**Section A:**

By nature, LLM have zero shot, and few shot reasoning but few limitations they exhibit like Fact checking and retrieval, temporal awareness, language selection, etc., so to avoid this there are few papers introduced in Reasoning and Action which enhance the output by giving proper outcomes instead of returning the delusion answers. Simple examples for agentic reasoning like in ReAct, it eliminates Chain of Thoughts and introduces Reasoning and Action on top of it, hence eliminates the delusional answer because of any false results during execution. Toolformers introduce many tools and api calls such as calculator, website and logical reasoning api to enhance the thought process operation in the reasoning part by elimination false answer and increase the accuracy. In the ReStT – ReAct method further increases the accuracy of the Reaction and Action method by training in the result of those operation and evaluation the correctness and returning the results. In Chain of Tools method, it enables the LLM to use multiple tools at the same time to reduce the inference time and provide more factual data for inference to reason it and generate the result. The Language Agent Tree method works by incorporating a tree search algorithm known as Monte Carlo tree, in which the main motive is to increase the efficiency of the model in which this method is mainly focused on Realtime text generation and code completion algorithms.

**Section B:**

**ReAct: Synergizing Reasoning and Acting in Language Models: Working**

LLMs often generate incorrect or hallucinated facts when reasoning in isolation. By integrating reasoning with actions that retrieve information, ReAct reduces these errors. For example, in a question-answering task, the model can first reason about the possible answer, then act by querying a database before making a final decision. Traditional LLMs often produce blackbox outputs with no clear explanation of how they arrived at a conclusion. ReAct improves transparency by allowing humans to see both the reasoning process and corresponding actions taken by the model.

**Toolformer: Language Models Can Teach Themselves to Use Tools: Working**

Toolformer let identify when and what tools is need for the operation and it have its own tools of operation such as Arithmetic calculations, Api call generator, calling external Api's such as calculator, search engine, translation system. This also helps in the model evaluate whether the API response helps reduce uncertainty in token prediction. Key takeaways of toolformer are Autonomous Decision, Self-Supervised Learning, Improved Reasoning Capabilities, Generalization Across Tasks

**ReST meets ReAct: Self-Improvement for Multi-Step Reasoning LLM Agent: Working**

This method have all the functionality of ReAct implementation, but on top of that it actually train on the result and tries to find the optimal solution for the query. Reinforced Self-Training (ReST) that allows the agent to learn from its previous reasoning trajectories. The ReST process involves iteratively collecting trajectories, filtering and ranking them based on quality, and fine-tuning the model with the refined data. This iterative self-improvement boosts both the performance and robustness of the LLM agent, allowing it to better handle complex reasoning tasks in future iterations

**Chain of Tools: Large Language Model is an Automatic Multi-tool Learner: Working**

This method introduces the Automatic Tool Chain (ATC) framework, allowing LLMs to effectively use multiple tools in a more dynamic way. The study also presents a black box probing method that enables LLMs to learn and document new tool usages independently. A new benchmark named ToolFlow has been established to evaluate the performance of this approach in complex scenarios requiring long-term planning.ATC framework allows LLMs to act as multi-tool users. This enables the LLMs to automatically generate a sequence of tool calls necessary to solve a given task, utilizing the protocols that provide meta-information about how tools operate. The LLM can parse tool responses, retrieve useful data, and derive a final solution, thus facilitating complex task execution without relying solely on a step-by-step approach

**Language Agent Tree Search Unifies Reasoning, Acting, and Planning in Language Models: Working**

This method works by combining Monte Carlo Tree Search with language models, which allows for better decision-making through methods like in-context learning, value functions, and self-reflection. The framework also benefits from external feedback, which improves its problem-solving abilities. The key features of this framework are Monte Carlo Tree Search (MCTS) Integration where LATS utilizes MCTS to explore a combinatorial space of reasoning and acting steps, enabling better planning and decision-making. It also has Reflection and Memory Use which incorporates a mechanism for the model to reflect on past actions, storing both failed attempts and successful strategies for future reference and finally General Applicability in which the framework is designed to be applicable across various reasoning and decision-making tasks, offering a structured approach to problem-solving

**Section C**

There are two ways to deploy the ai model either setup our own server with high gpu or get model api from vendors such as gemini models, aws bedrock or huggingface inference api, and setup a simple backend to just run the inference. Both have it own merits and demerits

**Inhouse LLM deployment:**

- **merits**
  - Have full control to the LLM and finetuning
  - Can have unlimited token calls since it will be in house
  - Can have multiple model selections
- **Demerits**
  - Need a higher gpu cloud instance which will increase in cost if higher gpu runs for 24 hrs
  - Become complex and might get shortage in gpu vram with increase in load
  - Load balancers can be implemented to avoid the traffic but won't efficiently tackle huge traffic

**Using LLM API:**

- **Merits**
  - No need to worry about multiple calls to agent or llm since a simple async function can be used to optimize the operation
  - Can be deployed in any simple vpc server where the main operation for the backend server is only to handle request -> api call -> response.
  - They are cheaper compared to in-house deployment and easy to maintain
- **Demerits**
  - The main demerits are the Tokenization, one should be aware of the number of tokens are passed in both input and output towards the api, since the cost of LLM API are purely with respect to number of tokens
  - The backend becomes complex by introducing methods to simplify the cache data and history if not will lead to huge cloud api cost.