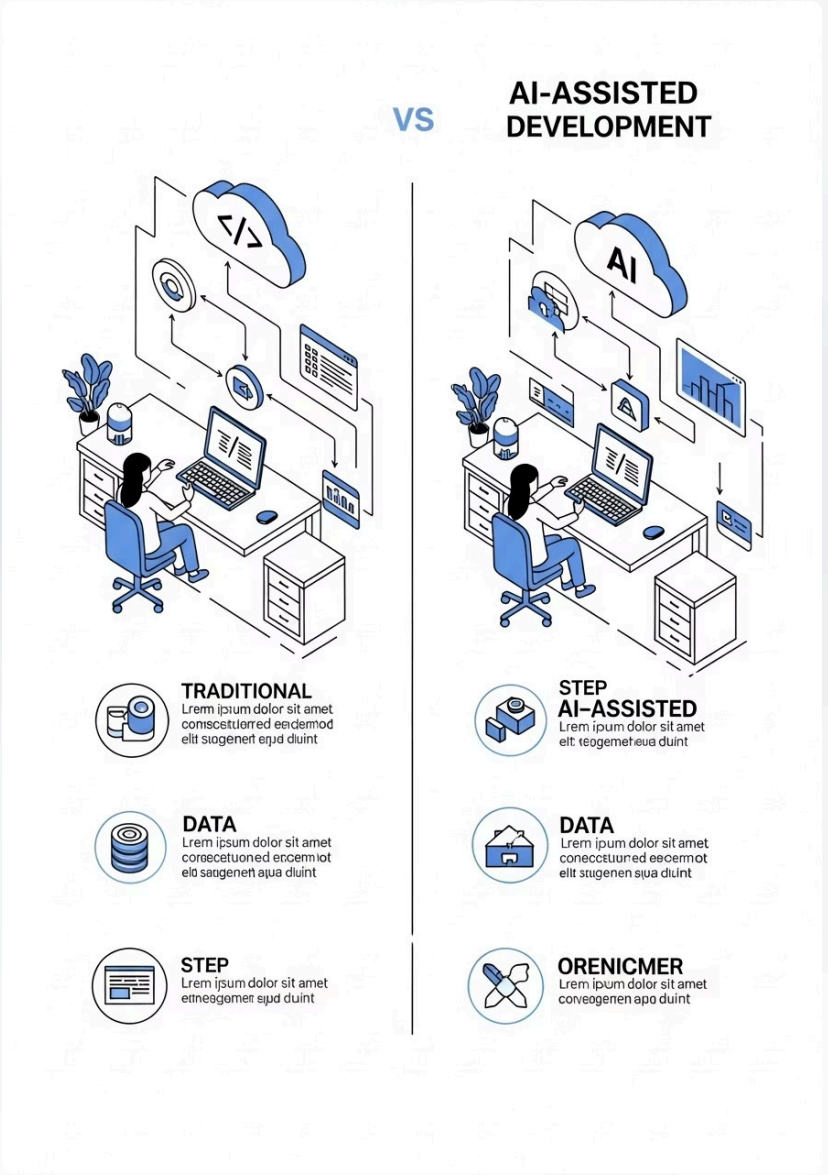
The AI-First Development Workflow

Transforming Your Engineering Practices with AI

by Versha Jain



The Prompting Shift

Traditional Workflow

Write code → Test → Debug → Refactor → Repeat

Developer leads the entire process with manual coding and iteration.

AI-First Workflow

Prompt → Plan → Build → Evaluate → Refine

Developer becomes an orchestrator, using AI to accelerate each step of the process.

Today's Goals

Prompt Engineering Foundations

Revisit the **anatomy of a good prompt** and understand how to craft effective instructions

LLM Landscape

Compare LLMs across their strengths, costs, and optimal use-cases to choose the right tool

Coding Specialists

Compare coding specialists like Copilot, Claude Code, Code Llama and others to find your ideal coding companion

RAG Systems

Learn **RAG** fundamentals: vector databases, chunking, retrieval strategies, and **n8n** orchestration

Advanced Techniques

Master **SRD prompting** and **prompt chaining** for complex development workflows

Practical Applications

Build a sentiment analysis pipeline with mini dashboard and **create a working RAG agent** with citations

Anatomy of a Good Prompt

A well-structured prompt is the foundation of effective AI-assisted development. The 5 key elements of a quality prompt:

Role

Who should the AI act as? (Senior developer, system architect, QA engineer)

Context

Background information, project details, prior work, and relevant constraints

Constraints

Technical limitations, standards, or requirements that must be followed

Deliverable

Exactly what you want returned (code, plan, analysis, explanation)

DoD

Definition of Done - specific criteria that must be met for success

DoD – Definition of Done

The Definition of Done provides clear, measurable criteria for what "complete" means in an AI build. Without this, you risk ambiguous outcomes and endless iterations.

Accuracy

Quantifiable performance target

Example: Sentiment analysis must achieve ≥85% accuracy on test dataset

Performance

Speed and resource utilization

Example: API response latency ≤2s under load of 100 concurrent requests

Completeness

Coverage and validation

Example: All unit tests pass and 90% code coverage achieved

When defining your DoD, be specific about metrics and thresholds. This gives the AI a clear target and makes it easier to evaluate the results.

Comparing AI Coding Specialists

IDEs • CLI Agents • Autocomplete Wrappers

The landscape of AI coding tools has evolved rapidly, with solutions targeting different workflows and developer preferences. Let's explore what real developers are saying about these tools.

> The following insights are compiled from actual developer reviews on Reddit, reflecting hands-on experience with these tools in production environments.

Category 1: IDEs

Full-featured environments with built-in AI

Examples: Cursor, Windsurf

Strengths

- Seamless integration with development workflow
- Polished UI & purpose-built experiences
- Fast inline edits & intelligent autocomplete

Weaknesses

- May lack depth for complex, multi-step work
- Higher cost for business licensing
- Limited customization compared to CLI tools

Reddit Reviews

> **Cursor** – "UI and autocomplete are unmatched… but for big features Claude Code wipes the floor."
>
> **Windsurf** – "Leader last year; still super fast for real-time coding."

Category 2: CLI Agents

Terminal-based AI assistants (editor-agnostic)

Examples: Claude Code, Roo Code, Cline, Opencode, Crush, Aider

Strengths

- Deep reasoning & large context windows
- Strong in planning & big refactors
- Can work across any editor or CI pipeline

Weaknesses

- Less convenient UI for some workflows
- Context switching from IDE to terminal
- Steeper learning curve for beginners

Reddit Reviews

> **Claude Code** – "Best in speed & quality; fewer reworks; handles massive codebases."
>
> **Roo Code** – "Better planning than Cursor; pricier but saves time."
>
> **Cline** – "Glass box transparency; BYOK pricing; runs inside VS Code."
>
> **Opencode** – "Great results; supports Claude/Copilot subs."
>
> **Crush** – "Great indexing; dropped subscription support."

Category 3: Autocomplete Wrappers

Boosts IDE completion & inline edits

Examples: Augment, Copilot, Kiro, Trae, Qwen Coder, Void

Strengths

- Instant productivity boost
- Enhances familiar IDE experience
- Great for continuous typing & small tweaks

Weaknesses

- Limited multi-step or architecture planning
- Often tied to specific IDEs
- Variable quality of suggestions

Reddit Reviews

> **Augment** – "Secret sauce for big codebases; indexing is unmatched."
>
> **Copilot** – "Solid, but autocomplete far behind Cursor; agent mode is lazy."
>
> **Kiro** – "Claude-powered; more tokens than Cursor; solid autocomplete."
>
> **Trae** – "Good inline editing; Cursor still faster."
>
> **Qwen Coder** – "Easy npm setup; tricky to learn."
>
> **Void** – "Shrug—no big impression."

Coding Tools: Conclusion

Category	Examples	Strengths	Weaknesses	Best For
IDEs	Cursor, Windsurf	Integrated workflow, UI polish, fast edits	Higher cost, may lack deep feature handling	Daily development, quick fixes
CLI Agents	Claude Code, Roo, Cline, Opencode, Crush, Aider	Powerful planning, huge context, editor-agnostic	Less convenient UI, context switching	Large features, refactors, code analysis
Wrappers	Augment, Copilot, Kiro, Trae, Qwen Coder, Void	Boost IDE autocomplete, smooth inline edits	Limited planning, weaker in big tasks	Typing flow, small code injections

Each category serves different development needs and workflows. Many developers use a combination of tools, leveraging the strengths of each for specific tasks in their workflow.

The LLM Lineup (2025)

- **GPT-5**
- **Gemini 2.5 Pro/Flash**
- **Grok 4**
- **Claude 3.7 Sonnet**
- **Perplexity**
- **DeepSeek R1**

The current landscape of large language models offers a diverse set of capabilities, each with unique strengths for different development scenarios. Let's explore what each brings to the table.

If you're interested in open source alternatives, also consider Meta's Llama 3.x and Google's Gemma 2.5 Flash-Lite for local or self-hosted deployments.

GPT-5 (OpenAI)

What It Is

OpenAI's latest flagship model with enhanced "thinking" capabilities and significantly improved coding and agent functionality.

Strengths

- Faster response times and better reasoning
- Superior end-to-end code generation
- Mature API ecosystem with strong tooling support

Watch-Outs

Model reshuffles have caused some churn in the ecosystem, with some teams still preferring the older GPT-4o style responses.

Reddit Vibe

> Mixed reactions—powerful capabilities, but frustration when preset behaviors change unexpectedly.

Claude 3.7 Sonnet (Anthropic)

What It Is

Anthropic's hybrid reasoning model with "extended thinking" capabilities and significant coding improvements, powering the standalone Claude Code product.

Strengths

- Exceptionally clean code output with fewer errors
- Thorough reasoning on complex tasks
- Strong enterprise channels through AWS Bedrock and Snowflake

Watch-Outs

Pro tier usage limits can be restrictive during heavy development sessions, with users noting hitting caps.

Reddit Vibe

> "Best for programming quality; code needs fewer rewrites."

Gemini 2.5 (Google)

What It Is

Google's multimodal model featuring an impressive 1 million token context window (with 2 million coming soon), particularly strong for code and long-context tasks.

Strengths

- Massive context window for handling entire codebases
- Integrated Code Assist plugins for major IDEs
- Gemini CLI with generous free usage limits
- Advanced personalization and memory capabilities

Watch-Outs

Quality varies by mode selection, and some users report odd behavior when pushing the limits of context length.

Reddit Vibe

> "Great value; improving rapidly; the huge context window is the killer feature."

Perplexity (Answer Engine)

What It Is

A search-native assistant providing real-time results with citations and a specialized "Deep Research" mode for thorough exploration.

Strengths

- Quickest path to authoritative sources with inline citations
- Tunable search context for different research needs
- Particularly strong for up-to-date information

Watch-Outs

Ongoing controversies around alleged plagiarism and web scraping practices, plus concerns about potential bias in recent partnerships.

News Pulse

Experiencing massive growth with significant new funding rounds pushing valuation higher.

Grok 4 (xAI)

What It Is

xAI's latest model featuring real-time X (formerly Twitter) data integration and a more relaxed approach to content guardrails.

Strengths

- Up-to-the-minute social signals and current events awareness
- Rapid release cadence (Grok-2→3→4 in under a year)
- Growing ecosystem integration, including Tesla vehicles

Watch-Outs

Quality can vary significantly depending on the task, and the community is still seeking comprehensive benchmark comparisons.

News Bit

Grok is now shipping in new Tesla vehicles, showing a push toward wider ecosystem integration.

DeepSeek R1 (+ Coder)

What It Is

A cost-effective reasoning-focused model (R1) paired with specialized coder variants, generating significant developer interest in 2025.

Strengths

- Exceptional mathematical and logical reasoning capabilities
- Budget-friendly pricing compared to alternatives
- Strong performance on specific reasoning tasks

Watch-Outs

Its open source/China-based stack can raise compliance questions in some organizations, with mixed evaluation results compared to "o" models.

Reddit Vibe

> "Frighteningly good at mathematical proofs and derivations."

Quick "What to Use When" Guide

Need Authoritative Sources Fast

Perplexity

Leverage the citation features and Deep Research mode to quickly find and verify information from credible sources.

Complex Code Reasoning

Claude 3.7 Sonnet or **GPT-5**

Use Claude for cleaner code with fewer bugs, or GPT-5 when building complex agent-based systems requiring tool use.

Long Documents or Repositories

Gemini 2.5 Pro/Flash

Leverage the 1M token context window to process entire codebases or long documentation at once. Use Flash for cost-efficient runs.

Social/Real-time Context

Grok

When you need the most current social insights or trending information from X and other sources.

Budget-conscious Deep Thinking

DeepSeek R1

For complex mathematical or logical reasoning tasks where you need strong performance at a lower cost point.

Reddit Pulse: What Developers Are Saying

On Claude vs ChatGPT

"Claude consistently produces higher quality code than ChatGPT, but the usage limits can be frustrating during intense development sessions."

On Gemini 2.5

"Gemini 2.5's value proposition is hard to beat; the huge context window is a game-changer for working with large projects."

On DeepSeek R1

"DeepSeek R1 is scary good at math and theoretical problems. I've been genuinely surprised by its ability to work through complex proofs."

On Grok

"Grok's release cadence is impressively fast; we're still waiting for comprehensive comparisons, but it's evolving rapidly."

These real developer perspectives highlight the practical strengths and limitations of each model in day-to-day usage.

LLM Selection Guide: Conclusion

Need	Top pick(s)	Why
Cited, current info	**Perplexity**	Real-time search + inline citations; Deep Research mode
Complex coding/agents	**GPT-5**, **Claude 3.7**	Stronger "thinking", coding, and tool use capabilities
Huge context tasks	**Gemini 2.5 Pro/Flash**	1M token context; integrated code assist + CLI tools
Live social signals	**Grok**	Direct access to X data; rapid release cadence
Low-cost reasoning	**DeepSeek R1**	Strong reasoning capabilities at lower cost

The ideal approach is often to leverage multiple models based on specific task requirements, rather than relying on a single solution for everything.

Remember that the AI landscape continues to evolve rapidly, with new capabilities and models emerging regularly. Stay flexible in your tool selection.

Back to Building Things

Flexing the Prompt Muscle

Now that we understand the landscape of available tools, let's focus on practical techniques to leverage them effectively in your development workflow.

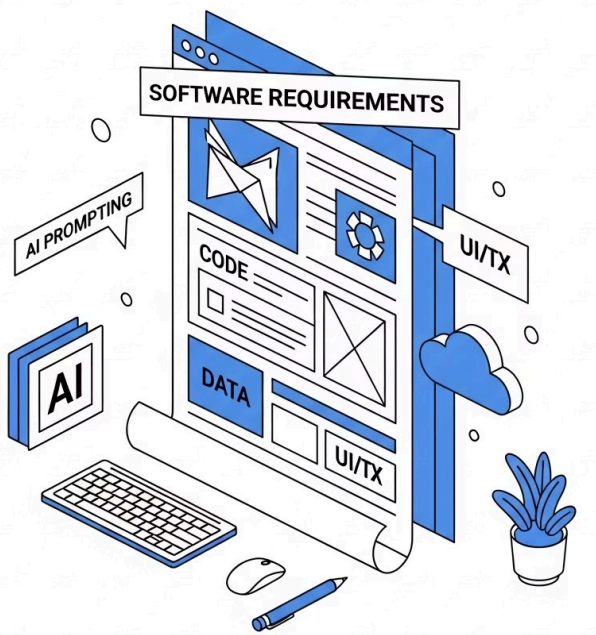
SRD Prompting & Planning

What is an SRD Prompt?

An SRD (Software Requirements Document) Prompt is a structured approach to crafting AI instructions that function as a "living" specification document.

Key characteristics:

- Comprehensive structure with clear sections
- Designed to be reusable across development stages
- Serves as a source of truth for the project
- Can be easily updated as requirements evolve



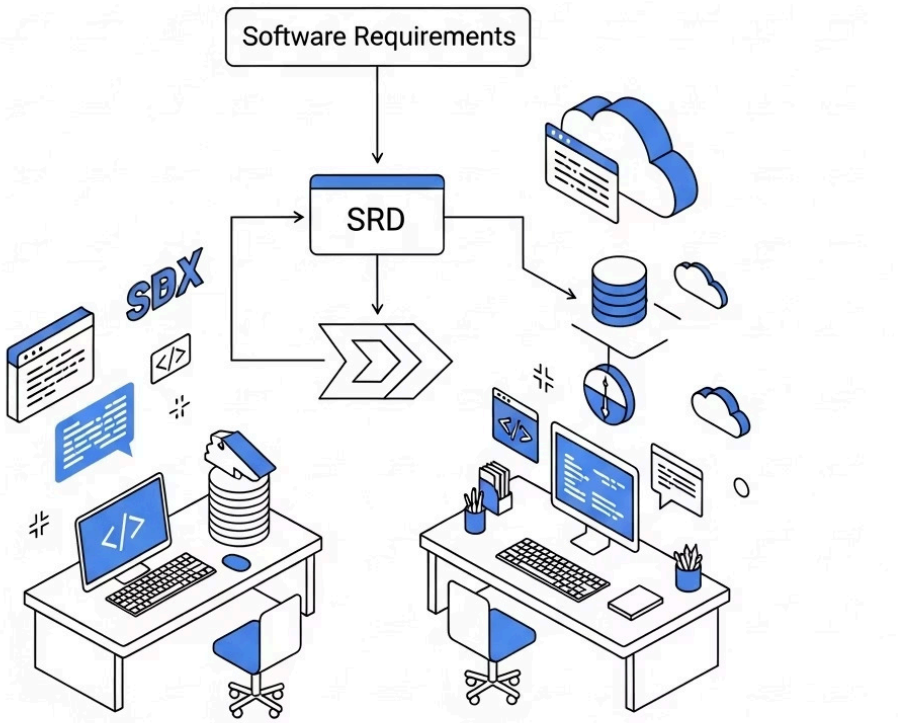
Why SRD Prompting Works Now

AI-Ready Documentation

Modern AI tools can maintain and update the SRD as requirements evolve, keeping it synchronized with development progress.

Downstream Enablement

A well-crafted SRD prompt creates a foundation for all subsequent prompts in the development process, ensuring consistency.



Initial SRD Prompt

Create comprehensive requirements document

Downstream Prompts

Generate architecture, code, tests, and documentation

Evaluation & Feedback

Test against DoD criteria and iterate as needed

SRD Updates

Maintain living document as requirements evolve

Parts of an SRD Prompt

Title

Clear, descriptive name for the project or component

`# Sentiment Analysis API with Dashboard`

Role

The expertise lens the AI should use

`As a senior full-stack developer with NLP expertise`

Context

Background, motivation, and relevant details

`We need to analyze customer feedback across multiple channels...`

Constraints

Technical limitations and requirements

`Must use Python, Node.js ecosystem; deploy to AWS...`

Deliverables

Concrete outputs expected

`REST API, React dashboard, deployment scripts...`

DoD

Specific criteria for completion

85% accuracy, 99% uptime, <2s response time...

What Is an Agent?

AI That Acts

An AI agent is a system that combines multiple capabilities to perform tasks autonomously on behalf of users.

Reasoning

Makes decisions based on context and goals, planning multi-step actions to solve problems.

Tool Use

Leverages external systems like APIs, databases, and file systems to accomplish tasks.

Memory

Maintains context across interactions and retains important information for future use.

Prompting

Uses sophisticated prompting techniques to guide LLM behavior and improve outcomes.

Common agent types include planners (breaking down tasks), RAG agents (retrieving and answering from documents), retrievers (finding information), and file editors (modifying code and content).

Key Design Considerations

- **Performance:** Optimize for low latency and high throughput, especially for real-time interactions.
- **Scalability:** Employ modular, stateless components and cloud-native patterns to handle varying loads.
- **Security:** Implement robust authentication, authorization, data encryption, and regular vulnerability assessments.
- **Pitfalls:** Avoid over-engineering, insufficient error handling, and neglecting data privacy from the start.
- **Decision Framework:** Use a structured approach considering trade-offs between cost, complexity, and desired outcomes.

Agent Architecture Best Practices

Building an AI Agent: A Step-by-Step Guide

Step 1: Define the Task and Environment

Clearly articulate the agent's purpose (e.g., customer support automation) and its operational context (e.g., web app integration). This aligns with business goals and identifies critical constraints like data privacy or real-time requirements.

Key Tools for Prototyping

- **LangChain/LangGraph:** Modular frameworks for defining tasks via prompts (free/open-source).
- **AgentGPT/AutoGPT:** Browser-based platforms for quickly simulating agent behaviors without code (open-source).
- **n8n:** No-code workflow builder for environment mapping (free tier available).

Core Technologies

- **Basic ML:** Initial task scoping with decision trees.
- **NLP:** Understanding user intents via intent recognition models.

Best Practice

Use diagramming tools like Miro or Notion to effectively visualize and map out agent tasks and workflows for clarity and collaboration.

Step 2: Form the Right AI Development Team

The composition of your AI development team should align with the project's scale. Solo developers can manage smaller projects with no-code tools, while larger initiatives necessitate multidisciplinary teams.

Technologies Involved

- **Collaborative ML platforms:** For team training and shared model development.
- **NLP tools:** Crucial for effective communication and understanding within the agent system.

Tools Available

CrewAI

A framework for multi-agent collaboration, enabling simulation of team roles (e.g., dedicated agents for data or ML tasks).

AutoGen (Microsoft)

An open-source framework for robust agent orchestration, ideal for multi-agent systems that mimic team structures.

Semantic Kernel

An enterprise framework for integrating AI capabilities into .NET/Python applications, facilitating team workflows (free for developers).

Outsourcing Options: Platforms such as Prismetric or Upwork to hire specialized ML engineers and data scientists.

Step 3: Gather and Prepare Data

Collect diverse datasets (internal, external, user-generated) and preprocess them (cleaning, labeling) to train the agent effectively.

Technologies Involved

- **Data Labeling**: For annotation of datasets.
- **Machine Learning (ML)**: For preprocessing (e.g., scikit-learn for feature extraction).
- **Natural Language Processing (NLP)**: For text handling (e.g., tokenization).

Tools Available

- **LabelStudio or Prodigy**: Open-source/free data labeling tools for tagging sentiments, intents, or topics.
- **Pandas/Numpy (Python libraries)**: Essential for data cleaning; integrated in most frameworks.
- **Hugging Face Datasets**: A free hub for sourcing and accessing preprocessed NLP datasets.
- **Composio**: A flexible tool praised for its ease of use in integrating data within agents.

Step 4: Choose the Right AI Tech Stack

Selecting the optimal tech stack involves choosing programming languages, core AI technologies (like ML, NLP, and RPA), and scalable solutions tailored to your agent's specific tasks.

Technologies Involved

- **Machine Learning (ML)**: e.g., TensorFlow/PyTorch for model development.
- **Natural Language Processing (NLP)**: e.g., spaCy or BERT models for text understanding.
- **Computer Vision**: e.g., OpenCV, if visual processing is needed.
- **Robotic Process Automation (RPA)**: for task automation.

Tools Available

- **Python with LangChain**: A leading choice for ML/NLP development, offering scalability with cloud platforms like AWS/Google Cloud.
- **Zapier Central**: A no-code solution ideal for integrating applications and automating tasks via RPA, with a free tier available.
- **IBM WatsonX.ai or OpenAI Agents SDK**: Enterprise-grade platforms for advanced NLP and LLM (Large Language Model) applications.
- **Tavily/Exa**: AI-native search tools designed for efficient external data fetching.

Step 5: Design the AI Agent

Define architecture (modular/concurrent), data handling, and UX (e.g., conversational interfaces).

Technologies Involved

Machine Learning for architecture (e.g., neural networks)

Natural Language Processing for response generation

Data Labeling for refining intents

Tools Available

Langflow or Flowise: Low/no-code visual designers for agent architecture; open-source.

Botpress: Beginner-friendly for UX design in chatbots; free.

CopilotKit: For integrating agents into apps with LangGraph; quick setup.

BaseAI: Open-source framework for local-first design.

Step 6: Test the AI Agent

Conduct unit, integration, functional, usability, and edge-case testing to ensure reliability.

Technologies Involved

Machine Learning Evaluation: Metrics like accuracy via scikit-learn for performance.

Natural Language Processing: For sentiment and linguistic testing.

Tools Available

LangSmith (LangChain): For debugging, evaluations, and tracking agent performance.

Pytest or unittest (Python): Standard frameworks for unit and integration testing.

Make or n8n: No-code platforms for simulating diverse scenarios and workflows.

Best Practice: Implement robust logging in CrewAI for effective monitoring and debugging.

Step 7: Deploy AI Agent and Monitor Performance

Integrate the AI agent into existing systems, implement robust security measures, and continuously monitor key performance metrics such as accuracy and response time.

Technologies Involved

MLOps

For efficient deployment and management of AI agents.

NLP

For comprehensive feedback analysis and agent improvement.

Tools Available

Langbase

A serverless platform designed for deployment with memory/tools, enabling easy scaling.

Vercel or Heroku

Popular choices for hosting Python-based agents, often with free tiers available.

Prometheus/Grafana

Open-source tools for comprehensive monitoring, with seamless integration for AWS environments.

Sentry

An essential tool for real-time error tracking and debugging critical for post-deployment.

These steps are adaptable for various development approaches, from no-code platforms like Zapier to more code-intensive solutions. When deploying, prioritize ethical considerations such as data security (e.g., GDPR compliance) and transparency, as highlighted in IBM's 2025 guide. Begin with small-scale deployments, iterate based on performance, and leverage community support from resources like r/AI_Agents.

Top Tools, Use Cases, and Next Steps

Popular Builders

Explore these platforms for developing AI agents:

- **Composio, AgentGPT, AutoGPT** (free/open-source)
- **No-Code**: n8n, Momen AI

Key Use Cases

AI agents are transforming various sectors:

- **Voice Agents** (e.g., Siri)
- **Customer Service** (e.g., Eno)
- **E-Commerce** (e.g., Amazon Recommendations)
- **LLM-Based** (e.g., GitHub Copilot)

Conclusion & Next Steps

AI agents drive innovation. Prioritize ethical considerations and engage with communities for continued learning.

- **Prioritize ethics** in development
- **Engage** with communities like r/AI_Agents

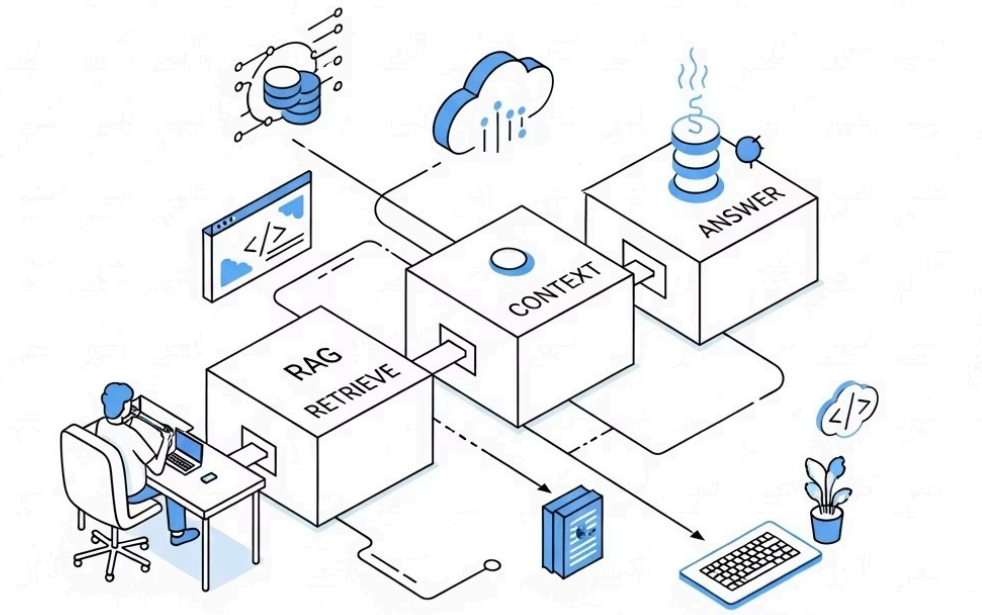
What is RAG?

Retrieve-Then-Answer (RAG)

RAG (Retrieval-Augmented Generation) is a technique that enhances LLM responses by retrieving relevant information from a knowledge base before generating an answer.

This approach grounds AI responses in facts from your documents, reducing hallucinations and improving accuracy.

> The key advantage of RAG is that it grounds every answer in source content, preventing the AI from making things up.



Retrieve

Find relevant information from your knowledge base

Context

Add retrieved information to the prompt

Answer

Generate response based on query and retrieved context

Vector DBs & Chunking

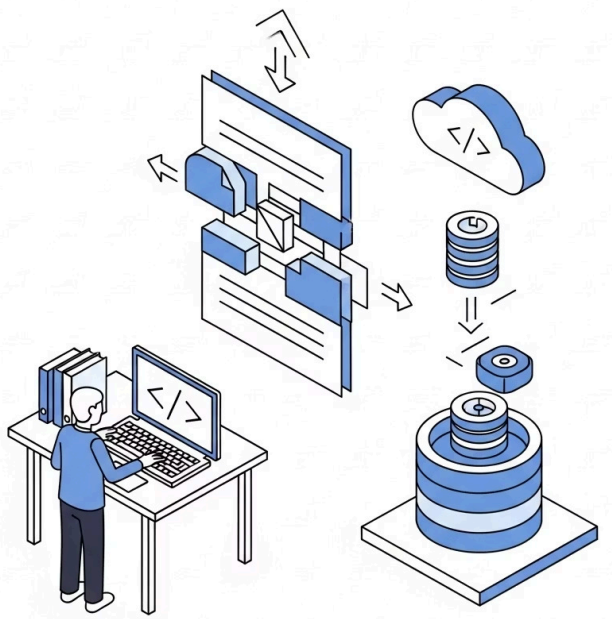
Store Docs as Embeddings

Vector databases are specialized storage systems that enable semantic search by storing document fragments as numerical representations (embeddings).

The RAG Process:

1. **Chunking**: Break documents into manageable pieces
2. **Embedding**: Convert chunks to vector representations
3. **Indexing**: Store vectors for efficient retrieval
4. **Similarity Search**: Find relevant chunks for queries
5. **Context Assembly**: Provide top-K chunks to the LLM

Key Insight: Chunk size and overlap are critical tuning parameters that significantly impact retrieval quality and context relevance.



n8n for Workflows

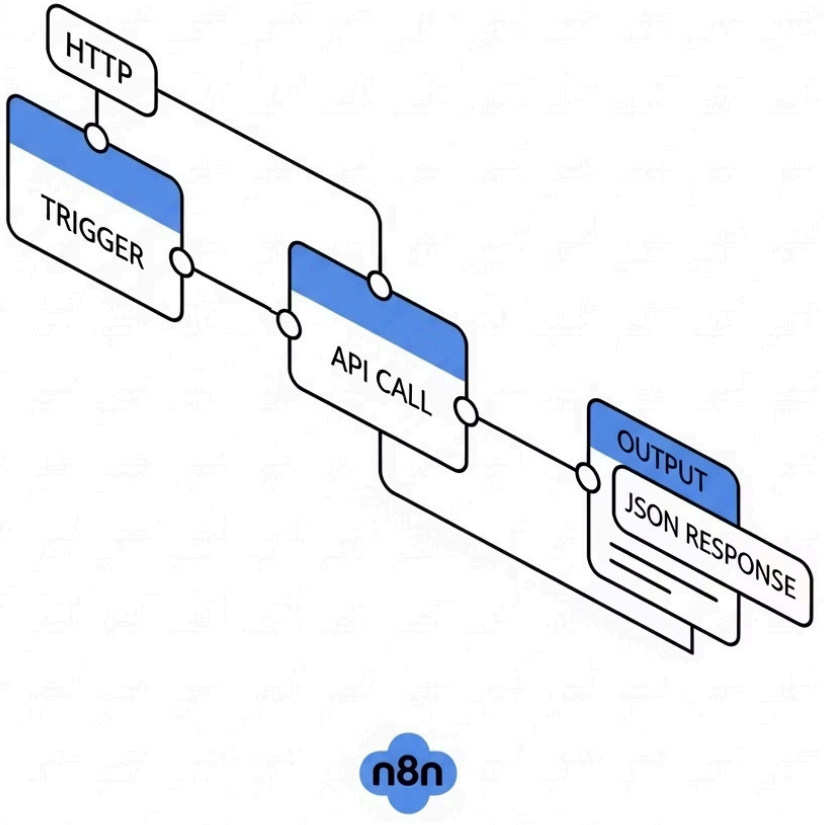
Low-Code Agent Flow

n8n is a powerful workflow automation platform that can orchestrate complex AI pipelines with minimal coding.

Key Benefits:

- Visual workflow design with drag-and-drop nodes
- Easy integration with APIs, databases, and services
- Built-in error handling and retry mechanisms
- Support for conditional logic and data transformations
- Self-hostable for privacy and compliance

Think of it as "Zapier for developers" – powerful enough for complex logic but accessible enough for rapid prototyping.



Prompt to Plan Conversion

From SRD to Actionable Tasks

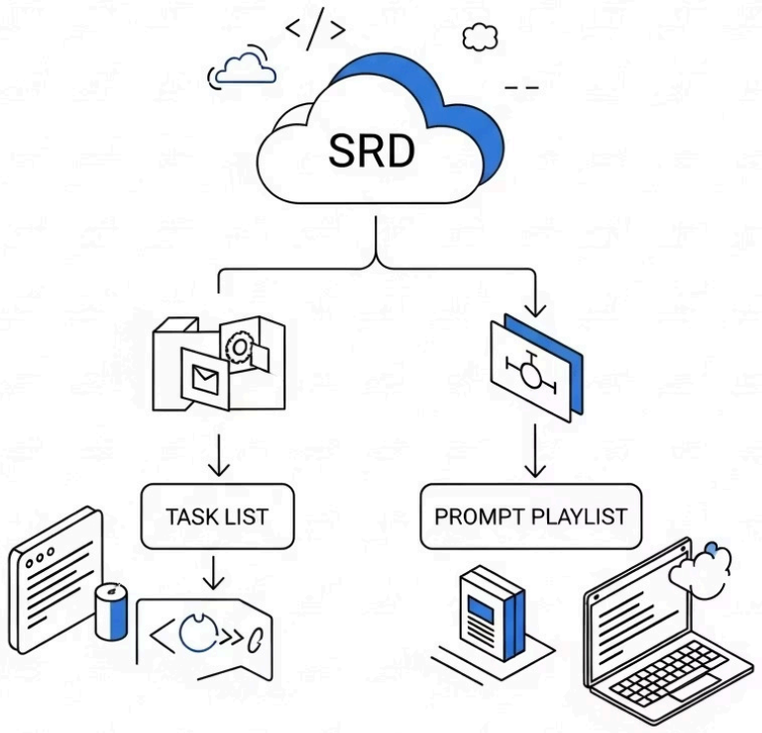
A well-structured SRD prompt can be automatically converted into:

Task List

- Detailed development tasks with dependencies
- Time estimates and resource requirements
- Prioritization based on technical needs

Prompt Playlist

- Sequence of targeted prompts for each stage
- Chained outputs that feed into subsequent steps
- Validation checks between prompt stages



This conversion process creates a bridge between high-level requirements and concrete implementation steps, making complex projects more manageable.

Eval Sets for Agents

Evaluation sets are curated collections of questions and expected answers used to test the accuracy and reliability of AI agents.

Creating Effective Eval Sets:

1. Develop 20+ representative Q&A pairs covering key use cases
2. Include edge cases and potential failure modes
3. Define clear scoring criteria (exact match, semantic similarity, etc.)
4. Implement automated evaluation pipeline
5. Track performance across model and parameter changes

Eval sets provide an objective measure of agent quality and help identify areas for improvement in your RAG implementation.

Guardrails & Safety

Implementing guardrails is essential for creating reliable and safe AI agents, especially in production environments.

Key Guardrail Types:

- **Token Limits**: Prevent runaway costs and timeouts
- **"No Answer" Protocol**: Clear handling for out-of-scope queries
- **Content Filtering**: Block inappropriate inputs/outputs
- **Comprehensive Logging**: Track all interactions for audit
- **Human Fallback**: Escalation path for uncertain cases

Well-designed guardrails protect both users and systems while improving the overall reliability of AI applications.

Demo – Docs Q&A Agent

Building a /ask Endpoint with Citations

We'll create a complete RAG system that can answer questions about your documentation with accurate source citations.

Step-by-Step Overview

Document Collection

Gather technical documentation, guides, knowledge base articles, and other relevant content.

Preprocessing & Chunking

Clean documents, remove formatting artifacts, and split into semantic chunks with appropriate overlap.

Embedding Generation

Convert each chunk into a vector embedding using a suitable embedding model (e.g., OpenAI, BERT, Sentence Transformers).

Vector Database Storage

Store embeddings in a vector database (e.g., Pinecone, Weaviate, Qdrant) with metadata including source information.

Query Processing

Convert user questions to embeddings and perform similarity search to retrieve relevant chunks.

Response Generation

Prompt LLM with question and retrieved context to generate answers with citations to source material.

Prompt Playlist (P0–P10)

A sequence of specialized prompts that work together to create our RAG system:

1. **P0: SRD Creation**
   Define overall requirements and architecture for the Q&A system
2. **P1: Document Processor**
   Design document loading and text extraction pipeline
3. **P2: Chunking Strategy**
   Determine optimal chunk size and overlap parameters
4. **P3: Embedding Configuration**
   Select embedding model and dimension settings
5. **P4: Vector DB Setup**
   Configure vector database schema and indexing parameters
6. **P5: Query Reformulation**
   Design prompt for query understanding and expansion
7. **P6: Retrieval Strategy**
   Implement top-K retrieval with re-ranking
8. **P7: Answer Generation**
   Create prompt template for generating answers with citations
9. **P8: n8n Workflow**
   Design workflow automation for the entire pipeline
10. **P9: API Endpoint**
    Implement REST API for question answering
11. **P10: Evaluation Harness**
    Create testing framework to measure performance

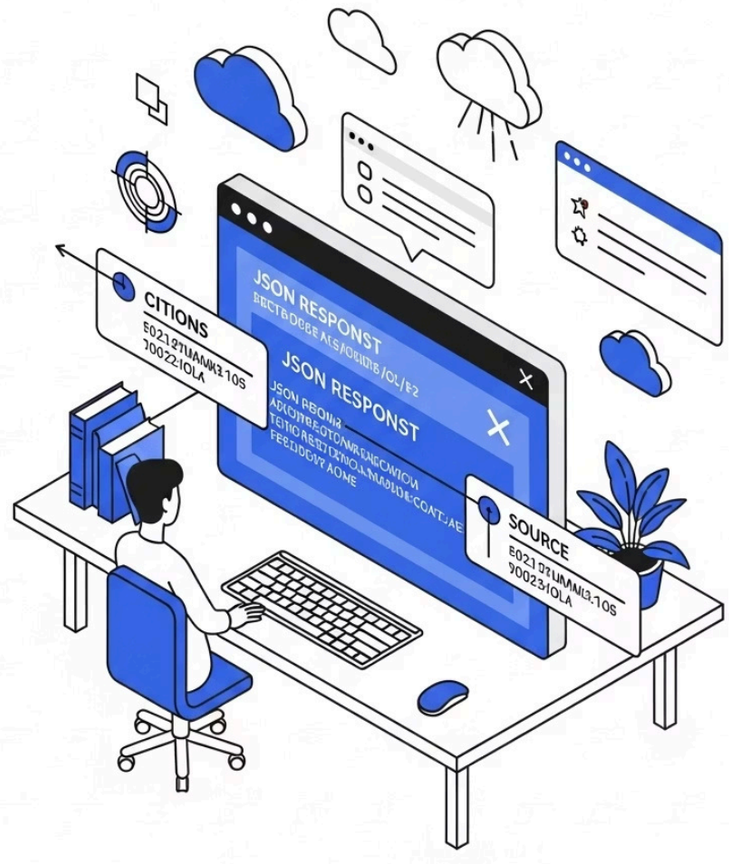
Live Results

Example Query


```
{
  "question": "How do I implement rate limiting in our API?",
  "max_sources": 3
}
```


API Response


```
{
  "answer": "To implement rate limiting in your API, you can use the built-in RateLimiter middleware. Add it to your routes by calling app.use(RateLimiter({ windowMs: 15*60*1000, max: 100 })). This will limit each IP to 100 requests per 15-minute window.",
  "sources": [
    {"title": "API Security Guide", "page": 12},
    {"title": "Middleware Reference", "page": 34},
    {"title": "Best Practices", "page": 8}
  ],
  "confidence": 0.92
}
```

Performance Metrics

- **Accuracy:** 87% (based on eval set)
- **Average Latency:** 1.2 seconds
- **Citation Precision:** 94%
- **No-answer Rate:** 5% (appropriate refusals)

What Makes It Done

Definition of Done Checklist

Accuracy Requirements

- ✅ 85%+ correct answers on evaluation set
- ✅ Less than 5% hallucination rate
- ✅ Citations match source content

Performance Requirements

- ✅ Average response time under 2 seconds
- ✅ Handles 50+ concurrent requests
- ✅ 99% uptime in load testing

Technical Requirements

- ✅ RESTful API with OpenAPI spec
- ✅ Appropriate error handling
- ✅ Comprehensive logging

Mini Lab: Write Your First SRD Prompt

Now it's your turn to practice creating an SRD prompt:

1. Use the template structure we've discussed
2. Define a small-scale project (e.g., a simple API endpoint or utility)
3. Include all essential sections (Role, Context, Constraints, Deliverable, DoD)
4. Focus on making the Definition of Done specific and measurable

Share your SRD with a partner for feedback and refinement.

Mini Lab: Build Eval Set + Score It

Exercise Steps:

1. **Create an Evaluation Set:**
   - Develop 5-10 sample questions about your documentation
   - Write expected "gold standard" answers for each question
   - Include metadata (difficulty, topic, required reasoning)
2. **Build a Simple Scorer:**
   - Use Python or JavaScript to compare generated answers to gold standard
   - Implement basic scoring metrics (exact match, ROUGE, semantic similarity)
   - Calculate precision, recall, and F1 scores
3. **Tune Your RAG System:**
   - Experiment with different chunk sizes (256, 512, 1024 tokens)
   - Try various overlap percentages (0%, 10%, 20%)
   - Adjust top-K retrieved chunks (3, 5, 10)
   - Compare performance across different configurations

Expected Outcomes:

By completing this exercise, you'll gain hands-on experience with:

- Creating effective evaluation criteria
- Measuring RAG system performance
- Understanding the impact of key parameters
- Making data-driven optimization decisions

Recap & Takeaways

SRD = Living Spec

Software Requirements Documents crafted as prompts become the foundation of AI-assisted development, serving as living specifications that evolve with your project.

RAG = Useful Baseline

Retrieval-Augmented Generation provides a solid foundation for knowledge-intensive applications, grounding AI responses in your documentation and reducing hallucinations.

n8n = Glue Logic

Low-code workflow tools like n8n provide the orchestration layer that connects your AI components into coherent systems without excessive custom code.

Prompts = Source of Truth

Well-crafted prompts become the source of truth for your development process, driving consistency across planning, implementation, and evaluation.

The AI-first development approach doesn't replace engineering skills—it amplifies them. By mastering these techniques, you'll be able to tackle more complex problems and deliver higher quality solutions more efficiently.

Next Steps & Resources

Continue Your Learning Journey

Prompt Library Access

Download our curated collection of production-ready prompts for common development tasks and customize them for your needs.

Tool Recommendations

Explore our detailed comparisons and setup guides for the AI development tools discussed in this session.

Open-Source Agent Starter

Clone our GitHub repository with a functional RAG agent template that you can adapt for your own projects.

Community Support

Join our Discord server to connect with other AI-first developers, share insights, and get help with implementation challenges.

Scan to access the resource repository

Upcoming Workshops

- Advanced RAG Techniques – September 15
- Building Multi-Agent Systems – October 3
- AI-First Testing Strategies – October 24

Thank you for participating! Please complete the feedback form to help us improve future sessions.