I Built Pocket Flow, an LLM Framework in just 1(Lines — Here is Why

ZACHARY HUANG
MAR 04, 2025

> *Have you ever stared at a complex AI framework and wondered, "Does it really need t this complicated?" After a year of struggling with bloated frameworks, I decided to stri away anything unnecessary. The result is [Pocket Flow](), a minimalist LLM framework i [100 lines of code]().*

Current LLM Frameworks Are Bloated

For the past year, I've been building AI applications using popular frameworks lik LangChain. The experience has been consistently frustrating:

- **Bloated Abstrac** *helpful at first wh high-level abstrac maintain."* These complexity.

n v Bu rat

- **Implementation Nightmares**: Beyond the abstractions, these frameworks bur developers with dependency bloat, version conflicts, and constantly changing interfaces. Developers often complain: *"It's unstable, the interface constantly cha the documentation is regularly out of date."* Another developer jokes: *"In the time took to read this sentence langchain deprecated 4 classes without updating documentation."*

This led me to wonder: Do we really need so many wrappers? What if we stripped everyt away? What is truly minimal and viable?

Enter Pocket Flow: 100 Lines For the Core Abstraction

After a year of building LLM applications from scratch, I had a revelation: beneath the complexity, LLM systems are fundamentally just **simple directed graphs**. By stripping away the unnecessary layers, I created Pocket Flow — a framework with bloat, zero dependencies, and zero vendor lock-in, all in just 100 lines of code.

	Abstraction	App-Specific Wrappers	Vendor-Specific Wrappers	Lines	Si
LangChain	Agent, Chain	Many (e.g., QA, Summarization)	Many (e.g., OpenAI, Pinecone, etc.)	405K	+16
CrewAI	Agent, Chain	Many (e.g., FileReadTool, SerperDevTool)	Many (e.g., OpenAI, Anthropic, Pinecone, etc.)	18K	+17
SmolAgent	Agent	Some (e.g., CodeAgent, VisitWebTool)	Some (e.g., DuckDuckGo, Hugging Face, etc.)	8K	+19
LangGraph	Agent, Gr				+51
AutoGen	Agent				+26 (core
PocketFlow	Graph				+5(

Comparison of AI system frameworks for abstraction, application-specific wrappers, vendor-specific wrappers, lines of code, and size.

The Simple Building Blocks

Think of Pocket Flow like a well-organized kitchen:

- **Nodes** are like cooking stations (chopping, cooking, plating):

```
class BaseNode:
  def __init__(self):
    self.params,self.successors={},{}
  def add_successor(self,node,action="default"):
self.successors[action]=node;return node
  def prep(self,shared): pass
  def exec(self,prep_res): pass
  def post(self,shared,prep_res,exec_res): pass
  def run(self,shared): p=self.prep(shared);e=self.exec(p);return
self.post(shared,p,e)
```

- **Flow** is the recipe dictating which station to visit next:

```
class Flow(BaseNode):
  def __init__(self,start): super().__init__();self.start=start
  def get_next_node(self,curr,action):
      return nxt=curr.successors.get(action or "default")
  def orch(self,shared,params=None):
      curr,p=copy.copy(self.start),(params or {**self.params})
      while curr:
curr.set_params(p);c=curr.run(shared);curr=copy.copy(self.get_next
ode(curr,c))
  def run(self,shared):
pr=self.prep(
self.post(sha
```


- **Shared store** is t It often an in-mem

```
load_data_node = LoadDataNode()
summarize_node = SummarizeNode()
load_data_node >> summarize_node
flow = Flow(start=load_data)

shared = {"file_name": "data.txt"}
flow.run(shared)
```

In our kitchen (agent system):

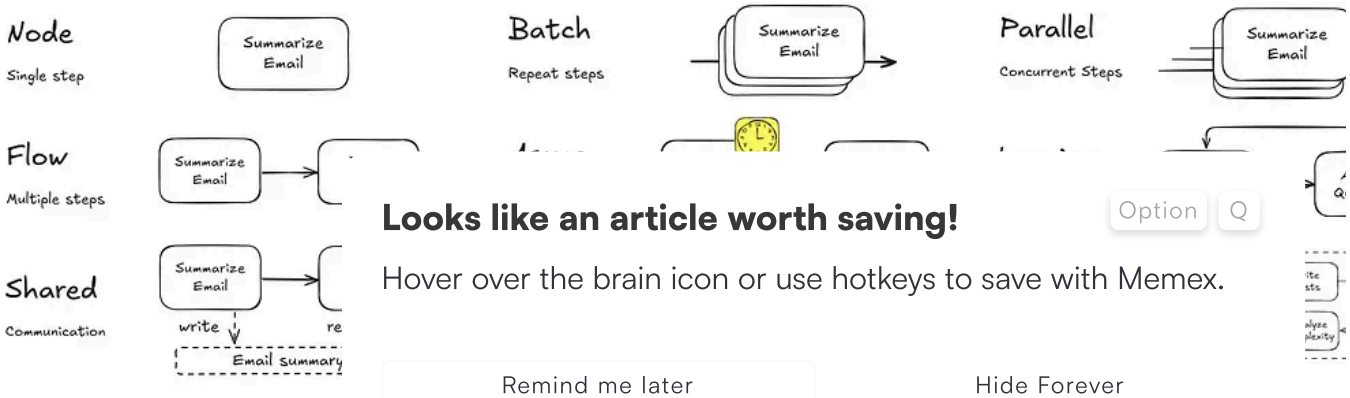
1. Each station (Node) performs three simple operations:

- **Prep**: Retrieve what you need from the shared store (gather ingredients)
- **Exec**: Perform your specialized task (cook the ingredients)
- **Post**: Return results to the shared store and determine next steps (serve the di and decide what to make next)

2. The recipe (Flow) directs execution based on conditions (**Orch**):

- "If vegetables are chopped, proceed to cooking station"
- "If meal is cooked, move to plating station"

We also support batch processing, asynchronous execution, and parallel processin both nodes and flows. And that's it! That's all you need to build LLM applications. unnecessary abstractions, no complex architecture — just simple building blocks t can be composed to create powerful systems.



What About Wrappers Like OpenAI?

Unlike other frameworks, Pocket Flow deliberately avoids bundling vendor-specif APIs. Here's why:

- **No Dependency Issues**: Current LLM frameworks come with hundreds of ME dependencies. Pocket Flow has zero dependencies, keeping your project lean nimble.
- **No Vendor Lock-in**: You're free to use any model you want, including local m like OpenLLaMA, without changing your core architecture.
- **Customized Full Control**: Want prompt caching, batching, and streaming? Bu exactly what you need without fighting against pre-baked abstractions.

What if you need an API wrapper? Just ask models like ChatGPT to write one on-fly. It's usually just 10 lines of code. This approach is far more flexible than rigid b in wrappers or abstractions that quickly become outdated.

```
def call_llm(prompt):
    from openai import OpenAI
    client = OpenAI(api_key="YOUR_API_KEY_HERE")
    r = client.chat.completions.create(
        model="gpt-4o", messages=[{"role": "user", "content": prompt}]
    )
    return r.choices[0].message.content

# Example usage
call_llm("Hello World!")
```

With this minimal b *RA*

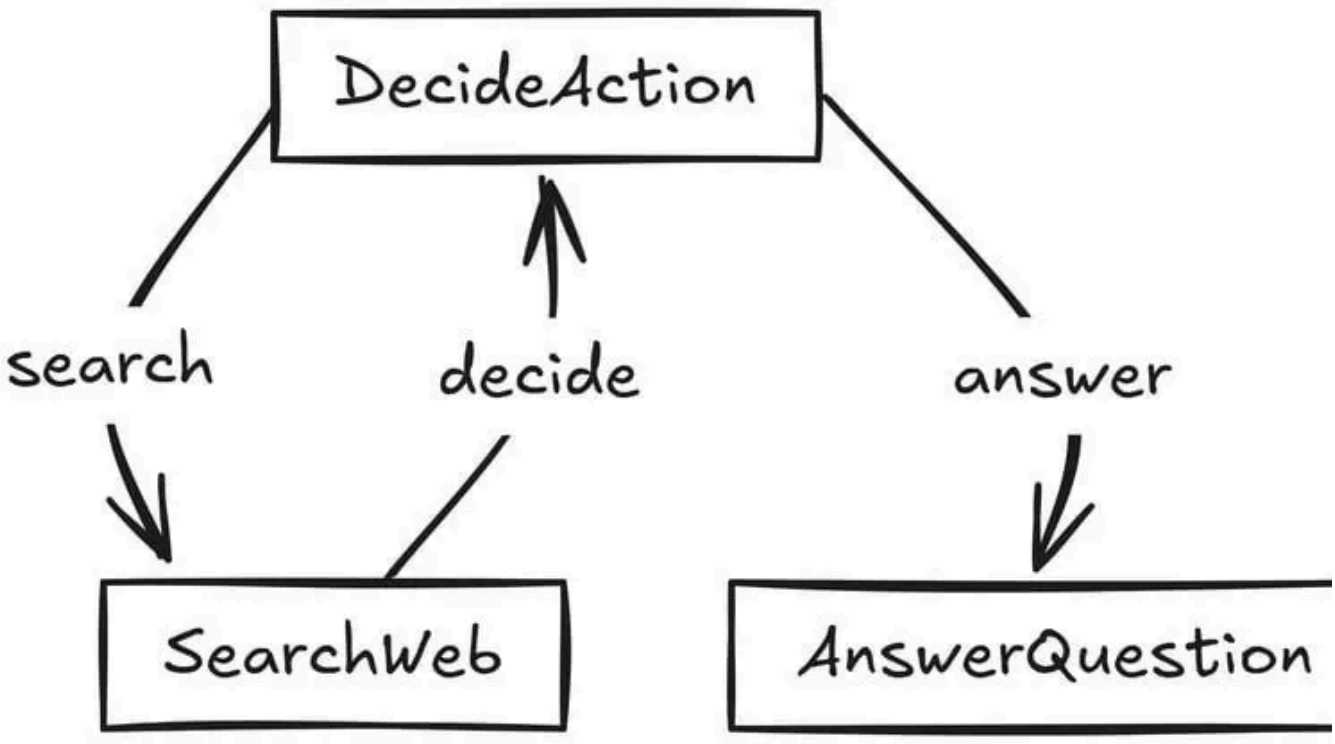
systems, and LLM wo *por*

Let's build a Web Search Agent with Pocket Flow

Let's build a simple [web search agent] using the building blocks from Pocket Flow. Such a simple web search AI agent that can search the web and answer questions similar to tools like Perplexity AI.

The Flow Design

Here's the agent's behavior modeled as a simple flow graph:




```
# Create instances of each node
decide = DecideAction()
search = SearchWeb()
answer = AnswerQu

# Connect the noc
# If DecideActior
decide - "search"

# If DecideAction returns "answer", go to AnswerQuestion
```

```
decide - "answer" >> answer

# After SearchWeb completes and returns "decide", go back to
DecideAction
search - "decide" >> decide

# Create and return the flow, starting with the DecideAction node
return Flow(start=decide)
```

What Happens at Each Node?

1. **DecideAction** — *"Should we search the web, or do we already know enough?"*

- **Prep**: Pulls in the original question and any previous search context from shar memory
- **Exec**: Asks the LLM whether to perform a web search or answer directly
- **Post**: Saves a search query if needed, and returns either `"search"` or `"answe` as the next action

2. **SearchWeb** — *"Let's go fetch some fresh information."*

- **Prep**: Retrieves the query generated in the last step
- **Exec**: Calls a web search API (Google, Bing, etc.), fetches results, and distills t into readable chunks
- **Post**: Adds the search results back into context, then loops back to **DecideAct** for re-evaluation

3. **AnswerQuestion** — *"We've got enough info — let's answer the question."*

- **Prep**: Collects the question and all search context
- **Exec**: Prompts tl
- **Post**: Stores the

The graph is dynami t
LLMs, swap out the search engine, or insert new decision points without ever

breaking the core logic.

Let's Walk Through an Example

Imagine you asked our agent: "Who won the 2023 Super Bowl?" Here's what would happen step-by-step for each node:

1. **DecideAction Node**:

- **LOOKS AT** (**Prep**): Your question and what we know so far (nothing yet)
- **THINKS** (**Exec**): "I don't know who won the 2023 Super Bowl, I need to search
- **DECIDES** (**Post**): Search for "2023 Super Bowl winner"
- **PASSES TO** (**Orch**): SearchWeb station

2. **SearchWeb Node**:

- **LOOKS AT** (**Prep**): The search query "2023 Super Bowl winner"
- **DOES** (**Exec**): Searches the internet (imagine it finds "The Kansas City Chiefs won")
- **SAVES** (**Post**): The search results to our shared countertop
- **PASSES TO** (**Orch**): Back to DecideAction station

3. **DecideAction Node**(**second time**):

- **LOOKS AT** (**Prep**): Your question and what we know now (search results)
- **THINKS** (**Exec**): "Great, now I know the Chiefs won the 2023 Super Bowl"
- **DECIDES** (**Post**): We have enough info to answer
- **PASSES TO** (**Orch**): AnswerQuestion station

4. **AnswerQuestion**

- **LOOKS AT** (**Pre**
- **DOES** (**Exec**): Cr
- **SAVES** (**Post**): The final answer

- **FINISHES** (**Orch**): The task is complete!

And that's it! Simple, elegant, and powered by search. The entire agent implementation requires just a few hundred lines of code, built on our 100-line framework. You can see the complete code and run it yourself using this cookbook

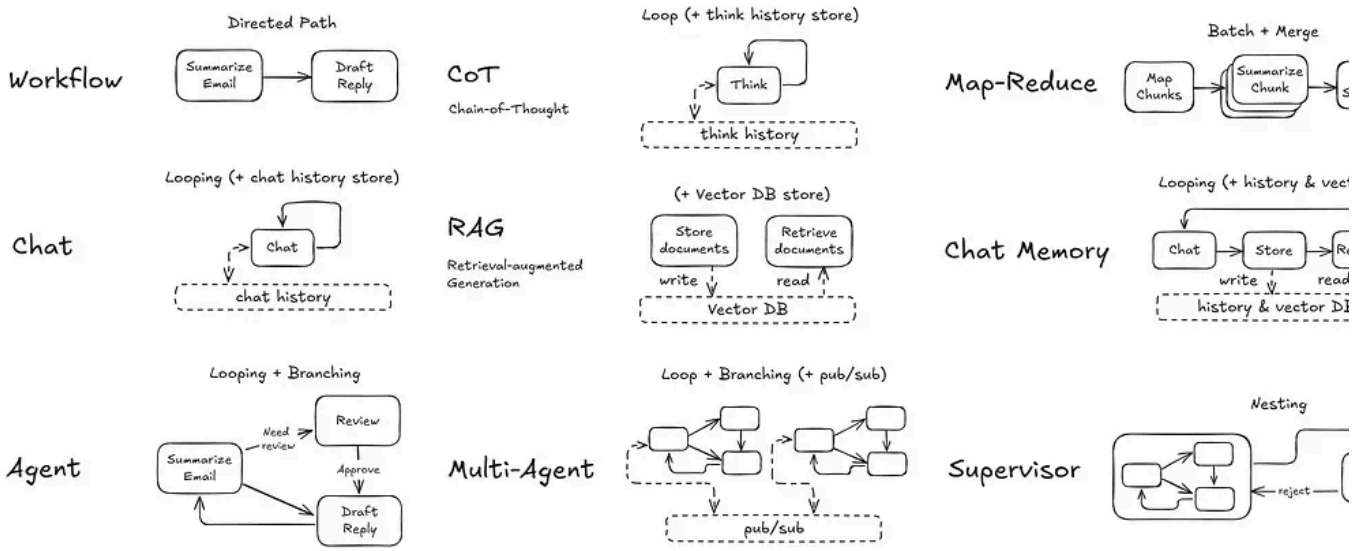
---

This is the essence of Pocket Flow: composable nodes and simple graphs creating smar reactive AI agents. No hidden magic. No framework gymnastics. Just clear logic and com control.

What else can we build?

Pocket Flow isn't limited to search agents. Build everything you love — Multi-Age Workflows, RAG systems, Map-Reduce operations, Streaming, Supervisors, Chat Memory, Model Context Protocol, and more — all with the same elegant simplicit Each implementation follows the same pattern: a few hundred lines of code built c first principles, with our minimal 100-line framework as the foundation.

No unnecessary abstraction. No bloat. Instead of trying to understand a gigantic framework with hundreds of thousands of files, Pocket Flow gives you the fundamentals so you can build your own understanding from the ground up. Find complete tutorials for all these implementations in the Pocket Flow GitHub repos and explore our basic tutorials to get started.



Design Patterns based on Pocket Flow

Future Vision of Pocket Flow: Agentic Coding

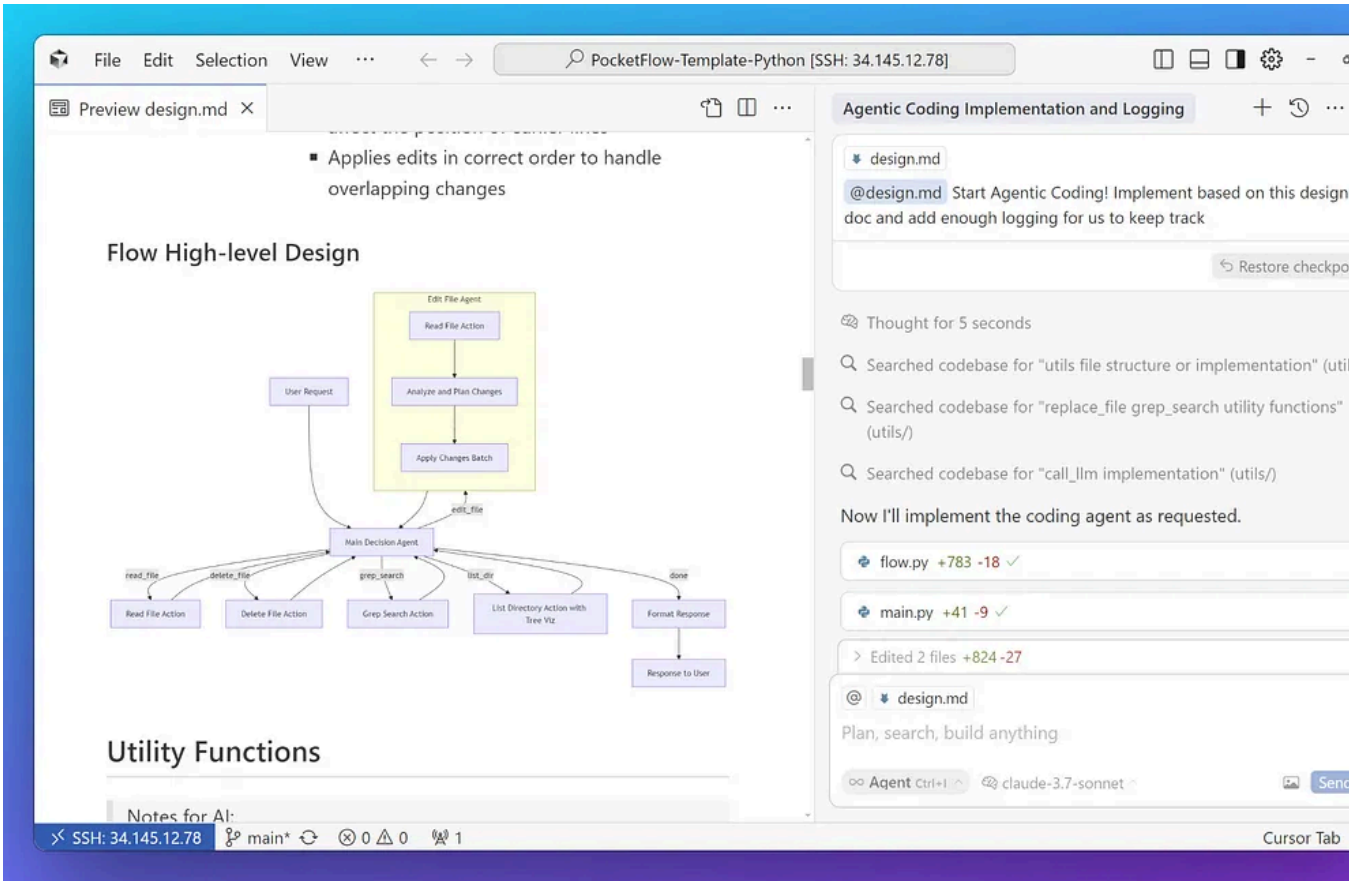
The true power of Pocket Flow extends beyond its minimalist design. Its most revolutionary aspect is enabling [Agentic Coding] — a new way of programming wh AI assistants help you build and modify AI applications.

What is Agentic Coding?

Agentic coding is simply the practice of working alongside AI to build software. T of it like building a house — you're the architect with the vision and expertise, wh the AI is your construction crew handling the detailed work:

- **You** focus on high-level design and strategic decisions (the human strength)
- **The AI assistant** handles implementation details and technical execution (the strength)
- **You** review and

This 10x productivity and more time on cre



Agentic Coding in Action

Teaching AI to Build LLM Applications

How do we teach AI to build powerful LLM applications? Previous frameworks to the wrong approach — they create hard coded wrappers for specific applications l summarization, tagging, and web scraping that end up bewildering both human developers and AI assistants alike.

Our solution is elegantly simple: **Documentation as the second codebase!** Instead hard coded wrappers, vibe code them in documentation. Pocket Flow provides jus lines of core building blocks, paired with clear documentation that teaches how to combine these blocks into powerful applications. We simply provide examples and AI agents implement solutions on the fly. This documentation-as-code approach allows AI assistants t

1. **Master the fund** drowning in frar

2. **Build customized solutions**: Generate implementations perfectly tailored to specific application needs
3. **Focus on architecture**: Think about system design rather than fighting frame limitations

We pass these "instruction manuals" directly to AI assistants as rule files (e.g., .cursorrules for cursor AI), giving them the knowledge to build sophisticated syste from simple components.

For deeper exploration of this approach, visit: Agentic Coding: The Most Fun Wa Build LLM Apps or check out my YouTube channel for more tutorials.

The future vision is even more exciting: as Pocket Flow patterns spread through the devel ecosystem, they'll eventually be absorbed into future LLMs' training data. At that point won't even need explicit documentation — AI assistants will intrinsically understand th principles, making LLM application development truly frictionless.

Conclusion: Simplicity Is the Ultimate Sophistication

Pocket Flow strips away the complexity, offering just what you need: 100 lines of c that model LLM applications as simple directed graphs. No bloat, no magic, just transparent logic and complete control.

If you're tired of framework gymnastics and want to build your understanding fro the ground up, Pocke nts today while preparin

Join our Discord con ck Flow!

Try Pocket Flow today and experience how 100 lines can replace hundreds of thousand

GitHub Repository | Documentation | TypeScript Version

Subscribe to Pocket Flow

By Zachary Huang · Launched 6 months ago

Pocket Flow: 100-line LLM framework for Agentic Coding

Pledge your support

30 Likes · 4 Restacks

Next

Discussion about this post

Comments Restacks

Write a comment...

Leandro Jul 16

Liked by Zachary

I am having fun v … fil
downloads its UI … t I t
got close enough

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

https://github.com/argenkiwi/pocketflow-m3u-downloader

LIKE (2) REPLY

Larry Maloney Jul 18

Liked by Zachary Huang

Thank you for building this. I have avoided that problem. Now that Pocket Flow exists, I will s with it.

LIKE (1) REPLY

11 more comments...