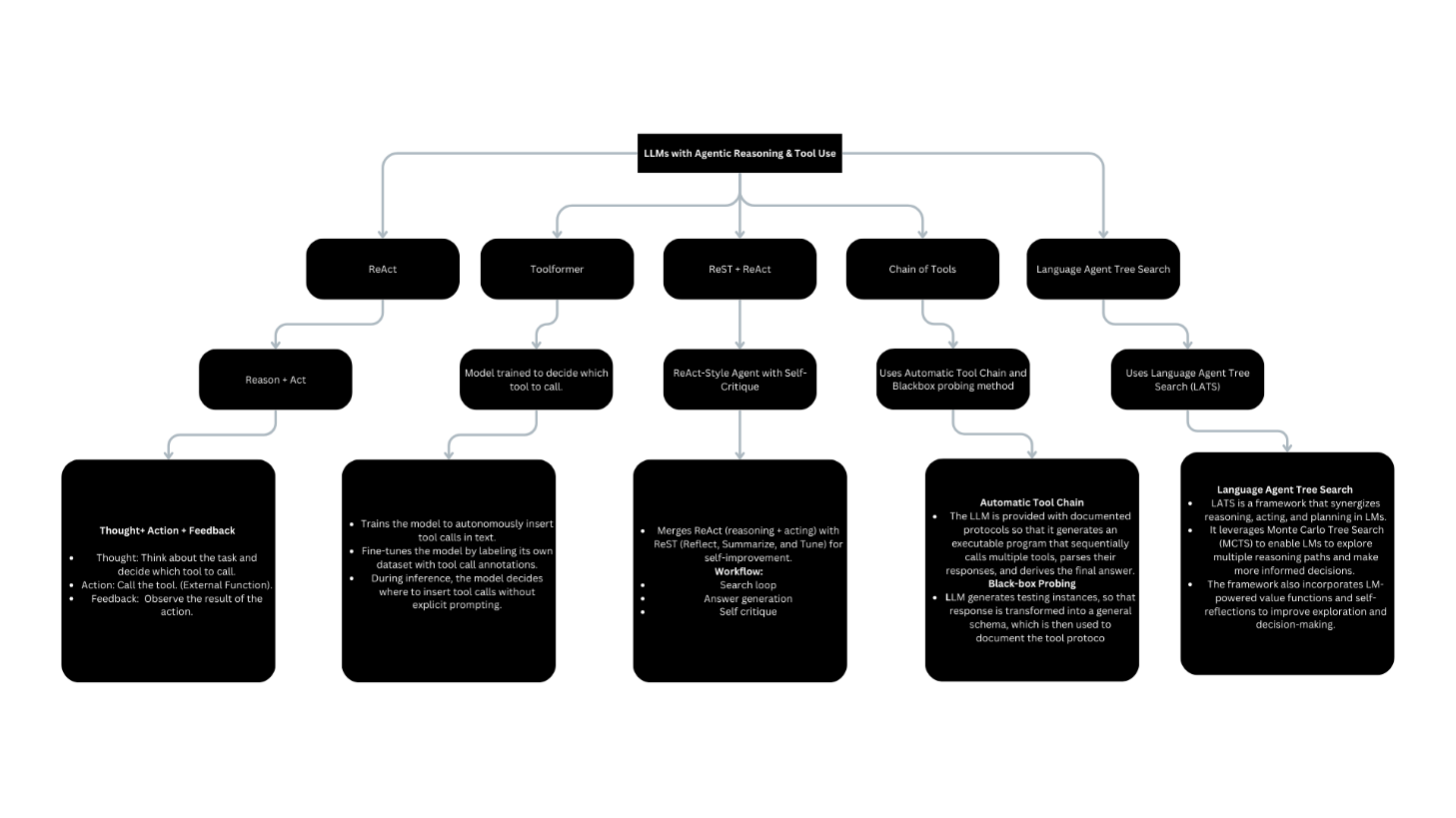
**Research Papers Analysis**

# Papers:

1. ReAct: Synergizing Reasoning and Acting in Language Models
2. Toolformer: Language Models Can Teach Themselves to Use Tools
3. ReST meets ReAct: Self-Improvement for Multi-Step Reasoning LLM Agent
4. Chain of Tools: Large Language Model is an Automatic Multi-tool Learner
5. Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models

# Conceptual Map:



# Analysis:

## ReAct:

### Concept:

* It combines reasoning with actionable outputs.
* Allows LM to interact with external env to retrieve information.

﻿﻿Key Features:

* ﻿﻿It is suitable for various applications across different domains.
* ﻿﻿It can quickly adapt to new tasks with less data.
* It demonstrates efficient learning capabilities.

### Working:

Thought -> Action -> Feedback

1. **Thought Generation**: Firstly, the model first generates **thought** (reasoning trace) about the task. This thought is a verbal reasoning (thinking) step that helps the model plan its next action. For e.g., the model might think: *"I need to search for info about X to answer the question"*
2. **Action Generation**: So, after thought generation, based on the thought, the model generates an **action** that it believes will help it progress toward solving the tasks. This action could involves interacting with an external env, such as querying a Wikipedia API or performing a google search. For e.g., the model might generate: *"Search[entity]"* to get information about a specific entity.
3. **Observation**: After the action is executed, the model receives an **observation** from the external env. This observation contains the result of the action, such as the text retrieved from a Wikipedia page or the result of a search query.
4. **Combined Output**: The model then combines the observation with its previous reasoning to generate the next thought or action. This process continues iteratively until the task is completed. For e.g., after observing the result of a search, the model might think: *"The search result indicates that X is true, so I need to look up Y next."*

Based on research papers, combination of **CoT and ReAct** performed best, which makes sense because during **Thought Generation** model can solve more complex task by calling function step by step.

Fine tuning smaller models using ReAct format trajectories, also results in best performance.

## Toolformer:

### Concept:

* LM can teach themselves to use external tools via simple API call.
* Basically, Toolformer is a model trained to decide when to call an API, which API to call, what arguments to pass, and how to incorporate the results into future token predictions.
* And this is done in a self-supervised way, requiring few demonstrations for each API.

### Key Features:

* **Self-Supervised Learning:** The Toolformer learns to use tools without requiring large amounts of human-annotated data. Because it generates its own training data by sampling potential APIs calls and filtering them based on whether they help in predicting future tokens.
* **Generalization**: It also retains its core language modelling abilities while learning to use tools. So basically, it can decide autonomously when and how to use tools, making it applicable across a wide range of tasks without being tied to specific use cases in instructions.
* **API Call Representation:** In LM output, API calls are represented as tuples and inserted into text using special tokens. The model learns to predict when to insert these calls and how to use the results. For e.g. -> (then tool name)

### Working:

1. **Sampling API Calls**:
   * Firstly, for each API, the model is prompted to generate potential API calls at various positions in the text like “I think I should call ->(weather API)”. The model samples positions where it thinks an API call would be useful.
   * Also, the model generates API calls by predicting the start and end of the API call sequence. like “I think I should call -><weather API>. And so on.”
2. **Executing API Calls**:
   * Then the generated API calls are executed, and the results are retrieved. For e.g., a calculator API call would return the result of a maths operation, while a search engine API call would return relevant info from Wikipedia.
3. **Filtering API Calls**:
   * Then model evaluates whether the API call and its result help in predicting future tokens. This is done by comparing the loss (perplexity) of the model with and without the API call.
   * The only API calls that significantly reduce the loss are retained for training.
4. **Model Fine-Tuning**:
   * Then the model is fine-tuned on a dataset augmented with the filtered API calls. This allows the model to learn when and how to use tools effectively.
   * The fine-tuning process ensures that the model retains its language modelling capabilities while improving its ability to use external tools.
5. **Inference**:
   * During inference, the model generates text as usual but can insert API calls when it feels necessity. When an API call is made, the model pauses, retrieves the result, and continues generating text based on the retrieved information.

After learning to use tools, Toolformer (which was GPT-J model) achieves strong zero-shot performance on various downstream tasks, outperforming much larger models like GPT-3.

But it cannot use the output of one tool as the input for another tool in a sequential manner.

## ReST meets ReAct:

### Concept:

* Its ReAct-style agent that combines reasoning and action, leveraging external knowledge retrieval to improve its ability to answer open-ended, knowledge-seeking questions.
* The key innovation is the use of a ReST-like (Reinforced Self-Training) algorithm to iteratively improve the agent's performance through self-critique, AI feedback, and synthetic data generation, without relying on human-labelled data.

### Key Features:

* **ReAct-Style Agent**: The agent combines reasoning (CoT) with actions (e.g., web search) and observations to answer complex questions. It uses a state machine to guide the reasoning process, ensuring that the agent can handle multi-step reasoning tasks effectively.
* **Self-Improvement using ReST:** The agent improves itself through an iterative process of genearating synthetic data, fine-tuning on that data, and using AI feedback to rank and filter the best actions. This process is inspired by the ReST algorithm, which grows the dataset by sampling from the latest policy and improves the policy by fine-tuning on the best samples.
* **AI Feedback:** Instead of relying on human-labelled data, the agent uses AI feedback to rank and select the best reasoning steps.
* **Self-Distillation:** The synthetic data generated during the self-improvement process is used to distill the agent into smaller models (e.g., PaLM 2-XS, S) while maintaining comparable performance to the larger teacher model (PaLM 2-L).
* **Auto-Evaluation:** The auto-eval is highly correlated with human evaluations.
* **Multi-Step Reasoning:** The agent is designed to handle tough questions that require multiple reasoning steps, such as those found in the Bamboogle and BamTwoogle datasets.

### Working:

1. **Agent Flow**: Firstly, we talk about the agent follows a structured flow (shown below):
   * It receives a question and decides whether it needs additional information. Like does it requires to use an external tool.
   * If more information is needed, it performs a web search, or some other task, and repeats the process until it has enough information to generate an answer.
   * Once the search loop is complete, the agent generates a draft answer and performs self-critique to ensure the answer is relevant and grounded in the retrieved information.
2. **Prompting**: The agent uses few-shot prompts formatted as Python code to guide its reasoning steps. This structured approach ensures that the agent's input and output are easily parsable and can be integrated with external tools.
3. **Fine-Tuning**: The agent's trajectories (seq of reasoning steps) are split into individual steps and used to create a fine-tuning mixture. The agent is then fine-tuned on this mixture, with the best samples selected using AI feedback.
4. **Iterating Improvement**: The agent undergoes multiple iterations of self-improvement. In each iteration, it generates new trajectories, fine-tunes on the best samples, and improves its performance. This process is repeated until the agent reaches a satisfactory level of performance.
5. **Eval**: The agent's performance is evaluated on the Bamboogle and BamTwoogle datasets, which consist of complex, multi-step questions. The authors use auto-eval to measure the agent's accuracy and ensure that it can handle open-ended questions effectively.
6. **Self-Critique**: The agent performs self-critique steps to verify that its answers are relevant to the original question and grounded in the retrieved information. This self-critique process provides a small but measurable boost to the agent's performance.

## Chain of Tools:

### Concept:

* Enable LLMs to generate a chain of tools programmatically.
* Enable LLMs to discover and document new tool usages, teaching themselves to master new tools.

### Working:

**Automatic Tool Chain (ATC):**

So, basically ATC framework, which enables LLMs to generate a chain of tools programmatically. The framework consists of **two** main components:

* **Chain of Tools Generation:**

LLM given a task, the LLM is provided with documented protocols for each tool in the candidate toolset. These protocols contain meta-information such as argument requirements, tool descriptions, and response schemas. The LLM then generates an executable program that sequentially calls multiple tools, parses their responses, and derives the final answer. This approach allows the LLM to learn the input-output schema and data flow dependencies among tools, enabling it to plan and execute complex tasks programmatically.

**Programming with Attributable Reflection**

To handle runtime errors in the generated programs, an **attributable reflection mechanism** is used. When a runtime error occurs, the LLM is guided to attribute the error to a specific tool, and pinpoint the incorrect tool usage, and revise the program accordingly. This mechanism iterates until the program executes successfully or reaches a maximum number of iterations.

* **Black box Probing for Toolset Extension**

TO solve the challenge of manually crafting documents protocols for diverse and fast-paced tools, the **black-box probing method** is used that enables LLMs to act as **active tool learners**. This method consists of two phases:

**Tool Probing**

The LLM generates testing instances that target the functionality of a tool, including relevant tasks and tool-use program solutions. The the tool's response is transformed into a general schema, which is then used to document the tool protocol. This process allows the LLM to probe the input-output schema of new tools and teach itself how to use them.

**Chain of Probing**

Some tools may require private arguments that are only accessible through other tools. To address this, the paper introduces a **chain of probing algorithm** that optimizes the cooperation among tools with strong input-output dependencies. The algorithm probes tools in an order that ensures the necessary arguments are available, enabling the LLM to master interconnected tools.

## Language Agent Tree Search:

### Concept:

* It is a framework that synergizes reasoning, acting, and planning in LM.
* LATS leverages **Monte Carlo Tree Search (MCTS)** to enable LMs to explore multiple reasoning paths and make more informed decisions.
* The framework also incorporates **LM-powered value functions** and **self-reflections** to improve exploration and decision-making.

### Working:

* LM Agent supports sequential reasoning or decision-making tasks. The agent receives observations from the env and takes actions based on a policy. The action space includes both permissible actions and reasoning traces.
* **LATS Framework**: The core of LATS is a search algorithm that uses MCTS to explore and evaluate multiple reasoning paths. The framework consists of six operations:
  1. **Selection**: Choose the most promising node for expansion.
  2. **Expansion**: Sample new actions and add child nodes to the tree.
  3. **Evaluation**: Assign a value to each new node based on LM-generated scores and self-consistency.
  4. **Simulation**: Expand the selected node until a terminal state is reached.
  5. **Backpropagation**: Update the values of nodes based on the outcome of the simulation.
  6. **Reflection**: Generate self-reflections to refine future decision-making.
* **Value Function**: The value function combines LM-generated scores and self-consistency to guide the search process. This allows LATS to balance exploration and exploitation effectively.

# Open Questions (Challenges):

1. In **ReAct**, the current approach relies on prompting llm with few-shot examples, which limits the complexity of reasoning and acting behaviours that can be supported. The input length constraints of in-context learning can restrict the number of demonstrations that can be provided, making it challenging to learn complex tasks with large action spaces.

We can train LM to more tasks could unlock further potential. Multi-task training could help the model generalize better across diverse domains and improve its ability to handle complex tasks.

1. In **Toolformer,**
   * When LM does not find any relevant result from the tool, that significantly affects the reasoning as shown in papers.
   * Also, it can use tools in chain which means output of one tool cannot be used as input to another tool.
   * Also, cannot use tool in interactive way.
2. In **ReAct - ReST** method, the iterative self-improvement process requires significant computational resources, especially due to repeated agent trajectory generations and auto-evaluations. Also, it requires large models like PaLM 2-L. It also requires manual few-shot prompt construction.

Some improvements could be to optimize sampling and fine-tuning strategies to make training and inference more **efficient**. We can also explore programmatic prompt generation for **automated prompt tuning** techniques.

1. The current **Chain of Tools** framework is limited to text-based tools and does not handle multi-modal tasks like image or speech inputs. The framework stops iteration if no runtime errors are raised, but the program may still produce incorrect answers. Future work will focus on calibrating program outputs to address this issue.
2. The biggest limitation of **LATS** is it has a higher computational cost compared to simpler prompting methods, and it assumes the ability to revert to earlier states in decision-making env.

Future work could be scaling LATS to more complex environments, improving efficiency, and exploring multi-agent frameworks.