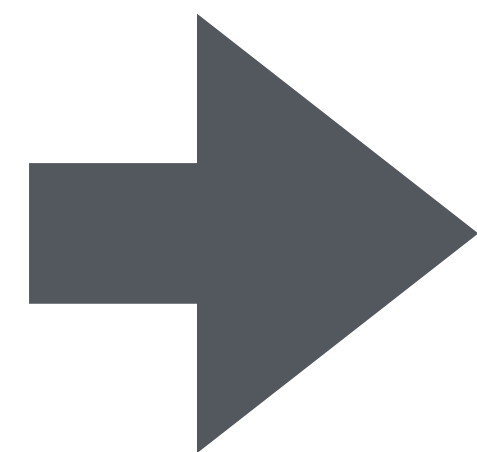


Tool Use

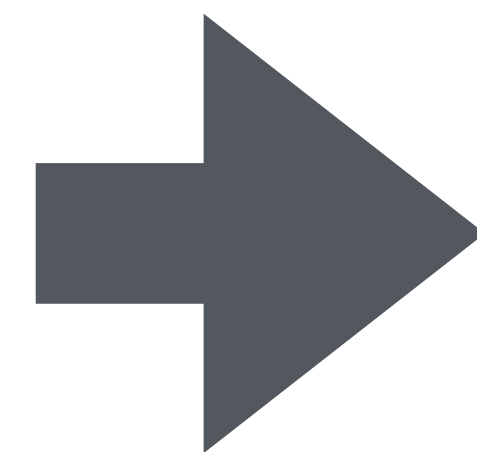
Full Picture

Basic LLM

Pre-training



Instruction
tuning



RLHF / DPO

A demo

What went wrong?

- LLM has no information to help answer the question
- LLM is bad at math
 - similar to humans
- Any of us could write some simple python code to find answer

```
> ollama run llama3.1
```

```
>>> In one word: is it morning?  
Yes.
```

```
>>> In one word: is it night?  
No.
```

```
>>> In one word: is it noon?  
Yes.
```

```
>>> In one word: is it afternoon?  
Yes.
```

```
ollama run llama3.1
```

```
>>> What is 54321 + 12345  
54321 + 12345 = 66766
```

A demo

LLMs that code

- Ask LLM to produce python code to compute answer
 - Why does this work?

```
ollama run llama3.1
```

```
>>> In one word: is it morning? Feel free to write a python program to answer.
```

```
**No**
```

Here's a simple Python program that checks the current time and prints whether it's morning (6am-11am):

```
```python
import datetime

now = datetime.datetime.now()
hour = now.hour

if 6 <= hour < 12:
 print("Morning!")
else:
 print("Not Morning")
```
```

You can run this code in a Python interpreter or save it to a file and run it as a script.

LLMs that code

- Ask LLM to produce python code to compute answer
 - Why does this work?

```
ollama run llama3.1
```

```
>>> What is 54321 + 12345. Feel free to write a python program to answer.
```

```
**Calculating the Sum of Two Numbers**
```

You can use Python to calculate the sum of two numbers, which in this case are `54321` and `12345`. Here's a simple program that does just that:

```
```python
Define variables for the two numbers
num1 = 54321
num2 = 12345

Calculate the sum
result = num1 + num2

Print the result
print(f"The sum of {num1} and {num2} is: {result}")
```
```

```
**Output:**
```

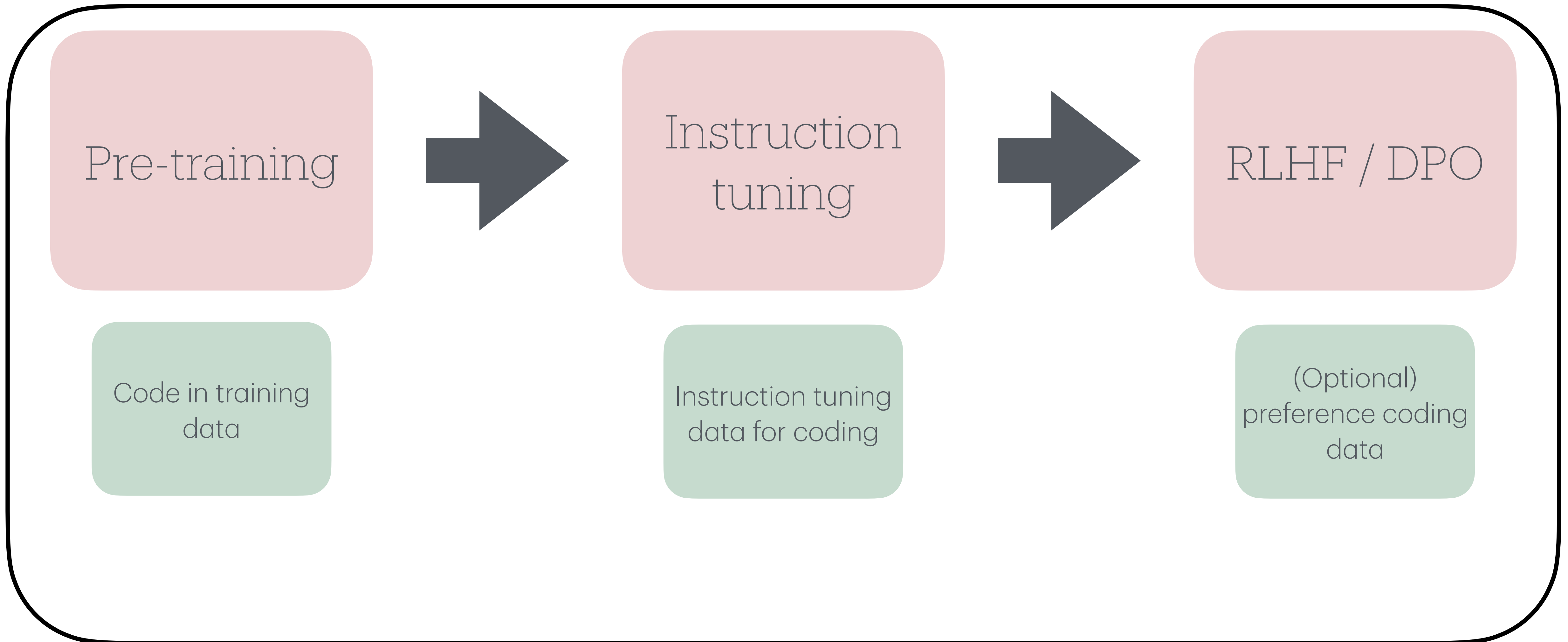
```
...
```

```
The sum of 54321 and 12345 is: 66666
```

```
```
```

# Full Picture

Basic LLM





# LLMs that code

- Ask LLM to produce python code to compute answer
- Model might not know how to use answer
- Not very secure
- All of python might be too broad

```
ollama run llama3.1
```

```
>>> In one word: is it morning? Feel free to write a python program to answer.
```

```
No
```

Here's a simple Python program that checks the current time and prints whether it's morning (6am-11am):

```
```python
import datetime

now = datetime.datetime.now()
hour = now.hour

if 6 <= hour < 12:
    print("Morning!")
else:
    print("Not Morning")
```
```

You can run this code in a Python interpreter or save it to a file and run it as a script.

# LLMs that code

- Ask LLM to produce python code to compute answer
- **Model might not know how to use answer**
- Not very secure
- All of python might be too broad

Fine-tune LLM with coding chat template

```
<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>
```

```
Environment: ipython<|eot_id|>
<|start_header_id|>user<|end_header_id|>
```

```
Write code to check if number is prime, use that to
see if the number 7 is prime<|eot_id|>
```

```
<|start_header_id|>assistant<|end_header_id|>
```

```
<|python_tag|>def is_prime(n):
 if n <= 1:
 return False
 for i in range(2, int(n**0.5) + 1):
 if n % i == 0:
 return False
 return True
```

```
print(is_prime(7)) # Output: True<|eom_id|>
```

```
<|start_header_id|>ipython<|end_header_id|>
```

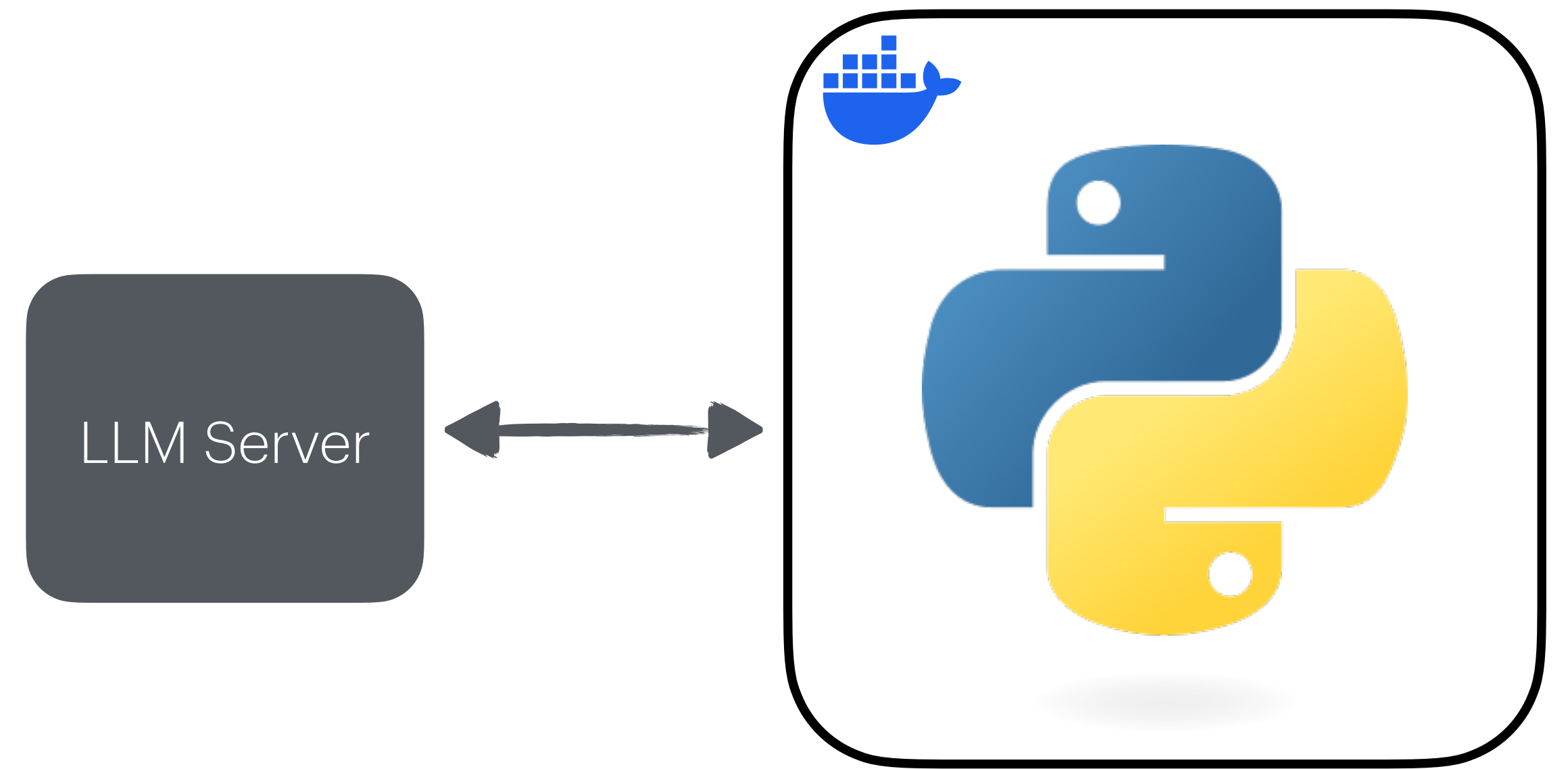
```
True<|eot_id|>
```

```
<|start_header_id|>assistant<|end_header_id|>
```

# LLMs that code

- Ask LLM to produce python code to compute answer
- Model might not know how to use answer
- **Not very secure**
- All of python might be too broad

Fine-tune LLM with coding chat template



Making arbitrary python code execution safe is almost impossible.

# LLMs that code

- Ask LLM to produce python code to compute answer
- Model might not know how to use answer
- Not very secure
- **All of python might be too broad**

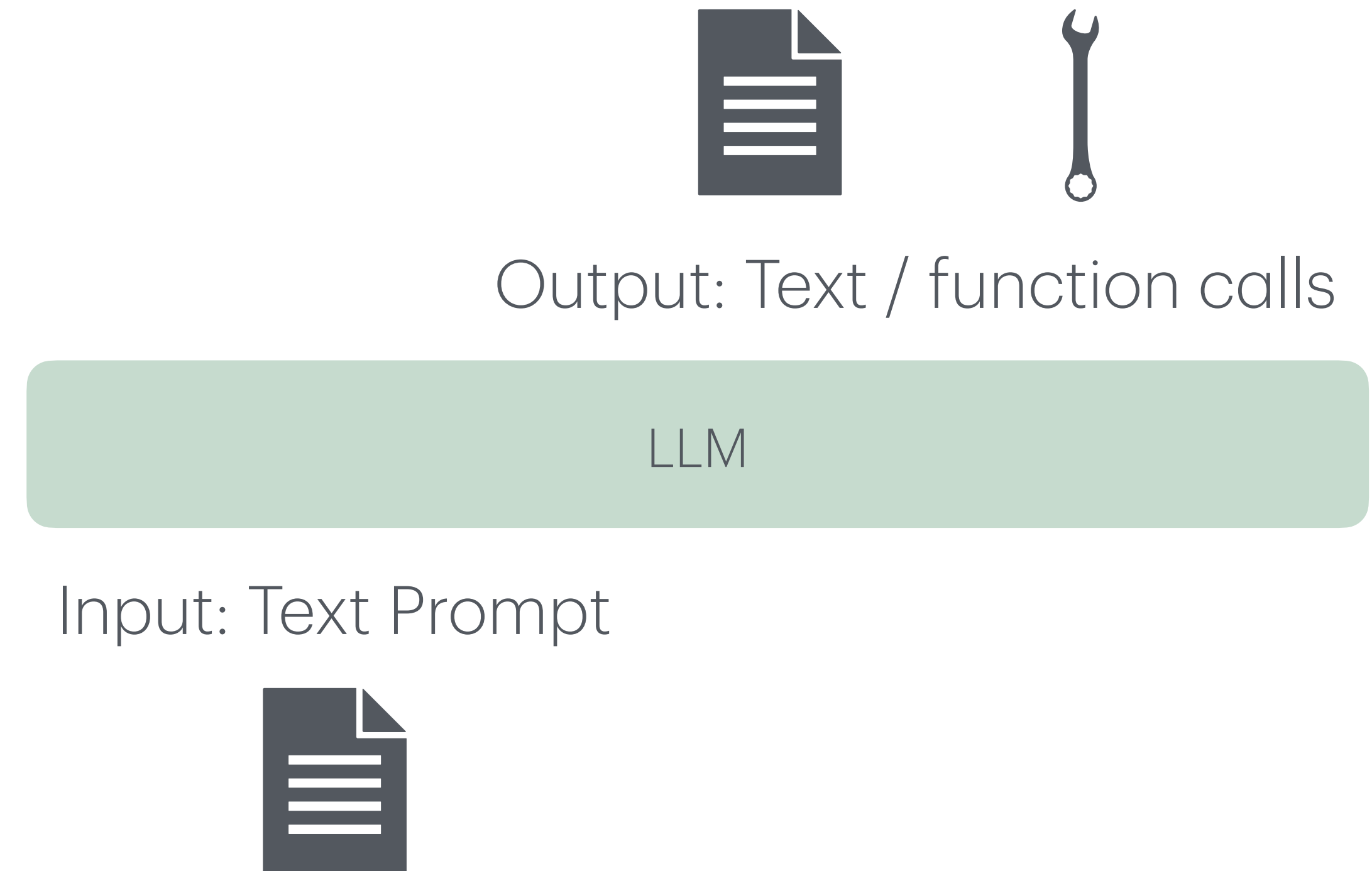
```
Built-in Functions
abs()
aiter()
all()
anext()
any()
ascii()
bin()
bool
breakpoint()
callable()
chr()
classmethod()
compile()
complex
delattr()
dir()
divmod()
enumerate()
eval()
exec()
filter()
float
format()
getattr()
globals()
hasattr()
hash()
help()
hex()
id()
input()
int
isinstance()
issubclass()
iter()
len()
locals()
map()
max()
min()
next()
object
oct()
open()
ord()
pow()
print()
property
repr()
reversed()
round()
setattr()
slice
sorted()
staticmethod()
sum()
super
type
vars()
zip()
__import__()
Built-in Constants
False
True
None
NotImplemented
Ellipsis
__debug__

string – Common string operations
re – Regular expression operations
difflib – Helpers for computing deltas
textwrap – Text wrapping and filling
unicodedata – Unicode Database
stringprep – Internet String Preparation
readline – GNU readline interface
rlcompleter – Completion function for GNU readline
Binary Data Services
struct – Interpret bytes as packed binary data
codecs – Codec registry and base classes
Data Types
datetime – Basic date and time types
zoneinfo – IANA time zone support
calendar – General calendar-related functions
collections – Container datatypes
collections.abc – Abstract Base Classes for Containers
heapq – Heap queue algorithm
bisect – Array bisection algorithm
array – Efficient arrays of numeric values
weakref – Weak references
types – Dynamic type creation and names for built-in types
copy – Shallow and deep copy operations
pprint – Data pretty printer
reprlib – Alternate repr() implementation
enum – Support for enumerations
graphlib – Functionality to operate with graph-like structures
Numeric and Mathematical Modules
numbers – Numeric abstract base classes
math – Mathematical functions
cmath – Mathematical functions for complex numbers
decimal – Decimal fixed-point and floating-point arithmetic
fractions – Rational numbers
random – Generate pseudo-random numbers
statistics – Mathematical statistics functions
Functional Programming Modules
itertools – Functions creating iterators for efficient looping
functools – Higher-order functions and operations on callable objects
operator – Standard operators as functions
File and Directory Access
pathlib – Object-oriented filesystem paths
os.path – Common pathname manipulations
fileinput – Iterate over lines from multiple input streams
stat – Interpreting stat() results
filecmp – File and Directory Comparisons
tempfile – Generate temporary files and directories
glob – Unix style pathname pattern expansion
fnmatch – Unix filename pattern matching
linecache – Random access to text lines
shutil – High-level file operations
Data Persistence
pickle – Python object serialization
copyreg – Register pickle support functions
shelve – Python object persistence
marshal – Internal Python object serialization
dbm – Interfaces to Unix “databases”
sqlite3 – DB-API 2.0 interface for SQLite databases
Data Compression and Archiving
zlib – Compression compatible with gzip
gzip – Support for gzip files
bz2 – Support for bzip2 compression
lzma – Compression using the LZMA algorithm
zipfile – Work with ZIP archives
tarfile – Read and write tar archive files
File Formats
csv – CSV File Reading and Writing
configparser – Configuration file parser
tomllib – Parse TOML files
netrc – netrc file processing
plistlib – Generate and parse Apple .plist files
Cryptographic Services
hashlib – Secure hashes and message digests
hmac – Keyed-Hashing for Message Authentication
secrets – Generate secure random numbers for managing secrets
Generic Operating System Services
os – Miscellaneous operating system interfaces
io – Core tools for working with streams
time – Time access and conversions
argparse – Parser for command-line options, arguments and subcommands
logging – Logging facility for Python
logging.config – Logging configuration
logging.handlers – Logging handlers
getpass – Portable password input

curses – Terminal handling for character-cell displays
curses.textpad – Text input widget for curses programs
curses.ascii – Utilities for ASCII characters
curses.panel – A panel stack extension for curses
platform – Access to underlying platform’s identifying data
errno – Standard errno system symbols
ctypes – A foreign function library for Python
Concurrent Execution
threading – Thread-based parallelism
multiprocessing – Process-based parallelism
multiprocessing.shared_memory – Shared memory for direct access across processes
The concurrent package
concurrent.futures – Launching parallel tasks
subprocess – Subprocess management
sched – Event scheduler
queue – A synchronized queue class
contextvars – Context Variables
_thread – Low-level threading API
Networking and Interprocess Communication
asyncio – Asynchronous I/O
socket – Low-level networking interface
ssl – TLS/SSL wrapper for socket objects
select – Waiting for I/O completion
selectors – High-level I/O multiplexing
signal – Set handlers for asynchronous events
mmap – Memory-mapped file support
Internet Data Handling
email – An email and MIME handling package
json – JSON encoder and decoder
mailbox – Manipulate mailboxes in various formats
mimetypes – Map filenames to MIME types
base64 – Base16, Base32, Base64, Base85 Data Encodings
binascii – Convert between binary and ASCII
quopri – Encode and decode MIME quoted-printable data
Structured Markup Processing Tools
html – HyperText Markup Language support
html.parser – Simple HTML and XHTML parser
html.entities – Definitions of HTML general entities
XML Processing Modules
xml.etree.ElementTree – The ElementTree XML API
xml.dom – The Document Object Model API
xml.dom.minidom – Minimal DOM implementation
xml.dom.pulldom – Support for building partial DOM trees
xml.sax – Support for SAX2 parsers
xml.sax.handler – Base classes for SAX handlers
xml.sax.saxutils – SAX Utilities
xml.sax.xmlreader – Interface for XML parsers
xml.parsers.expat – Fast XML parsing using Expat
Internet Protocols and Support
webbrowser – Convenient web-browser controller
wsgiref – WSGI Utilities and Reference Implementation
urllib – URL handling modules
urllib.request – Extensible library for opening URLs
urllib.response – Response classes used by urllib
urllib.parse – Parse URLs into components
urllib.error – Exception classes raised by urllib.request
urllib.robotparser – Parser for robots.txt
http – HTTP modules
http.client – HTTP protocol client
ftplib – FTP protocol client
poplib – POP3 protocol client
imaplib – IMAP4 protocol client
smtplib – SMTP protocol client
uuid – UUID objects according to RFC 4122
socketserver – A framework for network servers
http.server – HTTP servers
http.cookies – HTTP state management
http.cookiejar – Cookie handling for HTTP clients
xmlrpc – XMLRPC server and client modules
xmlrpc.client – XML-RPC client access
xmlrpc.server – Basic XML-RPC servers
ipaddress – IPv4/IPv6 manipulation library
Multimedia Services
wave – Read and write WAV files
colorsys – Conversions between color systems
Internationalization
gettext – Multilingual internationalization services
locale – Internationalization services
Program Frameworks
turtle – Turtle graphics
cmd – Support for line-oriented command interpreters
shlex – Simple lexical analysis
...
```

# LLMs with tool calls

- Give LLM access to limited set of tools
  - Input: Text prompt
  - Output: Text or function call



# LLMs with tool calls

- What tools should we use?
  - Toolformer: calculator, QA-model, translation, wiki search, date-function
  - General: User defined tools in context / system prompt

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

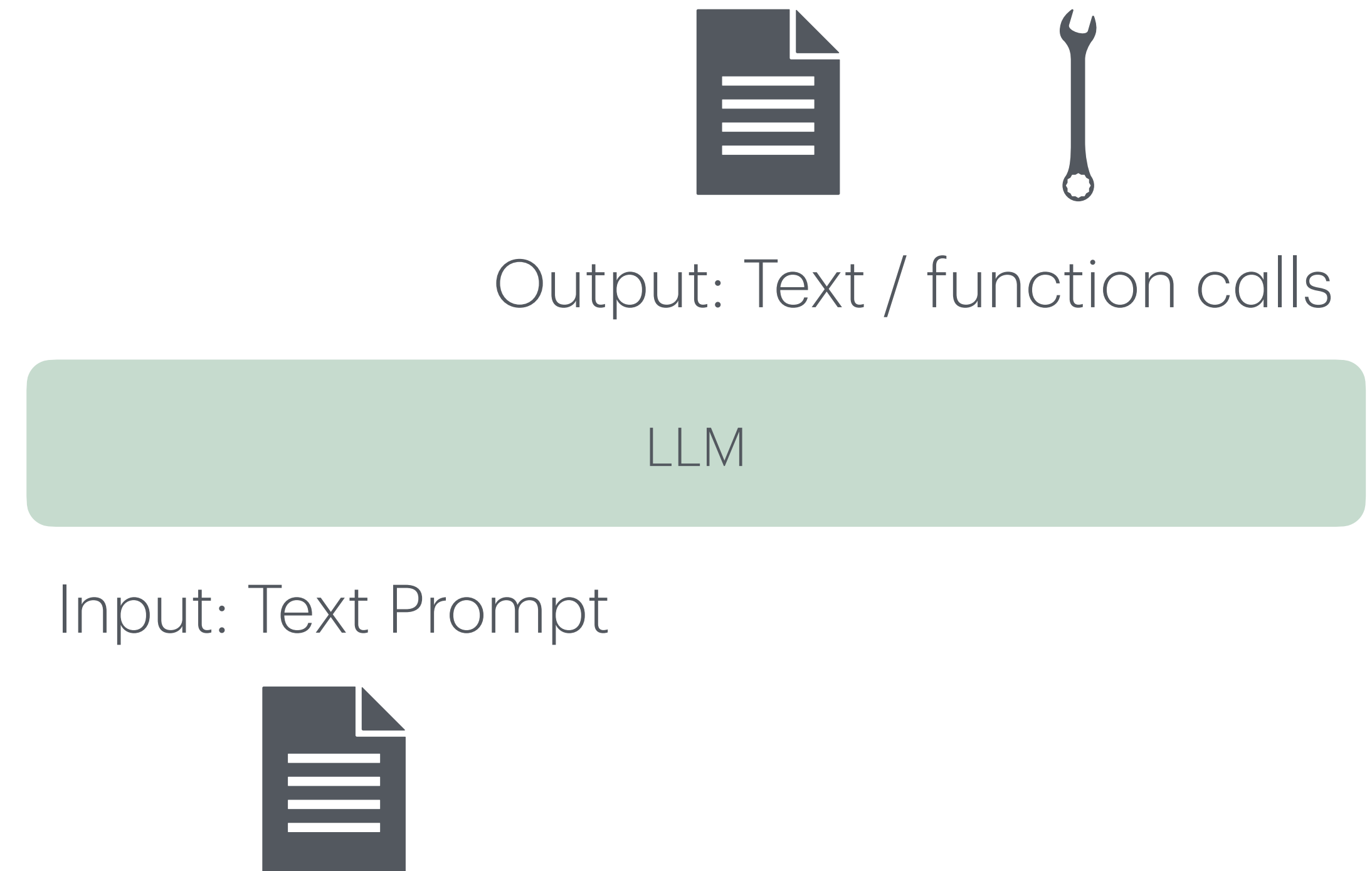
Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.

# LLMs with tool calls

- How does inference work?
  - Without tools: Outputs become new inputs
  - With tools?



# LLMs with tool calls

- How does inference work?
  - Without tools: Outputs become new inputs
- With tools:
  - Tool call, evaluate, append to input
- Input and output not the same

Output: Text / function calls



LLM

Input: Text Prompt / tool call + result





# LLMs with tool calls

- How does training work?
  - Without tools
    - Outputs = inputs shifted by 1
  - With tools
    - Outputs = inputs shifted by 1
    - No loss on tool result

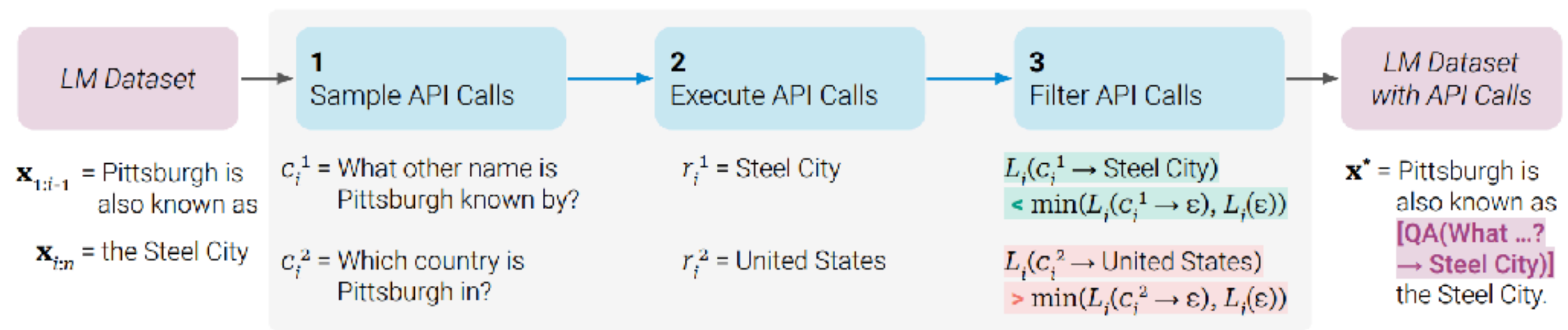


# LLMs with tool calls

- Where do we get training data from?
  - Option A: Lots of hand-designed data
  - Option B: We use a LLM to create training data (Toolformer)

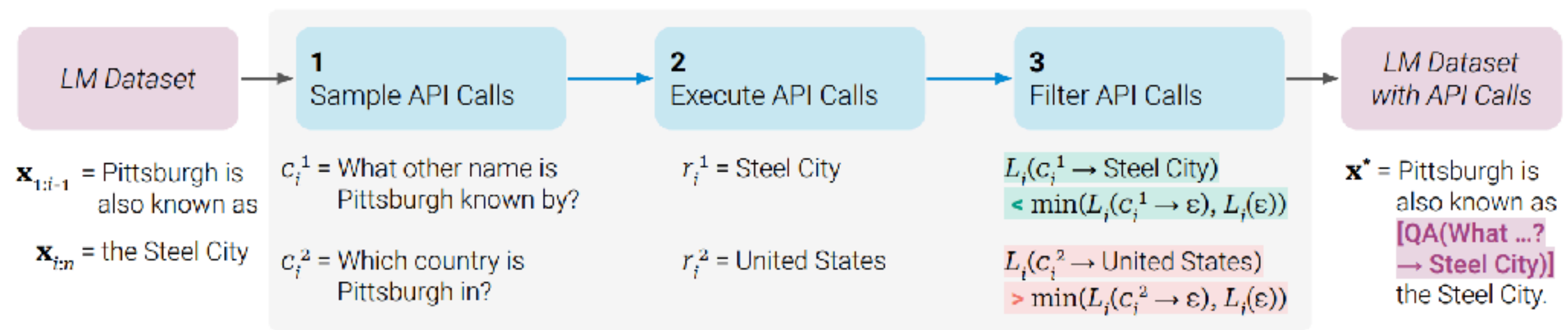


# ToolFormer



- Basic syntax
  - New tag: **<API>FN(Q)→R</API>**
  - Model learns to produce **<API>FN(Q)</API>**
  - Model learns to parse **<API>FN(Q)→R</API>**
- Task: Convert LM Dataset → LM Dataset with **<API>...</API>**

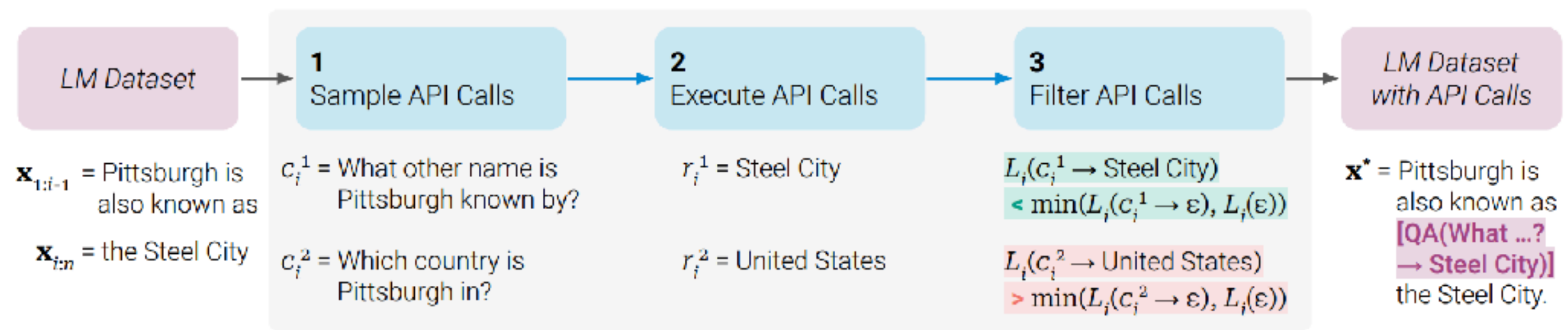
# ToolFormer



**Dataset item:** The president of the United States is Joe Biden.

- LM Dataset → LM Dataset with `<API>...</API>`
- Step 1: Load a dataset

# ToolFormer



- LM Dataset  $\rightarrow$  LM Dataset with  $\langle \text{API} \rangle \dots \langle / \text{API} \rangle$
- Step 1: Load a dataset
- Step 2: Use in-context learning to teach LLM to use API

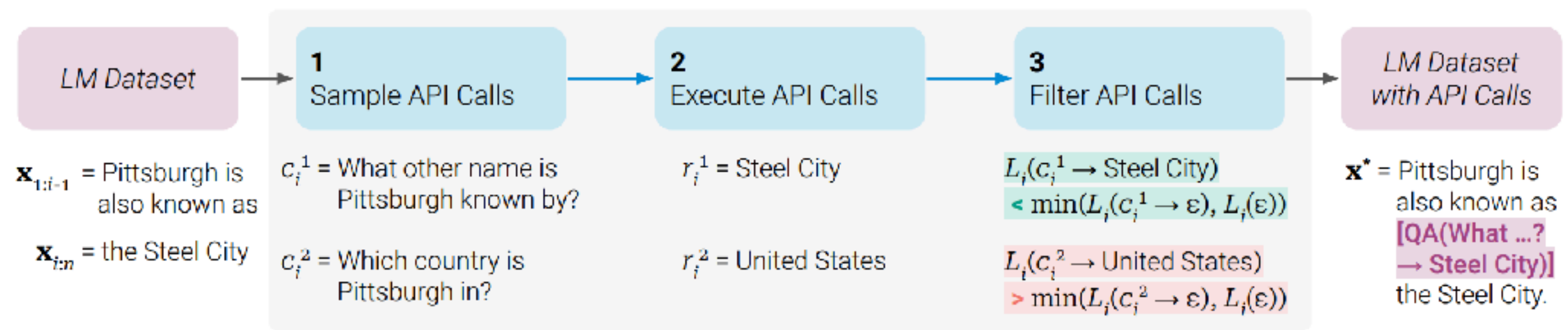
Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask.

Here are some examples of API calls:

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

# ToolFormer



- LM Dataset  $\rightarrow$  LM Dataset with **<API>...</API>**
- Step 1: Load a dataset
- Step 2: Use in-context learning to teach LLM to use API
- Step 3: Construct N prompts, stop at word  $i$  and open **<API>** (for all  $i$ ), let LLM complete

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask.

Here are some examples of API calls:

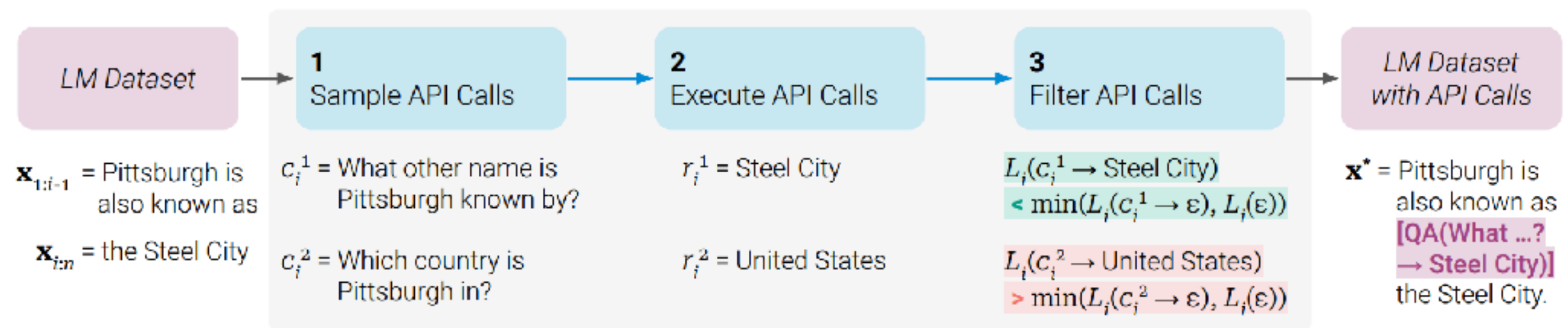
Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

Input: The president of the United States is Joe Biden.

Output: The president of the [

# ToolFormer



$L(\epsilon)$

Output: The president of the United States is Joe Biden

- LM Dataset  $\rightarrow$  LM Dataset with  $\langle \mathbf{API} \rangle \dots \langle / \mathbf{API} \rangle$

- Step 1: Load a dataset

- Step 2: Use in-context learning to teach LLM to use API

- Step 3: Construct N prompts, stop at word i and open  $\langle \mathbf{API} \rangle$  (for all i), let LLM complete

- Step 4: Keep useful examples

$$L(Q, a) + \tau \leq \min(L(\epsilon), L(Q))$$

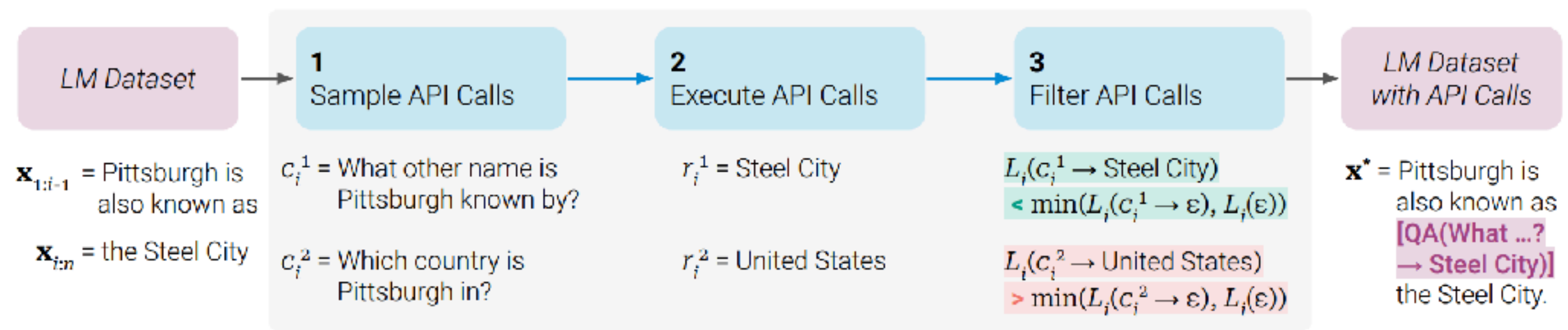
$L(Q, \epsilon)$

Output: The president of the United States is [Who is the president of the United States] Joe Biden

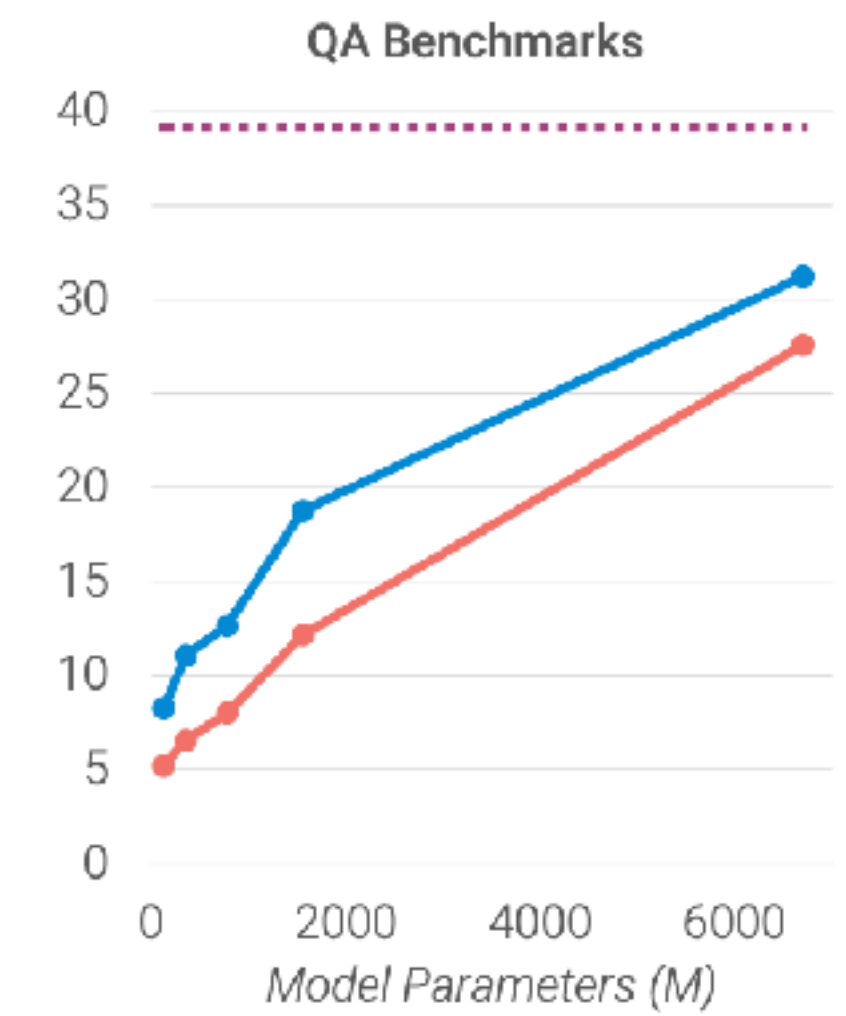
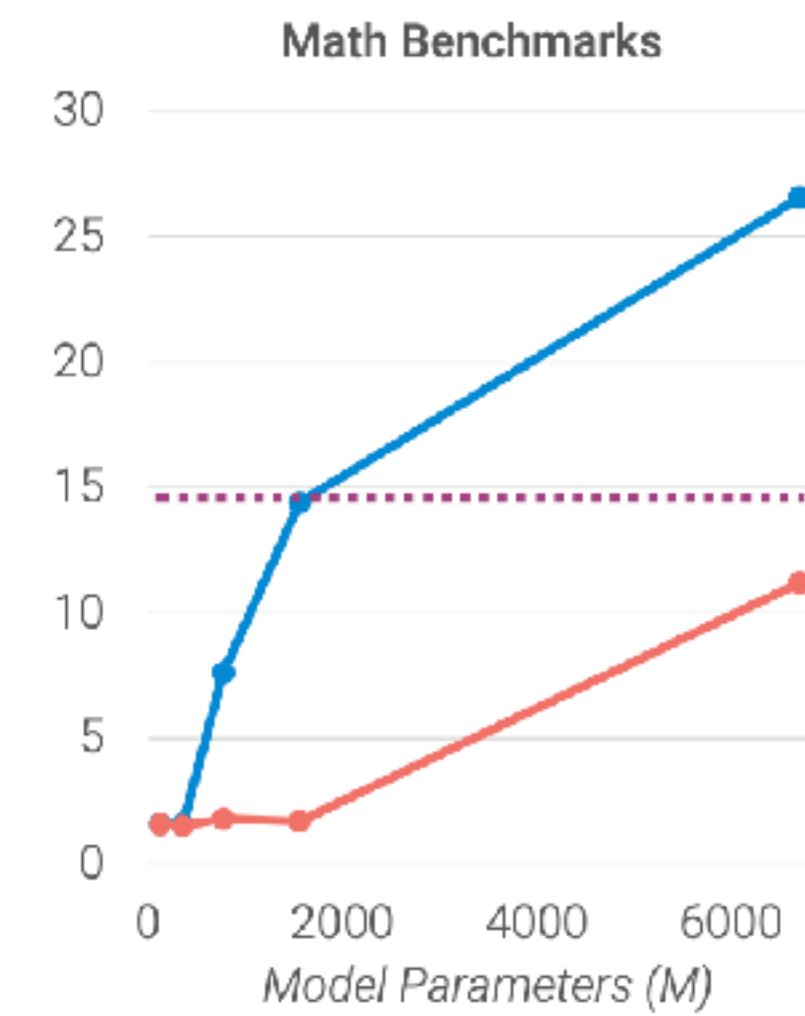
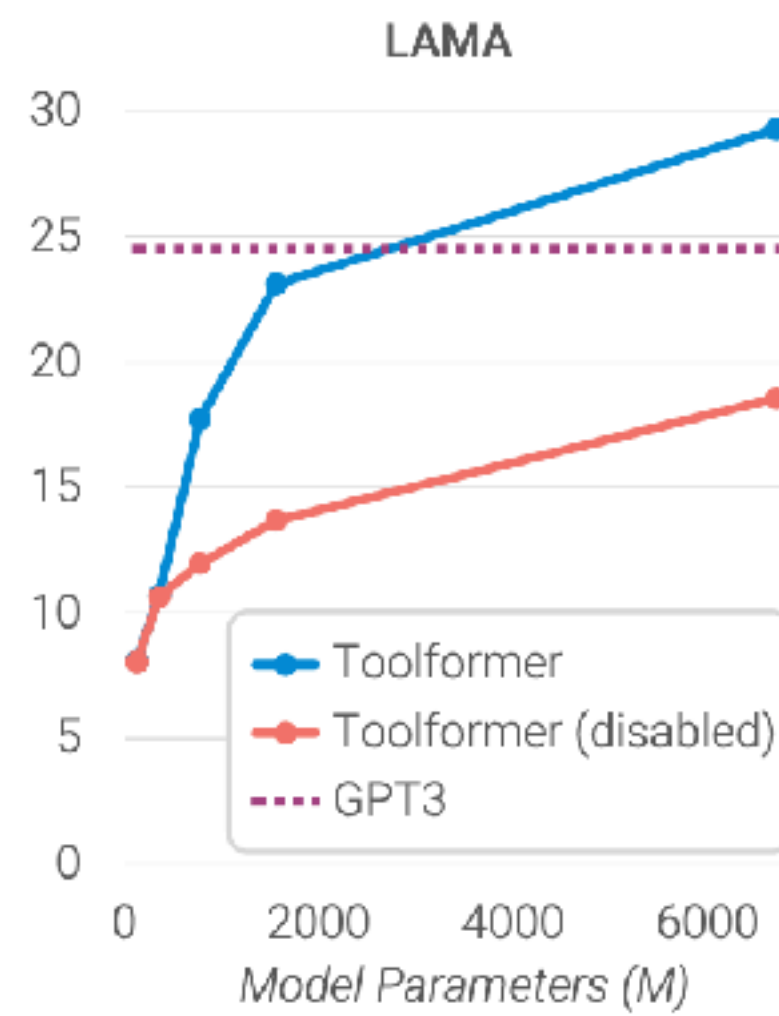
$L(Q, a)$

Output: The president of the United States is [Who is the president of the United States  $\rightarrow$  Joe Biden] Joe Biden

# ToolFormer



- Use LM Dataset with API calls to finetune model





# LLMs with tool calls

- Tool datasets
  - Option A: Lots of hand-designed data
  - Option B: We use a LLM to create training data (Toolformer)



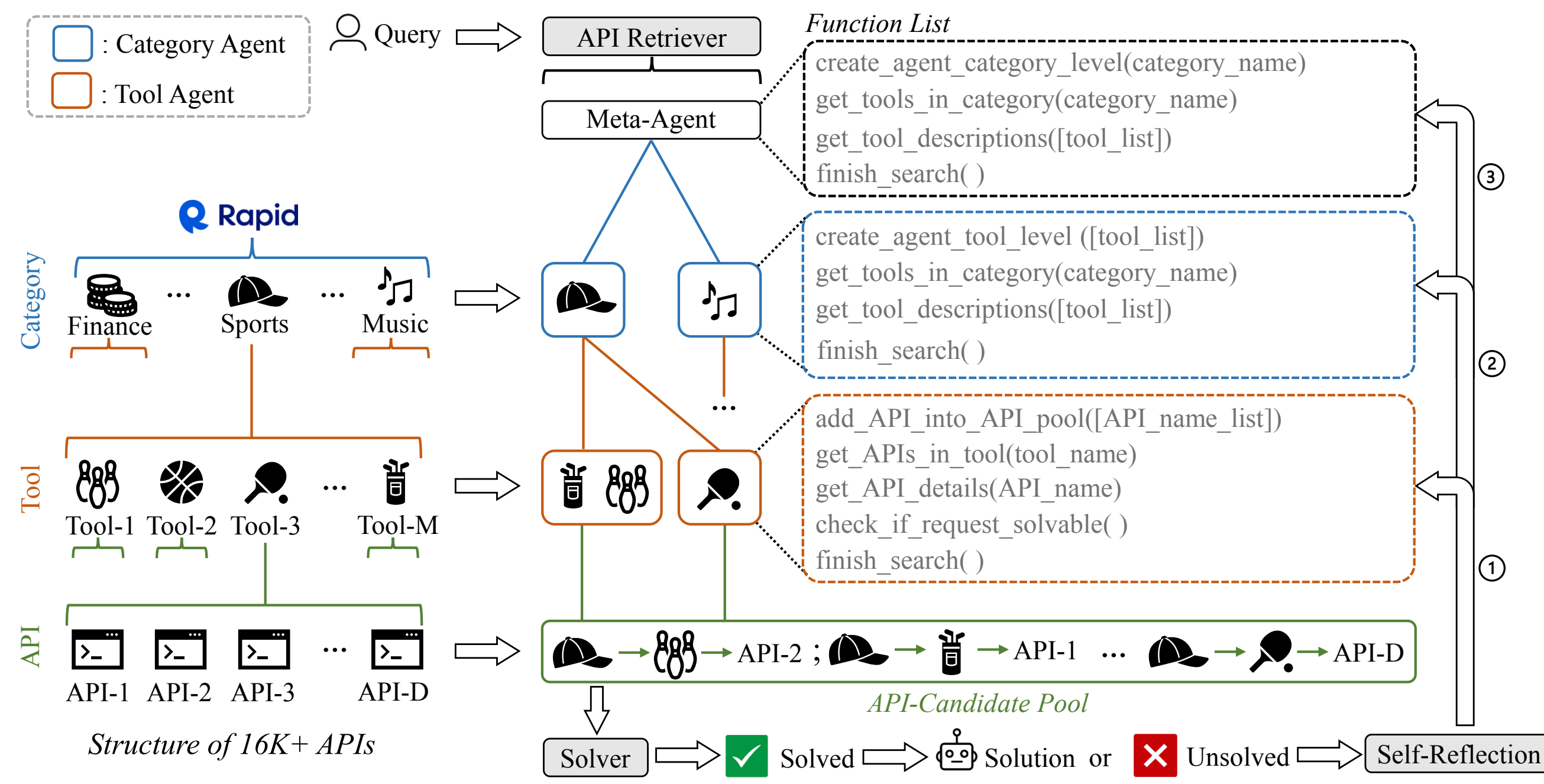
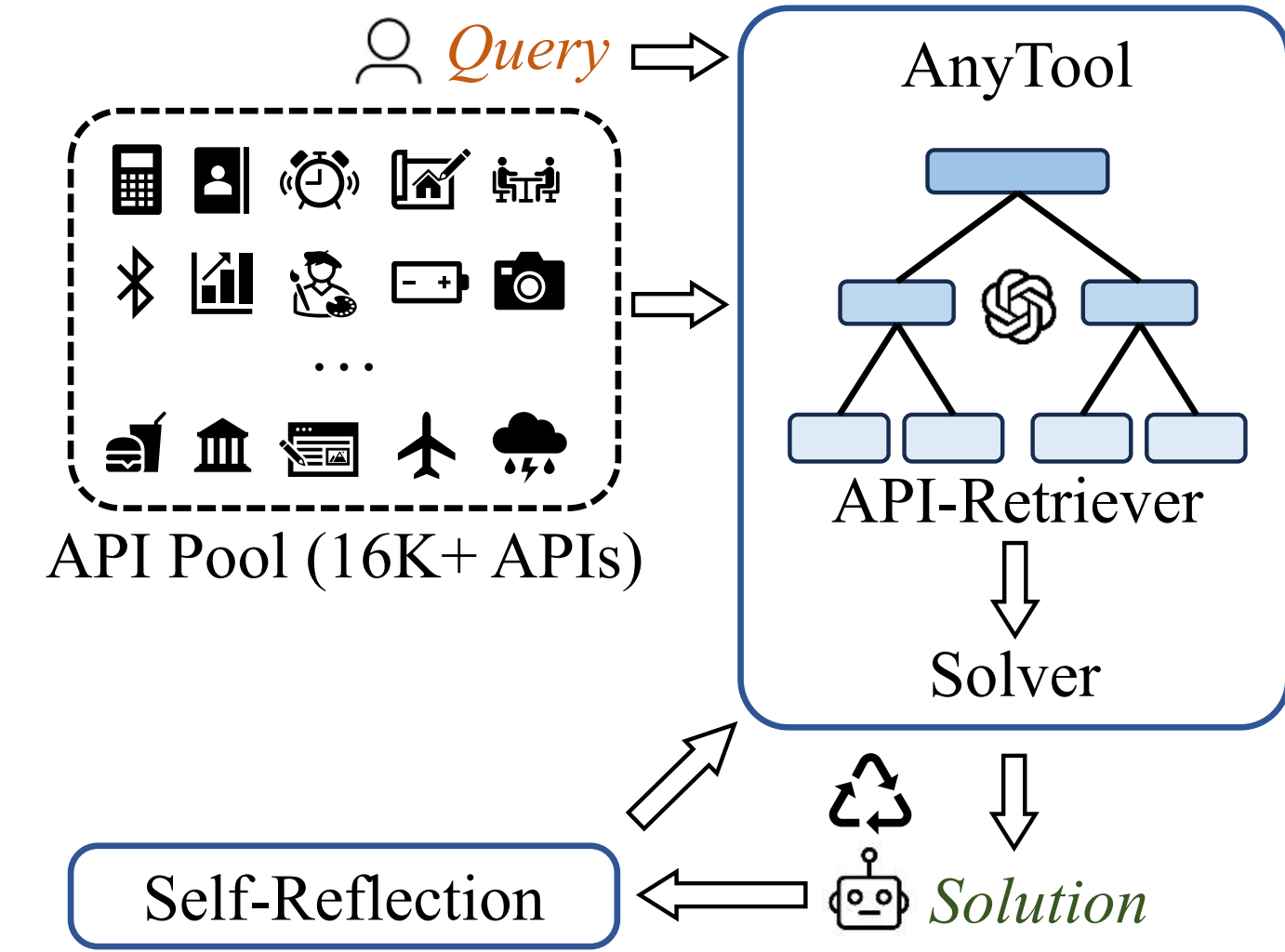
# Zero-shot tool use

- Train with many more tools
- Always show in-context examples
  - Structured / json description
- Fine-tune model



# AnyTool

- Task: Tool retrieval
- Expand planner to 16k+ APIs
- 4 levels of agents
  - Meta-Agent
  - Category-Agent
  - Tool-Agent
  - LLM solver



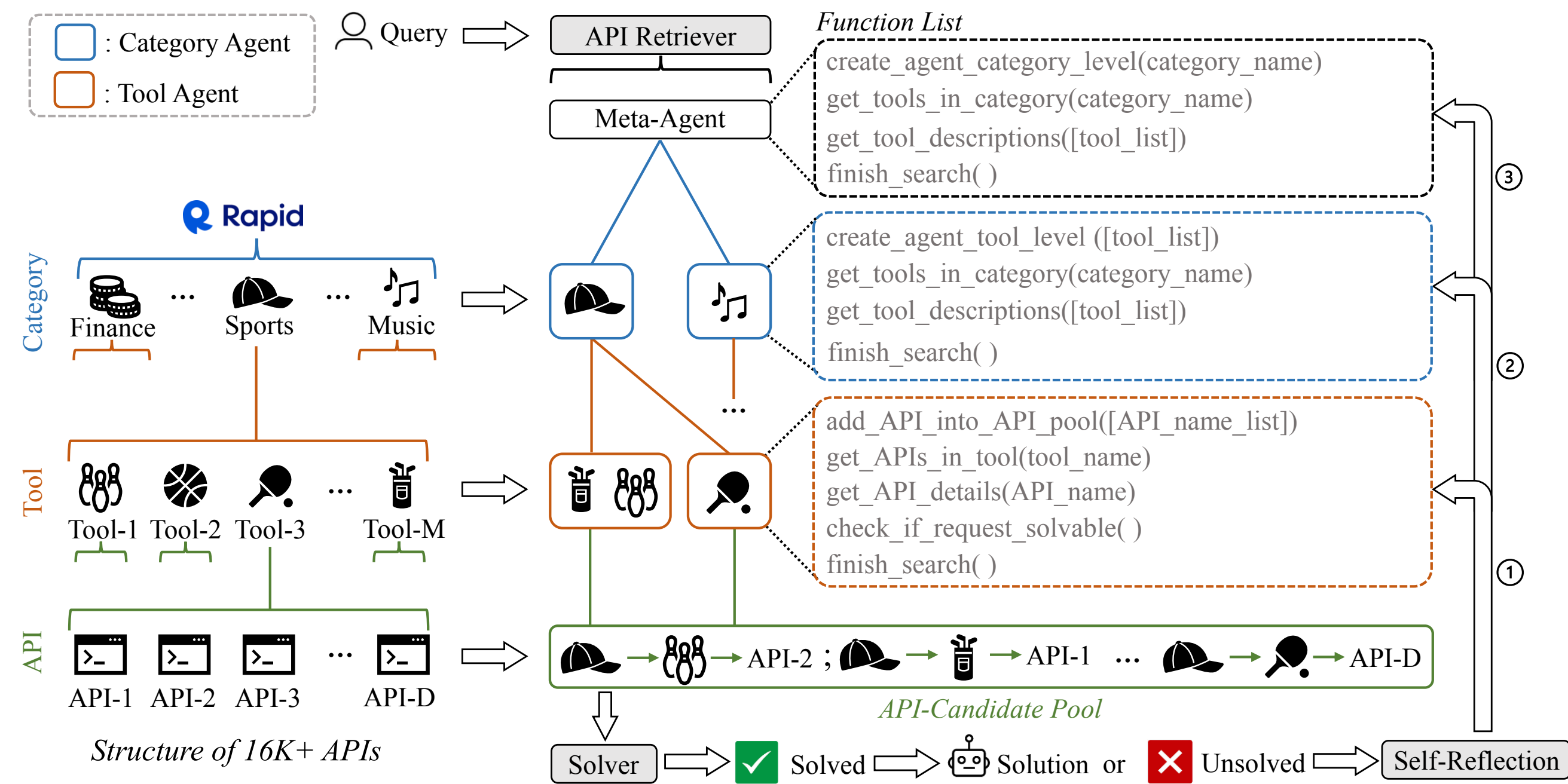
# AnyTool

- First 3 levels build candidate tool set
  - Meta-Agent
  - Category-Agent
  - Tool-Agent
- Called recursively until tool set is defined (each search terminates with **finish\_...**)

| Type           | Function Name               | Functionality                                                         | Input         | Output              |
|----------------|-----------------------------|-----------------------------------------------------------------------|---------------|---------------------|
| Meta Agent     | create_agent_category_level | Create a category agent.                                              | Category name | Category agent      |
|                | get_tools_in_category       | Get tool names under a category.                                      | Category name | [Tool names]        |
|                | get_tool_descriptions       | Get description of each tool.                                         | [Tools]       | [Tool descriptions] |
|                | finish_search               | Send out finish signal.                                               | None          | None                |
| Category Agent | create_agent_tool_level     | Create a tool agent.                                                  | [Tools]       | Tool agent          |
|                | get_tools_in_category       | Get tool names under a category.                                      | Category name | [Tool names]        |
|                | get_tool_descriptions       | Get description of each tool.                                         | [Tools]       | [Tool descriptions] |
|                | finish_search               | Send out finish signal.                                               | None          | None                |
| Tool Agent     | add_API_into_API_pool       | Add APIs into candidate pool.                                         | [APIs]        | None                |
|                | get_APIs_in_tool            | Get API names under a tool.                                           | Tool name     | [API names]         |
|                | get_API_detail              | Get detail* of each API.                                              | [API names]   | [API details]       |
|                | check_if_request_solvable   | Check whether the query is solvable using the current candidate pool. | None          | True\False          |
|                | finish_search               | Send out finish signal.                                               | None          | None                |

# AnyTool

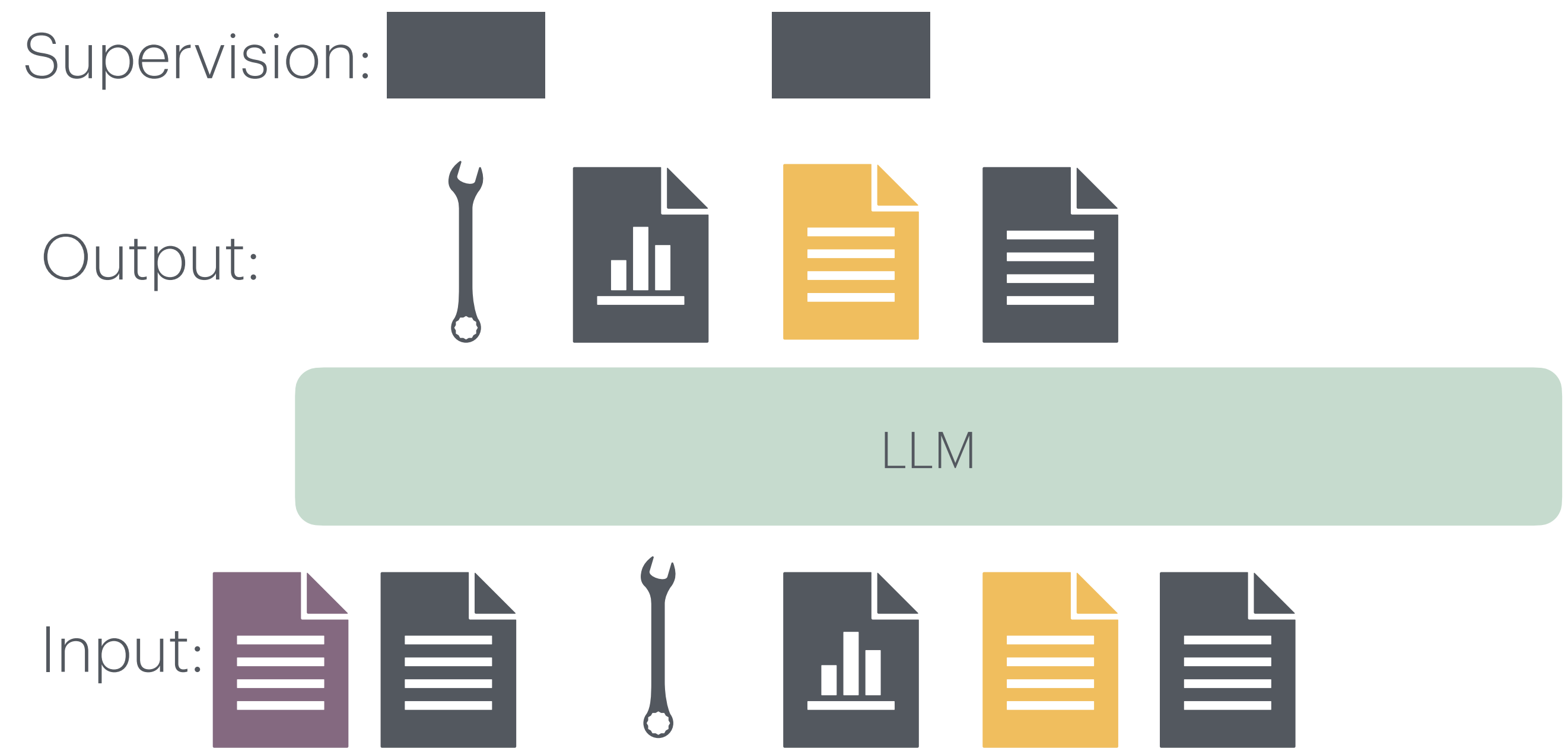
- Final agent (LLM solver)
- Solves task using toolset
- Self-Reflection (Reflextion) to correct output
- All models are prompted GPT-4 variants



|           | AnyTool    |
|-----------|------------|
| Turn      | Single     |
| Step      | Multi      |
| Learned   | No         |
| Objective | Goal State |
| Toolset   | Open       |
| Feedback  | Yes        |

# LLMs with tools

- Allow LLM to output tool calls
- Special tags
- Special chat-template



# References

- [1] Toolformer: Language Models Can Teach Themselves to Use Tools, Schick et al 2023
- [2] The Llama 3 Herd of Models, Dubey et al 2024
- [3] AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls, 2024