12 Total):

1. No-Code Business Workflow Agents (Expandable - 9 agents)

- Lindy
- Bardeen
- Relevance AI
- Lovable
- Flowice AI
- Portals AI
- Zapier AI Agents
- Virtualsync
- Make

2. Low-Code Internal Tool Agents (Expandable - 9 agents)

- Budibase
- Dronahq
- Retool AI
- Tooljet AI
- Appsmith AI
- Interaratio
- Pory.io
- Jext
- Superblocks Admin AI

3. API & Integration-Focused Agents (Expandable - 9 agents)

- Portman AI
- Marit AI
- Tray.io
- Stack AI
- Pipedream
- Lyzer Agent Studio
- Questmate AI
- Zams
- Workato AI
- n8n

4. Solopreneur & Creator Agents (Expandable - 9 agents)

- Promptly
- Slated AI
- Promptly
- Tugron AI
- BerriAI
- Copy Monkey

- Adflow AI
- Vzy AI
- Typestream AI

5. Open Source Agent Frameworks (Expandable - 9 agents)

- LangChain Agents
- Flowise
- AgentGPT
- CrewAI
- Botpress
- SuperAgent
- Bricklayer AI
- OpenAgents
- Dify (self-hosted)

6. Data & Knowledge Agents (Expandable - 9 agents)

- MindStudio
- Genei
- Humata
- AskViable
- Harpa AI
- Komo AI
- Kadoa
- Kanch AI
- Dataherald

7. Autonomous & Reasoning Agents (Expandable - 9 agents)

- ChatGPT Agent
- E2B Agent
- AutoGPT

- AgentOps
- Dify
- Langroid
- CrewAI
- OpenDevin
- MetaGPT

8. Enterprise Automation Agents (Expandable - 9 agents)

- Vertex AI Agent
- Automation Anywhere AI
- Microsoft Copilot
- Business AI for Enterprise
- Beam AI
- UIPath AI Centre
- Arcee AI
- Superblocks AI

9. Multimodal & UX-Aware Agents (Expandable - 9 agents)

- Landbot AI
- Quest AI
- Fermat
- Durable AI
- Bubble + AI Plugin
- TeleportHQ AI
- Typedream AI
- Plasmic AI
- Chipp AI

10. E-Commerce & Retail Agents (Expandable - 9 agents)

- Triple Whale

- Rep AI
- Octane AI
- ShopAgain AI
- Chatling Engine
- LimeChat AI
- Cartloop
- Tidio AI
- Zowie AI

11. Customer Support & Sales Chat Agents (Expandable - 9 agents)

- Chatsimple
- Genei
- Tiledesk
- Komo AI
- Intercom Fin AI
- Konch AI
- HeyDay AI
- AskViable

12. Voice-Based Agents (Expandable - 9 agents)

- Synthflow AI
- PolyAI
- Verage AI Studio
- Voys
- Resemble AI
- Voiceflow
- Voyae
- Alan AI
- Speechify
- Vapi.ai

Agent Card Structure (When Expanded):

- Icon/Logo
- Agent Name
- One-line description
- Primary use case
- Link to official site

INTERACTIVE FEATURES IMPLEMENTATION

1. Learning Journal

Location: Accessible from all tabs via floating button

Structure:

```javascript
{
  entries: [
    {
      id: unique_id,
      date: timestamp,
      tab: "tab_name",
      section: "section_name",
      timeSpent: minutes,
      confidence: 1-5, // star rating
      notes: "user notes",
      topics: ["topic1", "topic2"]
    }
  ]
}
```

UI Components:

- Floating "+ Journal Entry" button (bottom right)
- Modal with form:
  - Auto-filled: Current tab, current section, timestamp
  - User input: Time spent (number input), Confidence (5-star rating), Notes (textarea)

    - Save button stores to persistent storage
- Journal view: List of all entries, filterable by tab/date, sortable

2. Progress Tracking

Implementation: Checkbox system for each major section

Storage Structure:

```javascript
{
  progress: {
    tab1: {
      section1: {completed: true, status: "mastered"},
      section2: {completed: false, status: "learning"}
    }
  },
  stats: {
    totalSections: 150,
    completed: 45,
    percentComplete: 30
  }
}
```

Visual Progress Bars:

- Tab-level: Overall completion % for each tab
- Section-level: Individual section status indicators
- Status options: "Not Started" (gray), "Learning" (yellow), "Understood" (blue), "Mastered" (green)
- Click to cycle through statuses

Dashboard:

- Overall progress: Circular progress indicator showing total completion
- Per-tab breakdown: Horizontal bar charts
- Recent activity: Last 5 sections marked complete

3. Search Functionality

Implementation: Full-text search across all content

Search Features:

- Prominent search bar in header
- Real-time filtering as user types
- Search across: Agent names, concepts, frameworks, definitions, code examples
- Results display: Card with snippet, breadcrumb (Tab > Section), highlight matching text
- Click result to jump to that section

Search Algorithm:

```javascript
function search(query) {
  const results = [];
  // Search all tabs
  tabs.forEach(tab => {
    tab.sections.forEach(section => {
      if (section.content.includes(query)) {
        results.push({
          tab: tab.name,
          section: section.name,
          snippet: extractSnippet(section.content, query),
          relevance: calculateRelevance(section.content, query)
        });
      }
    });
  });
  return results.sort((a, b) => b.relevance - a.relevance);
}
```

4. Category Filtering

Implementation: Filter dropdowns/pills for categorized content

Filter Options:

- **Agent Directory**:
  - Category (No-Code, Low-Code, Enterprise, etc.)
  - Use Case (Business, Development, Customer Support, etc.)
- **Frameworks**:

    - Language (Python, JavaScript, Both)
    - Type (Multi-Agent, RAG-focused, General)
- **RAG Types**:
    - Application (QA systems, Enterprise, Real-time, etc.)
    - Complexity (Basic, Intermediate, Advanced)
- **Design Patterns**:
    - Maturity Level (0-4)
    - Pattern Type (Reasoning, Tool Use, Multi-Agent)

UI:

- Pills/Tags for active filters
- Dropdown menus for selecting filters
- "Clear All Filters" button
- Result count updates dynamically

5. Expandable/Collapsible Sections

Implementation: Accordion-style sections with smooth animations

Behavior:

- Default state: Most sections collapsed (except first in each tab)
- Click header to expand/collapse
- Smooth CSS transitions (300ms)
- Icon indicates state: ▶ collapsed, ▼ expanded
- "Expand All" / "Collapse All" buttons at tab level

CSS:

```
css
```

```
.section {
  overflow: hidden;
  transition: max-height 0.3s ease;
}
.section.collapsed {
  max-height: 60px; /* Header only */
}
.section.expanded {
  max-height: 5000px; /* Content visible */
}
```

6. Code Language Toggle

Implementation: Switch between Python and JavaScript code examples

UI Components:

- Toggle switch at top of code sections
- "Python" | "JavaScript" pills
- Active language highlighted
- Instant code swap (no page reload)

Storage:

- User preference saved
- Applies to all code examples in app
- Persists across sessions

Data Structure:

```
javascript
```

```
{
  codeExamples: {
    example1: {
      python: "python code here",
      javascript: "js code here"
    }
  },
  userPreference: "python" // or "javascript"
}
```

7. Bookmark/Favorites

Implementation: Star/bookmark system for quick access

Features:

- Star icon next to agent names, frameworks, concepts
- Click to toggle favorite status
- "Favorites" tab or sidebar showing all bookmarked items
- Organized by type (Agents, Frameworks, Concepts, etc.)
- Quick jump to bookmarked content

Storage:

javascript

```
{
  favorites: [
    {type: "agent", name: "CrewAI", tab: "Frameworks", section: "Major Frameworks"},
    {type: "concept", name: "ReAct Pattern", tab: "Design Patterns"},
    {type: "agent_tool", name: "Cursor", tab: "Coding Agents"}
  ]
}
```

8. Comparison Mode

Implementation: Side-by-side comparison of selected items

Features:

- "Compare" button on agent/framework cards

- Select 2-4 items to compare
- Side-by-side table view:
  - Features
  - Pros/Cons
  - Use cases
  - Pricing
  - Links
- Common comparisons pre-populated:
  - AI Agent vs Agentic AI vs Traditional AI
  - LangChain vs CrewAI vs AutoGen
  - Vector RAG vs Graph RAG
  - ReAct vs Planning vs Reflection patterns

UI:

- Checkbox selection on cards
- "Compare Selected" button (appears when 2+ selected)
- Modal with comparison table
- "Clear Comparison" to start over

TECHNICAL IMPLEMENTATION DETAILS

Persistent Storage

Use window.storage API (NOT localStorage - it's not supported)

Storage Schema:

```
javascript
```

```
{
  // User preferences
  preferences: {
    codeLanguage: "python",
    theme: "vibrant",
    defaultTab: "fundamentals"
  },

  // Progress tracking
  progress: {
    tab_section_id: {
      status: "mastered",
      lastUpdated: timestamp
    }
  },

  // Learning journal
  journal: [
    {id, date, tab, section, timeSpent, confidence, notes, topics}
  ],

  // Favorites
  favorites: [
    {type, name, tab, section, link}
  ],

  // Search history
  searchHistory: ["query1", "query2"]
}
```

Data Persistence Functions:

```
javascript
```

```
// Save user progress
async function saveProgress(tabId, sectionId, status) {
  const key = `progress:${tabId}:${sectionId}`;
  await window.storage.set(key, JSON.stringify({
    status: status,
    lastUpdated: Date.now()
  }));
}

// Save journal entry
async function saveJournalEntry(entry) {
  const entries = await getJournalEntries();
  entries.push(entry);
  await window.storage.set('journal', JSON.stringify(entries));
}

// Save favorite
async function toggleFavorite(item) {
  const favorites = await getFavorites();
  const index = favorites.findIndex(f => f.name === item.name);
  if (index > -1) {
    favorites.splice(index, 1);
  } else {
    favorites.push(item);
  }
  await window.storage.set('favorites', JSON.stringify(favorites));
}
```

Responsive Design

Breakpoints:

- Desktop: > 1024px (3-column grid)
- Tablet: 768px - 1024px (2-column grid)
- Mobile: < 768px (1-column stack)

Mobile Optimizations:

- Hamburger menu for navigation
- Collapsible tabs instead of horizontal tabs
- Touch-friendly buttons (min 44px)

- Stacked search filters
- Simplified comparison view

Animation & Transitions

Smooth Interactions:

```css
/* Card hover effects */
.card {
  transition: transform 0.2s ease, box-shadow 0.2s ease;
}
.card:hover {
  transform: translateY(-4px);
  box-shadow: 0 12px 24px rgba(0,0,0,0.15);
}

/* Tab transitions */
.tab-content {
  animation: fadeIn 0.3s ease;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(10px); }
  to { opacity: 1; transform: translateY(0); }
}

/* Progress bar animation */
.progress-bar {
  transition: width 0.5s ease;
}

/* Expand/collapse */
.collapsible-content {
  transition: max-height 0.3s ease, opacity 0.3s ease;
}
```

Helper Functions

```javascript
```

```
// Tab switching
function switchTab(tabId) {
  // Hide all tabs
  document.querySelectorAll('.tab-content').forEach(tab => {
    tab.style.display = 'none';
  });
  // Show selected tab
  document.getElementById(tabId).style.display = 'block';
  // Update active tab indicator
  document.querySelectorAll('.tab-button').forEach(btn => {
    btn.classList.remove('active');
  });
  document.querySelector(`[data-tab="${tabId}"]`).classList.add('active');
  // Save preference
  window.storage.set('lastTab', tabId);
}

// Search functionality
function performSearch(query) {
  const results = [];
  const searchableContent = getAllSearchableContent();

  searchableContent.forEach(item => {
    if (item.text.toLowerCase().includes(query.toLowerCase())) {
      const snippet = extractSnippet(item.text, query);
      results.push({
        ...item,
        snippet: snippet,
        relevance: calculateRelevance(item.text, query)
      });
    }
  });

  return results.sort((a, b) => b.relevance - a.relevance);
}

// Progress calculation
function calculateProgress() {
  const total = getTotalSections();
  const completed = getCompletedSections();
  return Math.round((completed / total) * 100);
}
```

```
// Filter functionality
function applyFilters(items, filters) {
  return items.filter(item => {
    return Object.keys(filters).every(key => {
      if (!filters[key] || filters[key] === 'all') return true;
      return item[key] === filters[key];
    });
  });
}

// Export/Import data
async function exportUserData() {
  const data = {
    progress: await window.storage.get('progress'),
    journal: await window.storage.get('journal'),
    favorites: await window.storage.get('favorites'),
    preferences: await window.storage.get('preferences')
  };
  const blob = new Blob([JSON.stringify(data, null, 2)], {type: 'application/json'});
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = 'ai-agents-tutorial-data.json';
  a.click();
}
```

REFERENCE GRAPHICS GALLERY

Implementation: Clickable thumbnail gallery at the bottom of each relevant section

Features:

- Thumbnail grid (150px × 150px thumbnails)
- Click to view full-size in lightbox modal
- Image title/caption
- Navigation arrows in modal (prev/next)
- Close button (X) or click outside to close
- Zoom functionality

Gallery Structure:

```html
<div class="graphics-gallery">
  <h3>Reference Graphics</h3>
  <div class="thumbnail-grid">
    <div class="thumbnail" onclick="openLightbox(1)">
      <img src="graphic1_thumb.jpg" alt="AI Agents in Real Life">
      <p>AI Agents in Real Life</p>
    </div>
    <!-- More thumbnails -->
  </div>
</div>

<div id="lightbox" class="lightbox">
  <span class="close">&times;</span>
  <img class="lightbox-content" id="lightbox-img">
  <div class="lightbox-caption"></div>
  <button class="prev">‹</button>
  <button class="next">›</button>
</div>
```

All 36 Graphics Organized by Tab:

Tab 1 - Fundamentals:

1. AI Agents in Real Life (2025 Edition)
2. Types of AI Agents You Should Know in 2025
3. How Agentic AI Actually Works, Explained to Kids
4. 3 Ways to Use AI (Non-Agentic, AI Agent, Agentic AI)
5. Automation vs. AI Workflow vs. AI Agent (Definitions)

Tab 2 - Key Concepts: 6. AI Agent Trends of 2025 (Circular infographic) 7. AI Agents vs RAG vs LLM Workflow (Concepts to know) 8. Types of Agent Memory (Context window diagram) 9. Top 16 Agentic AI Terms 10. Agentic RAG Explained (RAG vs Agents vs Agentic RAG) 11. Vector RAG vs Graph RAG 12. 16 Types of RAG (Full table)

Tab 3 - Frameworks & Tools: 13. Agentic AI Tech Stack (8 layers) 14. AI Agents Cheat Sheet (Components diagram) 15. Open Source RAG Stack 16. Top 50 Must-Have Skills for Expert Agentic AI Developers

Tab 4 - Design Patterns: 17. Top 6 Design Patterns for Agentic AI 18. AI Agents Workflows and Patterns (Automated, AI, Agentic) 19. Components of AI Agents (Agent Runtime) 20. Agentic RAG Workflow 21. Tool Use Pattern 22. Reflection Pattern 23. Planning Pattern

Tab 5 - Coding Agents: 24. Top 10 Coding Agents (IDEs to ease Dev Workflow in 2025) 25. How to Build an AI Agent (8 steps)

Tab 6 - Learning Path: 26. How to Start Learning AI Agents (Level 1 & 2) 27. 7 Stages of AI Agent Mastery 28. AI Agent vs Agentic AI (Maturity Levels, Build Stack, Metrics) 29. AI Agents Quick Guide (Knowledge & Memory, Core Concepts)

Tab 7 - Use Cases & Projects: 30. 9 MCP, AI Agents, Context Engineering, and RAG projects 31. 10 Infrastructure Tasks AI Can Now Automate (Pulumi Neo) 32. AI Agents & Automation Cheat Sheet (5 step process)

Tab 8 - Agent Directory: 33. The AI Agents Directory (12 categories) 34. Multi-Agent Systems with CrewAI (Architecture example) 35. AI Agents Read the Docs (Strategy, Governance, Architecture, Sector-Specific) 36. Advanced Tool Use on Claude Developer Platform

---

COMPLETE DATA STRUCTURE

Note: Due to the massive amount of content, the full data will be structured in JavaScript objects. Here's the schema:

```javascript
```

```
const appData = {
  metadata: {
    appName: "AI Agents Tutorial",
    version: "1.0",
    lastUpdated: "2025-12-12"
  },

  tabs: [
    {
      id: "fundamentals",
      name: "Fundamentals",
      icon: "🎓",
      sections: [
        {
          id: "what-is-ai-agent",
          title: "What is an AI Agent?",
          type: "expandable",
          content: "Full content here...",
          graphics: [1, 2, 3, 4, 5],
          relatedLinks: []
        },
        // More sections...
      ]
    },
    {
      id: "key-concepts",
      name: "Key Concepts",
      icon: "💡",
      sections: [
        {
          id: "rag",
          title: "RAG (Retrieval-Augmented Generation)",
          type: "expandable",
          subsections: [
            {
              id: "rag-types",
              title: "16 Types of RAG",
              type: "collapsible-table",
              data: [
                {
                  type: "Standard RAG",
                  keyFeatures: "Basic retrieval and generation...",
                  benefits: "Improved accuracy...",
```

```
            applications: "General-purpose QA systems...",
            examples: "Hugging Face Transformers..."
          },
          // All 16 RAG types...
        ]
      }
    ],
    graphics: [6, 7, 10, 11, 12],
    relatedLinks: [
      {title: "LangChain RAG Tutorial", url: "https://..."},
      // More links...
    ]
  }
 ]
},
{
 id: "frameworks-tools",
 name: "Frameworks & Tools",
 icon: "🛠️",
 sections: [
  {
   id: "major-frameworks",
   title: "Major Frameworks",
   type: "cards",
   frameworks: [
    {
     name: "LangChain",
     logo: "langchain-logo.png",
     type: "General-purpose agent framework",
     description: "Platform for agent engineering...",
     bestFor: "Rapid prototyping, diverse use cases...",
     keyFeatures: ["Modular components", "LangSmith", "LangGraph", "Extensive integrations"],
     pros: ["Mature ecosystem", "Extensive documentation", "Active community"],
     cons: ["Can be complex", "Learning curve"],
     languages: ["Python", "JavaScript/TypeScript"],
     links: {
      docs: "https://docs.langchain.com/",
      github: "https://github.com/langchain-ai/langchain",
      website: "https://www.langchain.com/"
     },
     codeExamples: {
      python: "# Python code here...",
      javascript: "// JavaScript code here..."
     }
```

```
      },
      // All other frameworks...
    ],
    graphics: [13, 14, 15],
    relatedLinks: []
  },
  {
    id: "vector-databases",
    title: "Vector Databases",
    type: "comparison-table",
    data: [
      {
        database: "Pinecone",
        type: "Managed",
        bestFor: "Production, scale",
        keyFeatures: "Serverless, fast, easy setup",
        pricing: "Paid (free tier available)",
        link: "https://www.pinecone.io/"
      },
      // All vector DBs...
    ]
  }
 ]
},
{
 id: "design-patterns",
 name: "Design Patterns",
 icon: "🎨",
 sections: [
  {
   id: "core-architectures",
   title: "Core Agent Architectures",
   type: "detailed-cards",
   patterns: [
    {
     name: "ReAct Agent (Reason + Act)",
     definition: "A reasoning and acting framework...",
     howItWorks: [
      "Query → Reasoning → Action → Tool → Result",
      "Agent alternates between thinking and acting",
      "Each action informs next reasoning step"
     ],
     patternFlow: {
      steps: [
```

```
            "User Query",
            "Agent: I need to search",
            "Action: Use search tool",
            "Observation: Get results",
            "Agent: Now I can answer",
            "Response to user"
          ]
        },
        usedBy: "Most AI agent products, LangChain, AutoGen",
        bestFor: "Tasks requiring tools and iterative problem-solving",
        pros: ["Interpretable", "Works well with tools", "Handles complexity"],
        cons: ["Can be slow with many iterations"],
        codeExamples: {
          python: "# Full working example...",
          javascript: "// Full working example..."
        },
        relatedPatterns: ["Planning Pattern", "Tool Use Pattern"]
      },
      // All 6 patterns...
    ],
    graphics: [17, 18, 19, 20, 21, 22, 23]
  }
 ]
},
{
 id: "coding-agents",
 name: "Coding Agents",
 icon: "💻",
 sections: [
  {
   id: "top-10-coding-agents",
   title: "Top 10 Coding Agents (2025)",
   type: "detailed-cards",
   agents: [
    {
     name: "LOVABLE",
     logo: "lovable-logo.png",
     whatItDoes: "Converts plain English into working React code",
     promptExample: "Build a landing page with login",
     bestFor: "Frontend developers, React beginners",
     pricing: "Free tier + paid plans",
     link: "https://lovable.dev/"
    },
    // All 10 coding agents...
```

```
      ],
      graphics: [24, 25]
    }
  ]
},
{
  id: "learning-path",
  name: "Learning Path",
  icon: "🗺️",
  sections: [
    {
      id: "roadmap",
      title: "Learning Roadmap",
      type: "visual-path",
      levels: [
        {
          level: 1,
          category: "BEGINNER",
          title: "Understand What an AI Agent Is",
          goal: "For beginners exploring what AI agents are",
          topics: ["Performance vs. rate", "Function calling", "Prompt engineering"],
          skills: ["Understand AI agent basics", "Learn LLM fundamentals"],
          projects: ["ChatGPT Clone", "Simple Q&A Bot"],
          resources: [
            {title: "OpenAI documentation", url: "https://platform.openai.com/docs"},
            // More resources...
          ],
          timeEstimate: "1-2 weeks"
        },
        // All 7 levels...
      ],
      graphics: [26, 27, 28, 29]
    },
    {
      id: "skills-matrix",
      title: "Skills Matrix",
      type: "interactive-checklist",
      categories: [
        {
          name: "Foundation Skills",
          skills: [
            "Prompt Engineering",
            "LLM Integration",
            "API Wrappers",
```

```
          // All foundation skills...
        ]
      },
      {
        name: "Advanced Skills",
        skills: [
          "API Integration",
          "Latency Optimization",
          // All advanced skills...
        ]
      }
    ]
  }
 ]
},
{
 id: "use-cases",
 name: "Use Cases & Projects",
 icon: "🚀",
 sections: [
  {
   id: "productivity-use-cases",
   title: "AI Use Cases to 10x Your Productivity",
   type: "table",
   data: [
    {useCase: "Content Creation", tool: "Jasper, Copy.ai"},
    // All use cases...
   ]
  },
  {
   id: "mcp-projects",
   title: "9 MCP Projects for AI Engineers",
   type: "project-cards",
   projects: [
    {
     name: "Content Creation Workflow Using Meltia",
     description: "Multi-agent workflow...",
     components: ["Orchestrated web scraping", "Meltia agents"],
     techStack: ["Meltia", "Firecrawl", "LinkedIn API"],
     useCase: "Automated content generation",
     link: "https://mcp.DailyDoseOfDS.com",
     graphic: 30
    },
    // All 13 projects...
```

```
        ],
        graphics: [30, 31, 32]
      }
    ]
  },
  {
    id: "agent-directory",
    name: "Agent Directory",
    icon: "📚",
    sections: [
      {
        id: "directory",
        title: "AI Agents Directory",
        type: "expandable-categories",
        categories: [
          {
            name: "No-Code Business Workflow Agents",
            count: 9,
            agents: [
              {name: "Lindy", link: "https://www.lindy.ai/"},
              {name: "Bardeen", link: "https://www.bardeen.ai/"},
              {name: "Relevance AI", link: "https://relevanceai.com/"},
              // All 9 agents...
            ]
          },
          // All 12 categories...
        ],
        graphics: [33, 34, 35, 36]
      }
    ]
  }
],

graphics: [
  {
    id: 1,
    filename: "graphic1.jpg",
    title: "AI Agents in Real Life (2025 Edition)",
    description: "Overview of different AI agent types in real-world applications",
    tabs: ["fundamentals"]
  },
  // All 36 graphics...
```

```
  ]
};
```

STYLING SPECIFICATIONS

Vibrant Color Palette

```css
:root {
  /* Primary Colors */
  --electric-blue: #3B82F6;
  --vibrant-purple: #A855F7;
  --energetic-orange: #F97316;
  --fresh-green: #10B981;
  --hot-pink: #EC4899;

  /* Gradients */
  --gradient-1: linear-gradient(135deg, #3B82F6 0%, #A855F7 100%);
  --gradient-2: linear-gradient(135deg, #F97316 0%, #EC4899 100%);
  --gradient-3: linear-gradient(135deg, #10B981 0%, #3B82F6 100%);

  /* Neutrals */
  --white: #FFFFFF;
  --light-bg: #F9FAFB;
  --light-gray: #E5E7EB;
  --gray: #6B7280;
  --dark-gray: #1F2937;

  /* Shadows */
  --shadow-sm: 0 1px 2px rgba(0,0,0,0.05);
  --shadow-md: 0 4px 6px rgba(0,0,0,0.1);
  --shadow-lg: 0 10px 15px rgba(0,0,0,0.1);
  --shadow-xl: 0 20px 25px rgba(0,0,0,0.15);
}
```

Component Styles

```css
```

```css
/* Header */
.header {
  background: var(--gradient-1);
  padding: 24px;
  color: white;
  text-align: center;
  box-shadow: var(--shadow-lg);
}

/* Tabs */
.tabs {
  display: flex;
  gap: 8px;
  background: var(--light-bg);
  padding: 16px;
  overflow-x: auto;
}

.tab-button {
  padding: 12px 24px;
  border: none;
  border-radius: 12px;
  background: white;
  cursor: pointer;
  font-weight: 600;
  transition: all 0.2s ease;
}

.tab-button:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-md);
}

.tab-button.active {
  background: var(--gradient-1);
  color: white;
}

/* Cards */
.card {
  background: white;
  border-radius: 12px;
  padding: 24px;
```

```
  box-shadow: var(--shadow-md);
  transition: all 0.3s ease;
}

.card:hover {
  transform: translateY(-4px);
  box-shadow: var(--shadow-xl);
}

/* Code Blocks */
.code-block {
  background: #1F2937;
  color: #F9FAFB;
  padding: 20px;
  border-radius: 8px;
  overflow-x: auto;
  font-family: 'Courier New', monospace;
  font-size: 14px;
  line-height: 1.6;
}

/* Buttons */
.btn-primary {
  background: var(--gradient-2);
  color: white;
  padding: 12px 32px;
  border: none;
  border-radius: 24px;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.2s ease;
}

.btn-primary:hover {
  transform: scale(1.05);
  box-shadow: var(--shadow-lg);
}

/* Search Bar */
.search-bar {
  width: 100%;
  max-width: 600px;
  padding: 16px 24px;
  border: 2px solid var(--light-gray);
```

```
  border-radius: 50px;
  font-size: 16px;
  transition: all 0.2s ease;
}

.search-bar:focus {
  outline: none;
  border-color: var(--electric-blue);
  box-shadow: 0 0 0 4px rgba(59, 130, 246, 0.1);
}

/* Progress Bar */
.progress-bar {
  height: 8px;
  background: var(--light-gray);
  border-radius: 4px;
  overflow: hidden;
}

.progress-fill {
  height: 100%;
  background: var(--gradient-3);
  transition: width 0.5s ease;
}

/* Badges */
.badge {
  display: inline-block;
  padding: 4px 12px;
  border-radius: 12px;
  font-size: 12px;
  font-weight: 600;
  background: var(--light-bg);
  color: var(--dark-gray);
}

.badge.success {
  background: var(--fresh-green);
  color: white;
}

.badge.warning {
  background: var(--energetic-orange);
  color: white;
```

```
}

/* Expandable Section */
.expandable-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 16px;
  background: var(--light-bg);
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.2s ease;
}

.expandable-header:hover {
  background: var(--light-gray);
}

.expandable-content {
  max-height: 0;
  overflow: hidden;
  transition: max-height 0.3s ease, opacity 0.3s ease;
  opacity: 0;
}

.expandable-content.expanded {
  max-height: 10000px;
  opacity: 1;
  padding: 16px;
}
```

FINAL NOTES & INSTRUCTIONS

Build Priority:

1. ✅ Create HTML structure with all 8 tabs
2. ✅ Implement tab navigation system
3. ✅ Add all content from data structure
4. ✅ Implement all 8 interactive features
5. ✅ Add persistent storage using window.storage

6. ✅ Style with vibrant color scheme
7. ✅ Make fully responsive
8. ✅ Add reference graphics gallery
9. ✅ Implement code language toggle
10. ✅ Test all functionality

Content Completeness:

- ✅ All 36 graphics referenced and organized
- ✅ 100+ AI agents catalogued with expandable categories
- ✅ 16 RAG types fully detailed with collapsible table
- ✅ 6 major frameworks with Python & JavaScript examples
- ✅ 6 core design patterns with code examples
- ✅ 10 coding agents with details
- ✅ 7-level learning path with resources
- ✅ 13 real-world projects
- ✅ All interactive features specified

Key Success Criteria:

- ✅ Single-page HTML application (no external files needed except graphics)
- ✅ All JavaScript inline in `<script>` tags
- ✅ All CSS inline in `<style>` tags
- ✅ Fully functional without internet (except for external links)
- ✅ Uses window.storage API for persistence (NOT localStorage)
- ✅ Vibrant, modern, engaging design
- ✅ Mobile-responsive
- ✅ Production-ready code quality

Build This Application Now!

Create a complete, fully-functional, single-file HTML application that implements everything specified in this prompt. The application should be immediately usable without any additional setup or files.