LLM Agent's Arsenal: A Beginner's Guide to th Action Space

ZACHARY HUANG
JUL 01, 2025

> *Ever sent your AI agent into the "battle" of a complex task, only to watch it fumble wi blunt sword or use the wrong weapon for the fight? When an agent fails, our first instir to blame its "brain" (the LLM). But the real culprit is often the arsenal we equipped it —the collection of weapons was dull, confusing, or simply not right for the job.*

In our previous tutorial, LLM Agents are simply Graph — Tutorial For Dummies, revealed that every a s `Strike -> Repea` de plans the next move.

That 'Strike' is powe ls, spells it can draw upon. In technical terms, this is its **Action Space**. This isn't

list of functions; it is the very soul of your agent's power. A well-forged arsenal, wl every blade is sharp and serves a unique purpose, is the difference between an age that is defeated by the first obstacle and one that conquers any challenge.

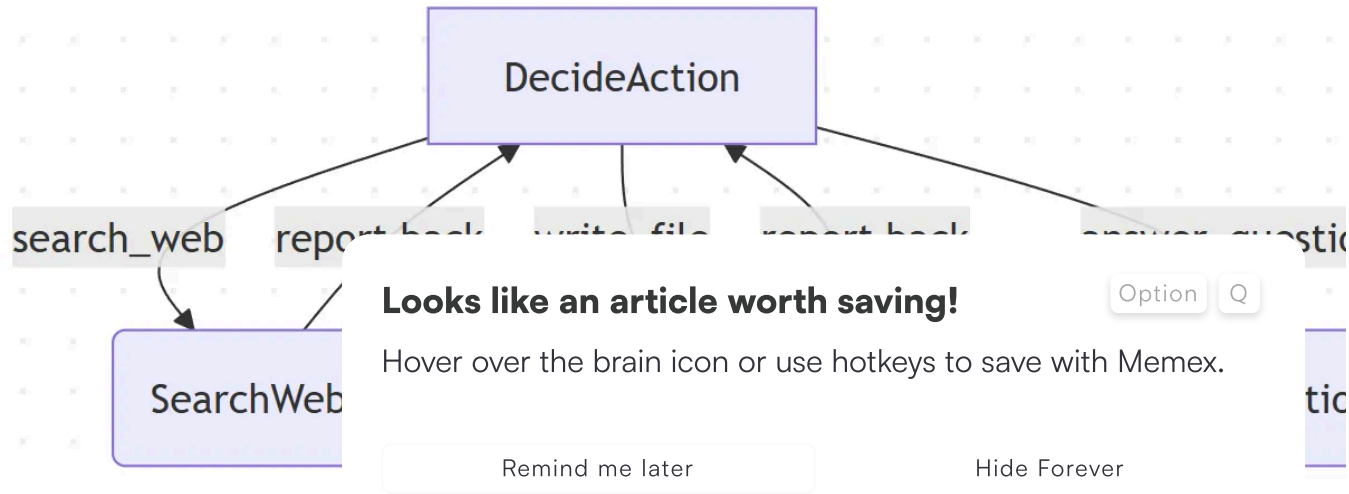
In this guide, you are the master blacksmith. Using the transparent and powerful [PocketFlow] framework as your forge, we will teach you how to craft an arsenal of actions that will turn your agent from a clumsy squire into a legendary warrior.

The Battle Tactician: How an Agent Chooses It Weapon

So, we have an arsenal. But how does the agent, our digital warrior, know when to draw a longsword for a close-quarters fight versus firing a bow from a distance?

This critical decision happens in the `DecideAction` node—the agent's battle tactician. At its core, every agent is just a simple loop that consults its tactician, w then chooses an action from the arsenal. The chosen action is performed, and the results are reported back to the tactician to plan the next move.

Visually, the battle plan looks like this:



In this diagram:

1. `DecideAction` (**The Tactician**): This is the brain. It analyzes the battlefield (user's request and current data).
2. **The Arrows** (**The Commands**): Based on its analysis, the tactician issues a command: `search_web`, `write_file`, or `answer_question`. This is the branch in the graph.
3. **The Action Nodes** (**The Specialists**): Each command goes to a specialist soldi who executes that one task.
4. **The Loop Back** (**The Report**): After the specialist completes their task, they re back to the tactician with new information, and the cycle begins again.

"But what magic happens inside that `DecideAction` node?" you ask. "How does *actually* think?"

This is the most misunderstood part of agent design, and the secret is shockingly simple. **It's just a prompt.** There's no complex algorithm, just a carefully written s instructions for the LLM.

The tactician's "brain" is a prompt that looks something like this:

```
### CONTEXT
You are a research assistant. Here is the current situation:
Question: {the user's original question}
Previous Actions: {a log of what has been done so far}
Current Information: {any data gathered from previous actions}

### ARSENAL (Available Actions)
Here are the weapons you can use. Choose one.

[1] search_web
  Description: Se
  Parameters:
    - query (str)

[2] write_file
  Description: Save text into a local file.
```

```
  Parameters:
    - filename (str): The name of the file to create.
    - content (str): The text content to write into the file.

[3] answer_question
  Description: Provide the final answer to the user.
  Parameters:
    - answer (str): The complete, final answer.

## YOUR NEXT COMMAND
Review the CONTEXT and choose the single best ACTION from your ARSENAL
to proceed.
Format your response as a YAML block.
```

That's it! The agent's entire decision-making process boils down to this: the LLM reads the description of the situation and the "user manual" for every weapon in it arsenal, and then it picks the one that makes the most sense.

The quality of its choice is **100% dependent on how clearly you describe its weap** A sharp, well-defined arsenal in your prompt leads to a smart, effective agent. A va confusing one leads to a warrior who brings a knife to a dragon fight.

Now, let's learn how to forge these weapons, from simple daggers to god-tier mag spells.

Level Up Your Arsenal: The Three Tiers of Weapon Complexity

As a master blacksmith, you wouldn't forge just one type of weapon. You need a fu range, from simple daggers for quick jabs to powerful, enchanted swords for epic battles. The same is true for your agent's arsenal. Actions can be designed with va levels of power and c

Level 1: The Si

A simple dagger is a ar does it reliably. These are actions that require **no parameters**.

Think of them as on/off switches or simple commands.

In the Forge (Code):

An action like `request_human_help` or `finish_task`.

In the Arsenal (Prompt Description):

```
[1] request_human_help
  Description: If you are stuck or need clarification, use this actior
to pause and ask the human user for guidance.
```

When to Use It:

For clear, binary decisions. When the agent needs to signal a state change, like "I'r finished," "I'm stuck," or "I've failed." They are perfect for controlling the overall of the battle plan.

Level 2: The Sharpshooter's Bow (The Parameterized Tool)

A bow is useless without an arrow and a target. This weapon requires input to be effective. These are the most common and versatile actions in an agent's arsenal—actions that require **specific parameters** to function.

To use these weapons, the agent must not only choose the bow but also aim it by providing the correct inputs.

In the Forge (Code):

An action like `search_web(query)` or `send_email(to, subject, body)`.

In the Arsenal (Prom


```
[2] search_web
  Description: Se
  Parameters:
    - query (str): The precise search term to look up. Must be a focus
```

```
string.

[3] send_email
  Description: Composes and sends an email to a recipient.
  Parameters:
    - to (str): The email address of the recipient.
    - subject (str): The subject line of the email.
    - body (str): The main content of the email.
```

The Crucial Link to Your Blacksmithing Skills:
How does the agent provide these parameters? This is where your skill in **structur output** becomes critical. As we covered in our guide, Structured Output for Begin you must instruct the LLM to format its response in a structured way (like YAML JSON) so your program can easily parse the action *and* its parameters.

Without this skill, you've given your agent a powerful bow but no way to nock an arrow.

Level 3: The Spellbook of Creation (The Programmable Action)

This is the ultimate weapon: a spellbook that doesn't contain a list of spells but teaches the agent how to *write its own*. These are **programmable actions** where the agent generates code or complex instructions on the fly.

This gives the agent god-like flexibility to solve novel problems you never explicitl trained it for.

In the Forge (Code):
An action like `execute_sql(query)` or `run_python_code(code)`.

In the Arsenal (Pron


```
[4] execute_sql
  Description: Wı                                                    s
database. The database contains tables named 'customers', 'orders', ar
'products'.
```

```
  Parameters:
    - sql_query (str): A valid SQL query string to execute.

[5] run_python_code
  Description: Write and execute a sandboxed Python script for complex
calculations, data manipulation, or interacting with APIs.
  Parameters:
    - code (str): A string containing the Python code to run.
```

The Power and the Peril:
A spellbook is the most powerful weapon in your arsenal, but it's also the most dangerous.

- **Power:** Your agent can solve almost any problem that can be expressed in cod It's no longer limited to pre-defined tools.
- **Peril:** It's much more likely to make a mistake (e.g., writing buggy code). More importantly, it opens up massive security risks if not handled carefully (e.g., executing malicious code like `os.remove("important_file.txt")`). Alw run such code in a secure, sandboxed environment.

Mastering these three tiers allows you to build a balanced and effective arsenal, equipping your agent for any challenge it might face.

Forging the Perfect Arsenal: 3 Golden Rules fo Your Weapon Inventory

A legendary warrior doesn't just carry a random assortment of weapons. Their ars is carefully curated—each item is perfectly crafted, serves a distinct purpose, and instantly accessible. As the master blacksmith for your agent, you must apply the s discipline. Here are t e.

Golden Rule # (Clarity is Kin

The descriptions for your actions and their parameters are not notes for yourself; t are the **user manual for the LLM**. If the manual is vague, the LLM will misuse the tool. Be painfully, relentlessly explicit.

A Dull Blade (Bad Description):

```
search: searches for stuff
```

The agent sees this and thinks, "What stuff? How? What do I provide?" The result wild guess.

A Sharpened Katana (Good Description):

```
search_web(query: str):
  Description: Searches the public internet for up-to-date information
on a single, specific topic. Returns the top 5 text snippets.
  Parameters:
    - query (str): A simple and focused search query, typically 3-5
words long.
```

Now the agent understands its constraints. It knows the tool is for *one topic* and th query should be *short*. It will correctly generate a command like `search_web(query: "2024 Nobel Prize Physics winner")`, leading to much better outcome.

Golden Rule #2: Don't Burden Your Warrior with a Junk Drawer (Keep it Concise)

A warrior grabbing a hundred options. Th too many actions lea and a higher chance

> **The Blacksmith's Guideline:** An arsenal of **10 weapons is formidable. An arse of 100 is a junk drawer.**

If your action space is growing too large, it's a sign that your tools are too granula Instead of creating `read_json_file`, `read_csv_file`, and `read_text_file` forge a single, more powerful weapon: `read_file(filename: str)`. Your cod handle the internal logic of parsing different file types. Keep the agent's choices cl and high-level.

Golden Rule #3: Make Every Weapon Unique (Slay Redundancy)

Every weapon in the arsenal should have a unique purpose. If the agent has two to that do similar things, it will get confused about which one to use. This is called a of "orthogonality."

The Confusing Arsenal (Bad Design):

- `read_csv_from_disk(file_path: str)`: Reads customer data from a lo CSV file.
- `query_database(sql: str)`: Queries the live customer database.

The agent is asked to "find the total sales for new customers from this quarter." W tool should it use? The data might be in the CSV, or it might be in the database. T agent doesn't know and might make the wrong choice.

The Pro-Gamer Move: Simplify the Battlefield
A true master blacksmith doesn't just forge weapons; they shape the battlefield to their advantage. Instead of giving the agent two ambiguous tools, do the work for behind the scenes.

The Decisive Arsena
Before the agent ever **table in the database**

Now, the agent's arsenal is clean and unambiguous:

- `query_database(sql: str)`: Queries the customer database, which conta all known customer data.

The ambiguity is gone. The agent has one, and only one, tool for retrieving custom data. You've eliminated redundancy and made the agent's decision trivial, guaranteeing it makes the right choice every time.

Conclusion: An Agent is Only as Sharp as its Arsenal

And so, the secrets of the forge are yours. You now understand that the true powe an LLM agent doesn't come from some mysterious, hidden algorithm. It comes fr the thoughtful, disciplined, and creative process of crafting its **Action Space**.

You've learned that agents are just warriors in a **loop with branches**, making decis based on a prompt that serves as their battle plan. And you've seen how to stock t arsenal for any challenge:

- With **Simple Daggers** for quick, decisive commands.
- With **Sharpshooter's Bows** for precise, targeted actions.
- With reality-bending **Spellbooks of Creation** for ultimate flexibility.

Most importantly, you now hold the three golden rules of the master blacksmith:

1. **Engrave a Clear Manual:** Your descriptions are the agent's guide to victory.
2. **Avoid the Junk Drawer:** A curated, concise arsenal is deadlier than a clutterec one.
3. **Slay Redundancy:** Make every weapon unique to ensure the agent never hesit

The next time you se de won't be intimidated al questions: "What's ir unique?"

Armed with this knowledge, you are no longer just a coder; you are an **agent blacksmith**. You have the power to forge not just tools, but intelligent, reliable, an effective digital warriors.

Ready to light the forge? Dive into the code and explore these principles in action by chec out [PocketFlow on GitHub](#)!

← Previous Next

Discussion about this post

I have a small question about Golden Rule #2: Don't Burden Your Warrior with a Junk Drawe it Concise).

Does the recommended number of tools assigned to an agent/model depend on the scale o local model being used (for example, Qwen-3B vs. Qwen-8B vs. Qwen-32B)?

In other words, do larger models handle a bigger arsenal of tools more effectively and accur

In my real-world scenario, there are many different APIs and functions that need to be integ So my "weapon arsenal" might easily exceed 10 tools. Would a larger model help manage th complexity better? Or is it always recommended to keep the toolset as concise as possible, regardless of model size?

LIKE (1) REPLY

1 reply by Zachary Huang

Everaldo Gomes Jul 3

Liked by Zachary Huang

Congrats, Zachary! Great article, great explanation, as always. Let me share some of my exp I never used Cursor for AI Coding, I used to go with CLINE/RooCode. But 2 weeks ago I have changed to Claude Code, since their plan is the most affordable and their agent is incredible the Agent researched an issue for 5 minutes in the Web and got a correct response). Also, C costs only $17 and is kind of unlimited (limited unlimited - after around 2 full context windov have to await a couple hours to use it again). Also, I will try the new Coding Agent: "Open Cc Everyone says it is great and also works with Claude PRO. And I'm willing to try a new tool fr dagger.io team/community called "container use". https://github.com/dagger/container-use

I hope this can be useful.

Best Regards

LIKE (1) REPLY

2 more comments...