Build an AI Agent to Automate Your Research

AMAN KHARWAL · NOVEMBER 11, 2025

If you're a student or working in AI, you probably feel overwhelmed by the amount of information out there. The answers exist, but sorting through and making sense of them takes a lot of time. Imagine if you could create an AI agent to handle that first round of research for you. In this guide, we'll build an AI agent to automate research that takes your question, searches the web, reads the top results, and gives you the most relevant passages along with a short summary.

How will our AI Agent for Research Work?

The agent we're building is a fantastic example of a simple **Retrieval-Augmented** system. The core idea isn't just to find pages with the right keywords (like a simple Ctrl+F), but to find passages with the right meaning.

We will use **vector embeddings**. Think of an embedding as a coordinate of meaning in a vast, high-dimensional space. The sentence-transformers library provides models that are experts at turning any piece of text into a list of numbers (a vector) that represents its location in that meaning space.

Our agent's entire job is to:

1. Turn your query into a vector.
2. Search the web, scrape the text, and turn all the content into more vectors.
3. Find the text vectors whose coordinates are closest to your query's coordinates.

That's it. It's just very clever math, and you can build it in about 100 lines of Python.

Now, Let's Build an AI Agent to Automate Your Research

First, create a new Python file and install the dependencies:

```
pip install ddgs requests beautifulsoup4 sentence-transformers numpy
```

Step 1: Import Libraries and Configure Parameters

The top of your file will define all imports and key settings:

```
import re
import urllib.parse
from ddgs import DDGS           # package name is 'ddgs' (d
import requests
from bs4 import BeautifulSoup
from sentence_transformers import SentenceTransformer
import numpy as np
import time
```

Next, configure constants like search results, summary size, and embedding model:

```
SEARCH_RESULTS = 6          # How many URLs to check
PASSAGES_PER_PAGE = 4       # How many passages to pull from
EMBEDDING_MODEL = "sentence-transformers/all-MiniLM-L6-v2"
TOP_PASSAGES = 5            # How many relevant passages to
SUMMARY_SENTENCES = 3       # How many sentences in the fina
TIMEOUT = 8                 # How long to wait for a webpage
```

Step 2: Search and Fetch Web Pages

We'll use **DuckDuckGo Search** for free, API-less search results:

```
def unwrap_ddg(url):
    """If DuckDuckGo returns a redirect wrapper, extract t
```

```
    try:
        parsed = urllib.parse.urlparse(url)
        if "duckduckgo.com" in parsed.netloc:
            qs = urllib.parse.parse_qs(parsed.query)
            uddg = qs.get("uddg")
            if uddg:
                return urllib.parse.unquote(uddg[0])
    except Exception:
        pass
    return url

def search_web(query, max_results=SEARCH_RESULTS):
    """Search the web and return a list of URLs."""
    urls = []
    with DDGS() as ddgs:
        for r in ddgs.text(query, max_results=max_results)
            url = r.get("href") or r.get("url")
            if not url:
                continue
            url = unwrap_ddg(url) # Clean up DDG redirect
            urls.append(url)
    return urls
```

Then, fetch and clean the page with requests and BeautifulSoup:

```
def fetch_text(url, timeout=TIMEOUT):
    """Fetch and clean text content from a URL."""
    headers = {"User-Agent": "Mozilla/5.0 (research-agent)
    try:
        r = requests.get(url, timeout=timeout, headers=hea
        if r.status_code != 200:
            return ""
```

```
        ct = r.headers.get("content-type", "")
        if "html" not in ct.lower(): # Skip non-HTML conte
            return ""

        soup = BeautifulSoup(r.text, "html.parser")

        # Remove all annyoing tags
        for tag in soup(["script", "style", "noscript", "h
            tag.extract()

        # Get all paragraph text
        paragraphs = [p.get_text(" ", strip=True) for p in
        text = " ".join([p for p in paragraphs if p])

        if text.strip():
            # Clean up whitespace
            return re.sub(r"\s+", " ", text).strip()

        # --- Fallback logic if <p> tags fail ---
        meta = soup.find("meta", attrs={"name": "descripti
        if meta and meta.get("content"):
            return meta["content"].strip()
        if soup.title and soup.title.string:
            return soup.title.string.strip()

    except Exception:
        return "" # Fail silently
    return ""
```

Step 3: Chunk, Embed, and Rank Passages

We'll break long articles into smaller passages and embed them using SentenceTransformer:

```
def chunk_passages(text, max_words=120):
    """Split long text into smaller passages."""
    words = text.split()
    if not words:
        return []
    chunks = []
    i = 0
    while i < len(words):
        chunk = words[i : i + max_words]
        chunks.append(" ".join(chunk))
        i += max_words
    return chunks

def split_sentences(text):
    """A simple sentence splitter."""
    parts = re.split(r'(?<=[.!?])\s+', text)
    return [p.strip() for p in parts if p.strip()]

class ShortResearchAgent:
    def __init__(self, embed_model=EMBEDDING_MODEL):
        print(f"Loading embedder: {embed_model}...")
        # This downloads the model on first run
        self.embedder = SentenceTransformer(embed_model)

    def run(self, query):
        start = time.time()

        # 1. Search
        urls = search_web(query)
```

```
        print(f"Found {len(urls)} urls.")

        # 2. Fetch & Chunk
        docs = []
        for u in urls:
            txt = fetch_text(u)
            if not txt:
                continue
            chunks = chunk_passages(txt, max_words=120)
            for c in chunks[:PASSAGES_PER_PAGE]:
                docs.append({"url": u, "passage": c})

        if not docs:
            print("No documents fetched.")
            return {"query": query, "passages": [], "summa
```

Now, initialize the embedding model and compute cosine similarity:

```
# 3. Embed (Turn text into numbers)
        texts = [d["passage"] for d in docs]
        emb_texts = self.embedder.encode(texts, convert_to
        q_emb = self.embedder.encode([query], convert_to_n

        # 4. Rank (Find similarity)
        def cosine(a, b):
            return np.dot(a, b) / (np.linalg.norm(a) * np.

        sims = [cosine(e, q_emb) for e in emb_texts]
        top_idx = np.argsort(sims)[::-1][:TOP_PASSAGES]
        top_passages = [{"url": docs[i]["url"], "passage":
```

Step 4: Generate a Mini Summary

Now, extract and rank sentences within top passages to form a concise summary:

```
        # 5. Summarize (Extractive)
        sentences = []
        for tp in top_passages:
            for s in split_sentences(tp["passage"]):
                sentences.append({"sent": s, "url": tp["ur

        if not sentences:
            summary = "No summary could be generated."
        else:
            sent_texts = [s["sent"] for s in sentences]
            sent_embs = self.embedder.encode(sent_texts, c
            sent_sims = [cosine(e, q_emb) for e in sent_em

            top_sent_idx = np.argsort(sent_sims)[::-1][:SU
            chosen = [sentences[idx] for idx in top_sent_i

            # De-duplicate and format
            seen = set()
            lines = []
            for s in chosen:
                key = s["sent"].lower()[:80] # Check first
                if key in seen:
                    continue
                seen.add(key)
                lines.append(f"{s['sent']} (Source: {s['ur
            summary = " ".join(lines)

```

```
        elapsed = time.time() - start
        return {"query": query, "passages": top_passages,
```

Step 5: Run the Agent

Finally, run the AI agent with any query:

```
if __name__ == "__main__":
    agent = ShortResearchAgent()
    q = "What causes urban heat islands and how can cities

    print(f"Running query: {q}\n")
    out = agent.run(q)

    print("\nTop passages:")
    for p in out["passages"]:
        print(f"- score {p['score']:.3f} src {p['url']}\n

    print("--- Extractive summary ---")
    print(out["summary"])
    print("--------------------------")
    print(f"\nDone in {out['time']:.1f}s")
```

```
Running query: What causes urban heat islands and how can cities reduce them?

Found 6 urls.

Top passages:
- score 0.852 src https://environment.co/what-are-urban-heat-islands-and-how-to-prevent-them/
  heat islands is an urgent environmental goal, as climate change impacts every aspect of human life. Here's a closer look at the causes o
f urban heat islands and how smart city planning can address the...

- score 0.839 src https://environment.co/what-are-urban-heat-islands-and-how-to-prevent-them/
  We are reader-supported. When you buy through links on our site, we may earn affiliate commission. Look at many metro areas from a dista
nce on a warm day, and you'll notice a dome-like atmospheric app...

- score 0.820 src https://earth.org/how-cities-around-the-world-are-tackling-the-urban-heat-crisis/
  As climate change accelerates and urbanisation intensifies, cities worldwide face a growing threat: urban heat. Rising temperatures, exa
cerbated by the urban heat island effect, are endangering public...

- score 0.729 src https://www.wri.org/insights/urban-heat-effect-solutions
  others. The reason for this comes down to the urban environment. Built infrastructure like roads, buildings and sidewalks, as well as na
tural infrastructure like trees and water bodies, determines how...

- score 0.708 src https://environment.co/what-are-urban-heat-islands-and-how-to-prevent-them/
  Fahrenheit. However, they can soar up to 15° to 20° Fahrenheit higher during the worst part of the afternoon. Excessive heat can kill. P
eople who live in urban heat islands run higher risks of heat-re...

--- Extractive summary ---
A primary cause of urban heat islands is the built environment. (Source: https://environment.co/what-are-urban-heat-islands-and-how-to-pre
vent-them/) Other causes of urban heat islands include: According to the EPA, urban heat islands raise temperatures by an average of 1° to
 7° (Source: https://environment.co/what-are-urban-heat-islands-and-how-to-prevent-them/) Here's a closer look at the causes of urban heat
 islands and how smart city planning can address them. (Source: https://environment.co/what-are-urban-heat-islands-and-how-to-prevent-them
```

Output

Final Words

You have just built the core logic of **RAG**, the architecture behind many of the most powerful GenAI systems today. First, you built the **Retriever** (Search, Fetch, Chunk); next, you built the **Ranker** (Embed, Cosine Similarity); and then, you built a simple **Generator** (the extractive summarizer).

The journey from a simple script to a powerful AI system is just a series of small, understandable steps. You just took the first and most important one.

I hope you liked this article on how to build an AI agent to automate your research. Feel free to ask valuable questions in the comments section below. You can follow me on **Instagram** for many more resources.

Aman Kharwal

AI/ML Engineer | Published Author. My aim is to decode data science for the real world in the most simple words.

ARTICLES: 2028

PREVIOUS POST

The 4 Best AI Courses to Take in 2026

NEXT POST

MLOps Project Ideas to Get You Hired

Recommended For You

20 Projects to Master ML for 2026

December 17, 2025

Build an AI Resume Screener with Python & Llama 3

December 16, 2025

The Ultimate 2026 ML Engineering Roadmap

December 15, 2025

Feature Engineering Tricks I Use as an ML Engineer

December 13, 2025 / 1 Comment

Leave a Reply