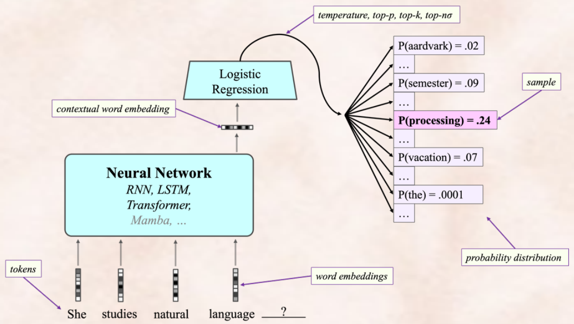
# ITCS 4101: Introduction to NLP

## Neuro-Symbolic AI: LLMs and Tool Use





Razvan C. Bunescu

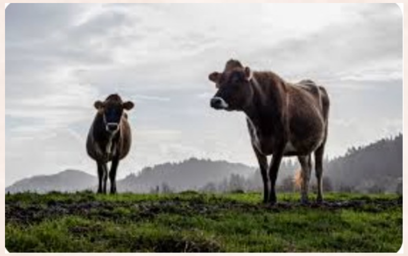
Department of Computer Science @ CCI

rbunescu@charlotte.edu

# ML and Generalization

- Types of generalization performance:
  - **Within distribution** (WiD) generalization:
    - Training and test examples come from the same underlying distribution.
  - **Out of distribution** (OoD) generalization:
    - Test examples come from a different distribution than training examples:
      - Need to satisfy invariant features that hold across the domain.

***WiD** example*



***OoD** example*

***Training** examples*

# ML and Generalization

- Types of generalization performance:
  - **Systematic Generalization**:
    - Human-like, higher-level generalization:
      - Generalize to unseen input lengths or sizes.
        - » Multiply 10-digit numbers, even though trained on only <5-digit numbers.
      - Generalize to unseen combinations of seen skills.
        - » *Turn right then pick object*, when trained to *turn right then walk one step* and *move to table then pick object*.
    - Strong version of systematic generalization:
      - Want to generalize with 100% accuracy on examples from the target domain.
    - Some illustrative studies:
      - Neural Networks and the Chomsky Hierarchy, ICLR 2023.
      - Limits of transformer language models on learning to compose algorithms, NeurIPS 2024.
      - Autoformalization of Mathematics and Code Correctness: Experiments with Elementary Proofs, MathNLP 2022.

# Neuro vs. Symbolic: Shortest Paths

Neural Turing Machines, 2014.

Differentiable Neural Computers, Nature 2016.

i.e., *LSTMs with external memory and read/write heads.*

*"**gave an average accuracy of 55.3%**" on 4-step SPs.*

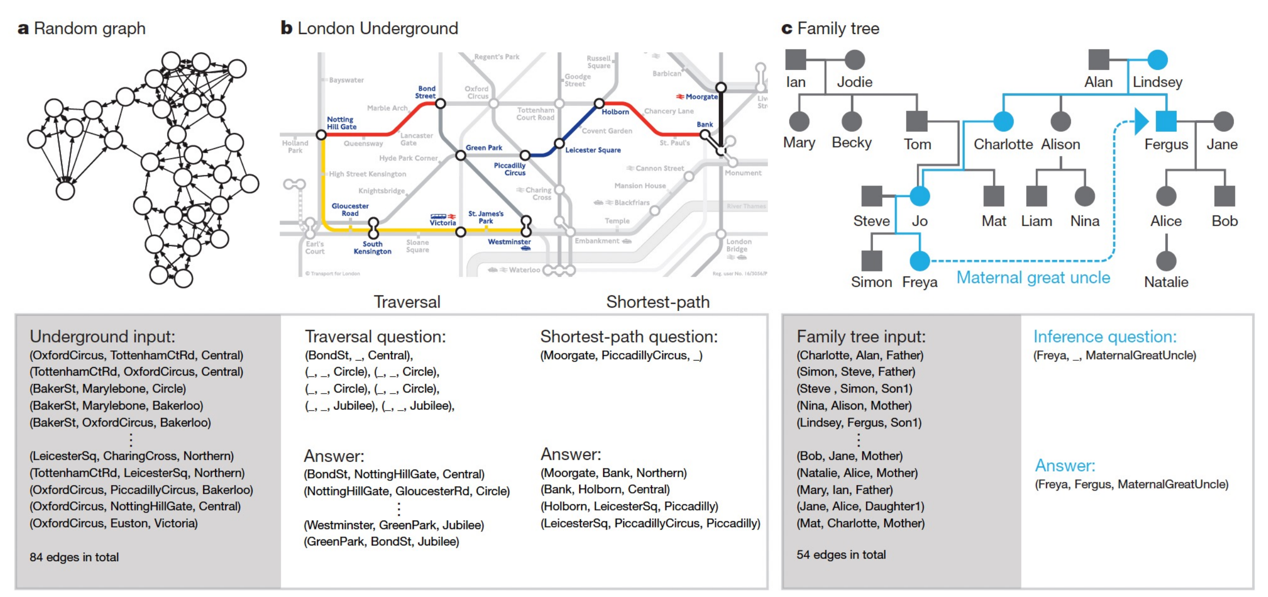



**Figure 2 | Graph tasks. a**, An example of a randomly generated graph used for training. **b**, Zone 1 interchange stations of the London Underground map, used as a generalization test for the traversal and shortest-path tasks. Random seven-step traversals (an example of which is shown on the left) were tested, yielding an average accuracy of 98.8%. Testing on all possible four-step shortest paths (example shown on the right) gave an average accuracy of 55.3%. **c**, The family tree that was used as a generalization test for the inference task; four-step relations such as the one shown in blue (from Freya to Fergus, her maternal great uncle) were tested, giving an average accuracy of 81.8%. The symbol sequences processed by the network during the test examples are shown beneath the graphs. The input is an unordered list of ('from node', 'to node', 'edge') triple vectors that describes the graph. For each task, the question is a sequence of triples with missing elements (denoted '_') and the answer is a sequence of completed triples.

Dijkstra's Shortest Path Algorithm, 1956.

***accuracy of 100%** on SPs of all lengths.*

**Dijkstra's algorithm**

```
DIJKSTRA (G(V, E), w, s)        /* s is the source */
  InitializeSingleSource(G, s);

  S := Ø;                       /* Make S empty */
  Q := V;                       /* put all vertices into
                                   a Priority Queue */
  while Q is not empty
       u := Extract-Min(Q);     /* get the vertex which is
                                   closest to the source s, and
                                   remove it from the queue */
       S := S ∪ u;              /* Add u to S */
       for each v ∈ Adj[u]      /* update the ds to s */
            Relax (u, v, w, Q);
```

```
InitializeSingleSource(G,s) {
    for each vertex v ∈ V do
      d[v] = ∞
      π[v] = nil
    d[s] = 0
}
```

```
Relax(u, v, w) {
    if d[v] > d[u] + w(u, v) then
       d[v] = d[u] + w(u, v)
       π[v] = u
}
```

# Neuro-Symbolic AI

- AI Overview Neuro-symbolic AI is a field of artificial intelligence that **integrates neural networks (the "neuro" part) with symbolic reasoning systems (the "symbolic" part)**.
  - It is a hybrid approach designed to combine the strengths of both traditional AI paradigms to create **more robust**, **versatile**, and **explainable** AI systems.

**1. The Neural Component (The "Neuro")**

- **What it does:** Excels at **pattern recognition, perception, and learning** from vast amounts of unstructured data.
- **How it works:** Uses **neural networks** (like deep learning models) to process raw, "messy" data such as images, text, and sensor readings. This component is excellent for tasks like image classification, natural language processing, and making statistical predictions.
- **Weaknesses alone:** Often acts as a "black box" (difficult to explain its decisions), struggles with logical reasoning, and is susceptible to "hallucinations" or errors when encountering data outside its training distribution.

**2. The Symbolic Component (The "Symbolic")**

- **What it does:** Excels at **logical reasoning, planning, and representing explicit knowledge** in a structured way.
- **How it works:** Uses **knowledge-based systems** (like rules, logic programming, or knowledge graphs) to apply rules and constraints to symbols, which represent entities and relationships. This component is excellent for tasks requiring clear, step-by-step logic, verifiability, and adherence to known facts.
- **Weaknesses alone:** Can be rigid, struggles to learn from raw, unstructured data, and requires human experts to pre-define all the knowledge and rules.

**The Integration**

Neuro-symbolic AI combines these two so that they complement each other:

- The **neural** network can handle the **perceptual task** of extracting concepts or symbols from data (e.g., recognizing a "dog" and a "ball" in an image).
- The **symbolic** system can then apply **logical reasoning** to these concepts (e.g., inferring that "if the dog is next to the ball, the dog might be playing").

**Key Benefits**

1. **Explainability/Transparency:** By incorporating explicit logic, the system's decisions can be tracked and explained, moving away from the "black box" problem.
2. **Robustness and Accuracy:** Logical constraints can prevent neural networks from making statistically plausible but logically impossible or incorrect decisions (e.g., a medical AI can't prescribe an antibiotic for a viral infection if a symbolic rule forbids it).
3. **Generalization:** It can learn and reason with fewer examples by leveraging structured knowledge, making it less data-hungry than purely neural approaches.
4. **Complex Reasoning:** It bridges the gap between low-level data processing and high-level abstract reasoning, leading to more human-like cognitive abilities.

# LLMs with Tools are Neuro-Symbolic

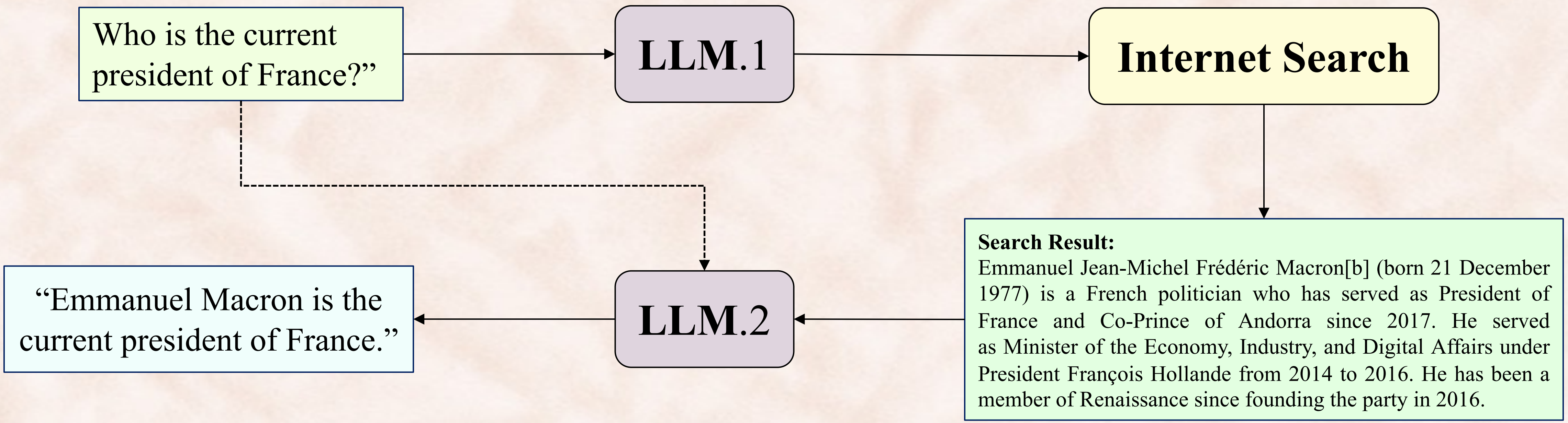
- Sometimes a language model is not the best for a task.
  - Most state-of-the-art language models still struggle with:
    - Recent knowledge, beyond training time cutoff.
    - Arithmetic.
    - Code execution.

- We want to allow LLM to interact with other agents or tools suited for the task:
  - Calculators.
  - Compilers/Interpreters.
  - Retrieval systems (Vector stores, Graphs, …).

- Provide the LLM with *one or more* tools, where each tool comes with its own:
  - **Interface**, i.e., function name and parameters.
  - **Description**, including a description of the problem it solves, its parameters and return values.

# Tool Scenario: Augment Knowledge

- Use tools to bring in knowledge that was not available at training time.
  - **Web Search** tool for accessing data about recent events.



# Before 2025: The ReAct Loop

- ReAct prompting is an approach for enhancing the decision making of LLMs, for example, choosing which tools to use for a given situation.
  - ReAct: Synergizing Reasoning and Acting in Language Models, ICLR 2023
  - **ReAct** instructs or fine-tunes LLMs to use a structured loop containing:
    - **Question**: The input to the model.
    - **Thought**: A place for the model to state intentions, which tends to increase performance.
    - **Action**: The tool (if any) does the model wants to use.
      - Action Input: The input to the tool.
    - **Observation**: The output of the tool.
  - The final answer is given in the format:
    - **Thought**: "I believe I have the final answer."
    - **Final Answer**: The answer the model gives to the user.

# Before 2025: The ReAct Loop

- The structure of the ReAct Loop:
  - <question> (<thought> <action> <observation>)* <thought> <answer>

**Question**: Who is the current president of France?

**Thought**: I should use a search engine to find relevant data.

**Action**: Open Google

Action Input: “Who is the current president of France?”

**Observation**: The current French president Emmanuel Macron.

Option 1: *You do not have enough information to answer the question.*

**Thought**: I now know the final answer.

**Answer**: Emmanuel Macron.

Option 2: *You have enough information to answer the question.*

# Before 2025: The ReAct Loop

- The structure of the ReAct Loop:
  - <question> (<thought> <action> <observation>)* <thought> <answer>

**Question**: What is 2 + 2?

**Thought**: I now know the final answer.

**Final Answer**: 4

Option 2: *You have enough information to answer the question.*

# Before 2025: The ReAct Loop

- Two ways to implement a ReAct behavior with LLMs:
  - **Instructions + In-Context Learning (ICL)** examples showing ReACT trajectories created manually for a few example queries.
    - See notebook for HotPotQA: https://github.com/ysymyth/ReAct/blob/master/hotpotqa.ipynb
    - *Solve a question answering task with interleaving **Thought**, **Action**, **Observation** steps. Thought can reason about the current situation, and Action can be three types:*
      - *(1) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search.*
      - *(2) Lookup[keyword], which returns the next sentence containing keyword in the current passage.*
      - *(3) Finish[answer], which returns the answer and finishes the task.*

      *Here are some examples:*
      *<examples> ...*
  - **Fine-tuning** on 3,000 example ReAct example trajectories created semi-automatically.

# ReAct Trajectories

- Sample trajectories for the HotPotQA task, taken from https://arxiv.org/abs/2210.03629

| | |
|---|---|
| Question | Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who? |
| Thought 1 | The question simplifies to "The Simpsons" character Milhouse is named after who. I only need to search Milhouse and find who it is named after. |
| Action 1 | Search[Milhouse] |
| Observation 1 | Milhouse Mussolini Van Houten is a recurring character in the Fox animated television series The Simpsons voiced by Pamela Hayden and created by Matt Groening. |
| Thought 2 | The paragraph does not tell who Milhouse is named after, maybe I can look up "named after". |
| Action 2 | Lookup[named after] |
| Observation 2 | (Result 1 / 1) Milhouse was named after U.S. president Richard Nixon, whose middle name was Milhous. |
| Thought 3 | Milhouse was named after U.S. president Richard Nixon, so the answer is Richard Nixon. |
| Action 3 | Finish[Richard Nixon] |

| | |
|---|---|
| Question | Which documentary is about Finnish rock groups, Adam Clayton Powell or The Saimaa Gesture? |
| Thought 1 | I need to search Adam Clayton Powell and The Saimaa Gesture, and find which documentary is about Finnish rock groups. |
| Action 1 | Search[Adam Clayton Powell] |
| Observation 1 | Could not find [Adam Clayton Powell]. Similar: ['Adam Clayton Powell III', 'Seventh Avenue (Manhattan)', 'Adam Clayton Powell Jr. State Office Building', 'Isabel Washington Powell', 'Adam Powell', 'Adam Clayton Powell (film)', 'Giancarlo Esposito']. |
| Thought 2 | To find the documentary, I can search Adam Clayton Powell (film). |
| Action 2 | Search[Adam Clayton Powell (film)] |
| Observation 2 | Adam Clayton Powell is a 1989 American documentary film directed by Richard Kilberg. The film is about the rise and fall of influential African-American politician Adam Clayton Powell Jr.[3][4] It was later aired as part of the PBS series The American Experience. |
| Thought 3 | Adam Clayton Powell (film) is a documentary about an African-American politician, not Finnish rock groups. So the documentary about Finnish rock groups must instead be The Saimaa Gesture. |
| Action 3 | Finish[The Saimaa Gesture] |

| | |
|---|---|
| Question | What profession does Nicholas Ray and Elia Kazan have in common? |
| Thought 1 | I need to search Nicholas Ray and Elia Kazan, find their professions, then find the profession they have in common. |
| Action 1 | Search[Nicholas Ray] |
| Observation 1 | Nicholas Ray (born Raymond Nicholas Kienzle Jr., August 7, 1911 – June 16, 1979) was an American film director, screenwriter, and actor best known for the 1955 film Rebel Without a Cause. |
| Thought 2 | Professions of Nicholas Ray are director, screenwriter, and actor. I need to search Elia Kazan next and find his professions. |
| Action 2 | Search[Elia Kazan] |
| Observation 2 | Elia Kazan was an American film and theatre director, producer, screenwriter and actor. |
| Thought 3 | Professions of Elia Kazan are director, producer, screenwriter, and actor. So profession Nicholas Ray and Elia Kazan have in common is director, screenwriter, and actor. |
| Action 3 | Finish[director, screenwriter, actor] |

| | |
|---|---|
| Question | Which magazine was started first Arthur's Magazine or First for Women? |
| Thought 1 | I need to search Arthur's Magazine and First for Women, and find which was started first. |
| Action 1 | Search[Arthur's Magazine] |
| Observation 1 | Arthur's Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century. |
| Thought 2 | Arthur's Magazine was started in 1844. I need to search First for Women next. |
| Action 2 | Search[First for Women] |
| Observation 2 | First for Women is a woman's magazine published by Bauer Media Group in the USA.[1] The magazine was started in 1989. |
| Thought 3 | First for Women was started in 1989. 1844 (Arthur's Magazine) < 1989 (First for Women), so Arthur's Magazine was started first. |
| Action 3 | Finish[Arthur's Magazine] |

# After 2025: Tool Use is Integrated in LLM APIs

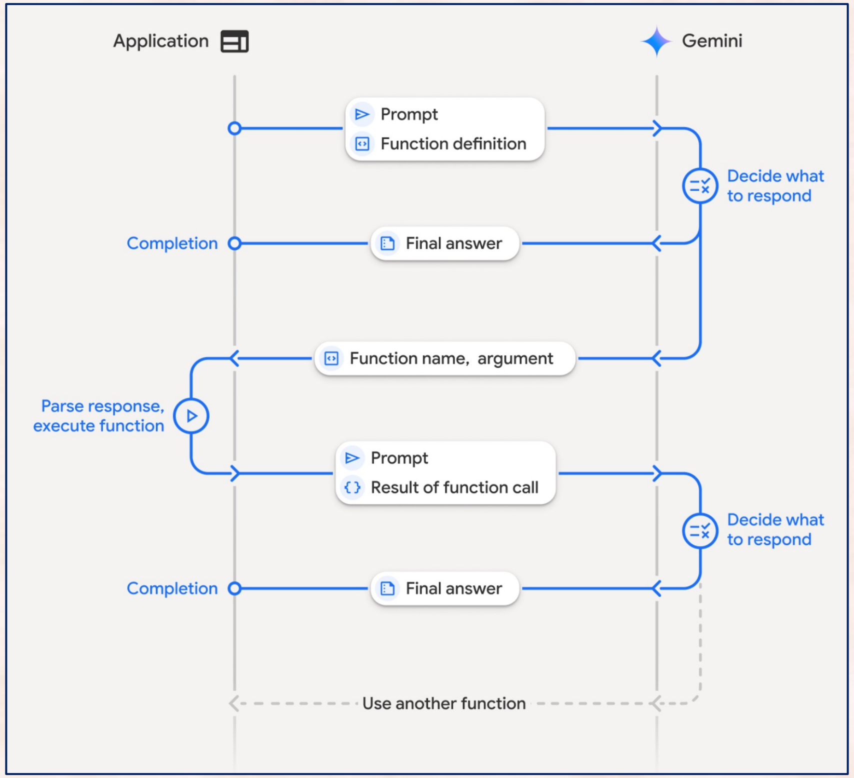
- Provide the LLM with *one or more* tools, where each tool comes with its own:
  - Interface, i.e., function name and parameters.
  - Description, including a description of the problem it solves, its parameters and return values.

**The ReAct Loop**

1. Given a prompt and tools, the LLM determines if:

   (a) it should respond directly.

   (b) it needs to use a tool.
2. If (a), LLM generates and **Returns** the answers.
3. If (b), LLM provides which tool to call and its arguments.
4. User code calls the tool.
5. User code provides the tool output back to the LLM.
6. **Repeat** from Step 1.



# Tool Use Example: Get Stock Price

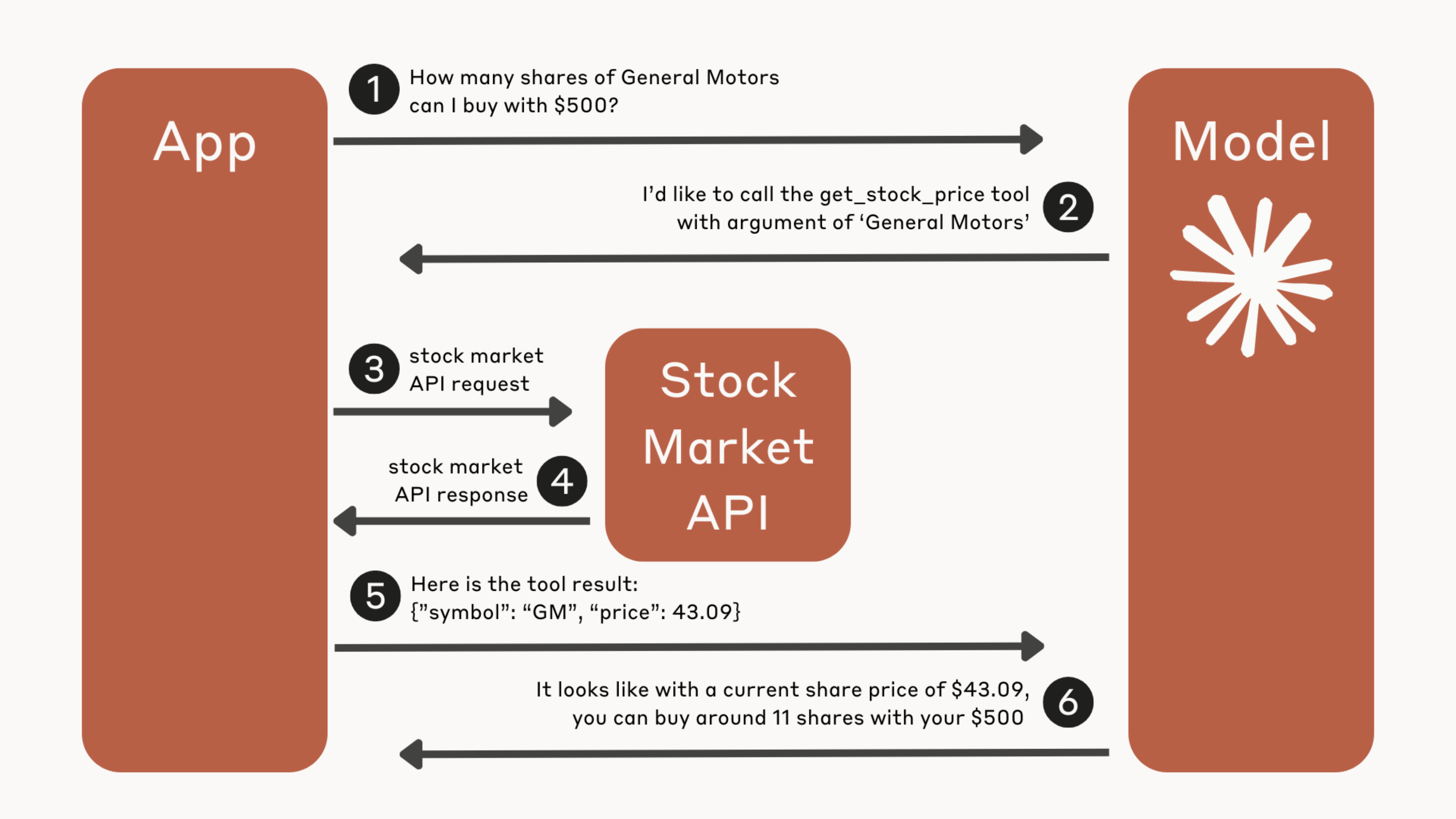



*Image credit to Ali Khaledi*

# Tool Use with the Gemini API: Function Calling

- **Function calling** lets you connect models to external tools and APIs.
- Instead of generating text responses, the model:
  - Determines when to call specific functions;
  - Provides the necessary parameters to execute real-world actions.
- This allows the model to act as a **bridge** between natural language and real-world actions and data. Function calling has 3 primary use cases:
  - **Augment Knowledge:** Access information from **external sources** like databases, APIs, and knowledge bases.
  - **Extend Capabilities:** Use external tools to perform computations and **extend the limitations of the model**, such as using a calculator or creating charts.
  - **Take Actions:** **Interact with external systems** using APIs, such as scheduling appointments, creating invoices, sending emails, or controlling smart home devices.

https://ai.google.dev/gemini-api/docs/function-calling

# Tool Use with the Gemini API: **Function Calling**

- **Define Function Declaration:**
  - The function's name, parameters, and purpose.
- **Call LLM with function declarations:**
  - Send user prompt along with the function declaration(s) to the model.
  - It analyzes the request and determines if a function call would be helpful.
    - If so, it responds with a structured JSON object.
- **Execute Function Code:**
  - It's your application's responsibility to process the response and check for Function Call:
    - If **Yes**: Extract the name and args of the function and execute the function in your application.
    - If **No:** The model has provided a direct text response to the prompt.
- **Create User friendly response:**
  - If a function was executed, user sends result back to the model in a subsequent turn of the conversation.
    - LLM will use the result to generate a response that incorporates the information from the function call.

# Tool Use with the Gemini API: **Function Calling**

- The tool use process can be repeated over multiple turns in a ReAct loop, allowing for complex interactions and workflows:
    1. Call LLM with function declarations (tools).
    2. Check LLM output, do one of the following:
        a) Execute function code, or
        b) Return answer.
    3. If function executed at (a), take return value, add instructions, repeat from 1.

- The model also supports:
    - Parallel function calling: Calling multiple functions in a single turn.
    - Compositional function calling: Calling multiple functions in sequence.

# Notebook with Tool Use Examples

1. Simple **tool use**.
2. Coding an *explicit* **ReAct loop**.
3. ReAct loop: one iteration:
   - Get stock price, compute number of shares
4. ReAct loop: two iterations:
   - Compare stock prices, compute number of shares
5. An *implicit* **ReAct loop** with **Automatic Function Calling**:
   - Compare stock prices, compute number of shares.
6. **Native tools** – **web search**:
   - One web search call to find stock price, then compute number of shares.
   - Multiple web search calls to find stock prices, then compute number of shares.

# Notebook with Tool Use Examples

7. **Native tools** – **web search** and **code generation** and **execution**:
   – Web search calls to find historical stock prices for two companies.
   – Code generation to fit linear predictor on historical data.
   – Code use to predict next stock value for the two companies.
   – Identify company most likely to appreciate in stock.
8. **Sequencing** of function calls.
9. Tool use API is a **leaky abstraction**!
   – Gemini 2.5 Flash is inconsistent in its behavior when tools are irrelevant to query
     - It confuses its “persona”.
       – *I am sorry, but I cannot answer this question. My capabilities are limited to … <tools>*
       – Or it gives the right response, depending on the sample.
   – **Gemini 2.5 Pro is much better**:
     - It seems to always be able to answer regular queries (when irrelevant tools are given).

# Examples with Gemini

- Shown in the Jupyter notebook.

# Supplemental Material: Gemini

- Gemini API reference:
  https://ai.google.dev/api
  https://ai.google.dev/gemini-api/docs
- Function Calling reference:
  https://ai.google.dev/gemini-api/docs/function-calling
- Grounding with Google Search:
  https://ai.google.dev/gemini-api/docs/google-search
- Code Execution:
  https://ai.google.dev/gemini-api/docs/code-execution
- URL context:
  https://ai.google.dev/gemini-api/docs/url-context

# Supplemental Material: GPT

- OpenAI developer platform:
  https://platform.openai.com/docs/overview
- Function Calling with OpenAI GPT:
  https://platform.openai.com/docs/guides/function-calling
- Web Search:
  https://platform.openai.com/docs/guides/tools-web-search?api-mode=responses
- Code interpreter:
  https://platform.openai.com/docs/guides/tools-code-interpreter
- File search and retrieval:
  https://platform.openai.com/docs/guides/tools-file-search
  https://platform.openai.com/docs/guides/retrieval
- Computer use:
  https://platform.openai.com/docs/guides/tools-computer-use