LLM Agent for Shoppin': Conceptual Map, Analysis, and Challenges

A. Conceptual Map: How LLMs Reason and Use External Tools

1. Overview:

Agentic LLMs refer to large language models (LLMs) that can autonomously plan, reason, and act by interacting with external tools, APIs, or databases to solve complex tasks. Unlike static LLMs that generate direct text responses, agentic LLMs break down problems into structured actions, making them more effective in real-world applications.

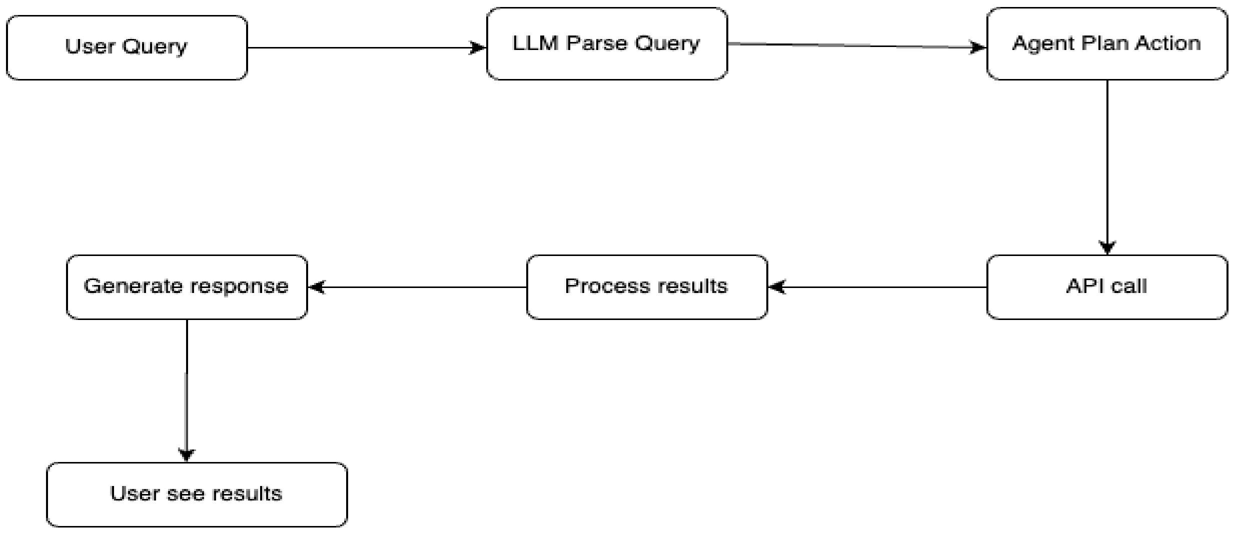
Key Capabilities of Agentic LLMs

1. Intent Understanding – Analyzing user input to extract structured queries.
2. Planning & Reasoning – Determining which external tools or APIs to use.
3. Tool Invocation – Calling APIs, retrieving external data, and processing results.
4. Response Generation – Synthesizing a final, coherent response from multiple sources.
5. Self-Improvement – Iteratively refining responses based on feedback.

3. Connections Across Research Papers

Paper	**Contribution to LLM Agent**
ReAct	Intertwines reasoning and acting, enabling structured API calls.
Toolformer	Trains models to decide when API calls are necessary.
ReST meets ReAct	Adds self-improvement techniques for multi-step reasoning agents.
Chain of Tools	Enables dynamic chaining of multiple tools in response to queries.
Language Agent Tree Search	Improves hierarchical planning and retrieval for complex fashion-related queries.

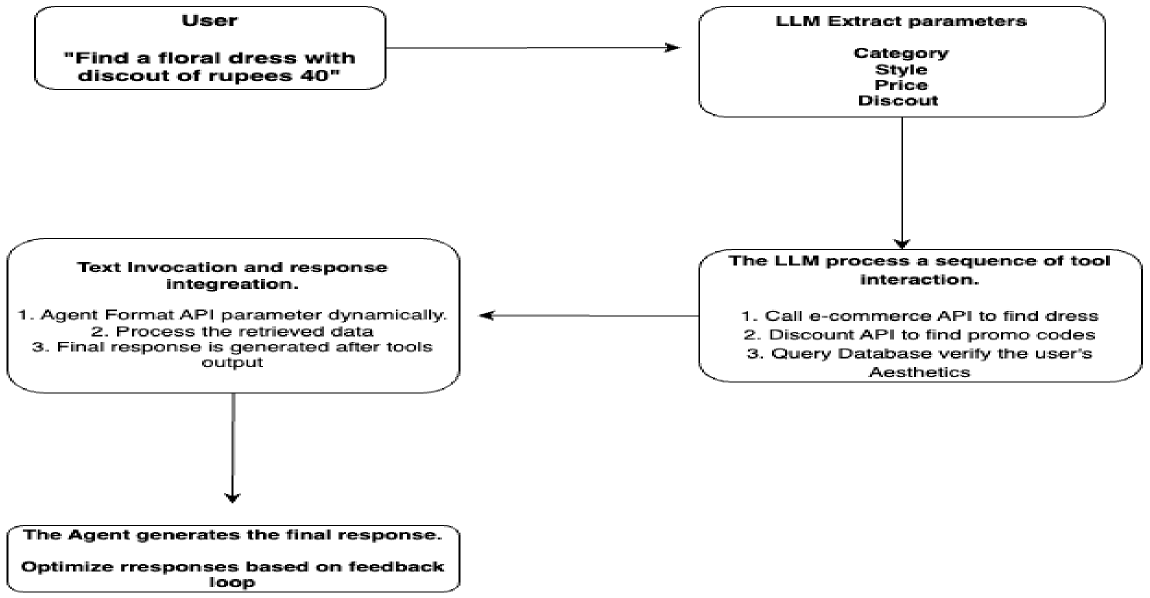
2. Flow of the LLM Agent



The Role of LLM in Shoppin'

Shoppin is an AI-powered **visual fashion search engine** that enables users to find outfits using **text queries, images, and constraints** (e.g., budget, brand, style). To improve search efficiency, the **LLM-powered agent** interacts with external tools such as:

E-commerce APIs – Fetch products based on style, price, and availability.
Trend Analyzers – Identify trending fashion based on social media and reviews.
Discount & Coupon Checkers – Validate promo codes and find real-time deals.


User
"Find a floral dress with discout of rupees 40"
LLM Extract parameters
Category
Style
Price
Discout
The LLM process a sequence of tool interaction.
1. Call e-commerce API to find dress
2. Discount API to find promo codes
3. Query Database verify the user's Aesthetics
Text Invocation and response integreation.
1. Agent Format API parameter dynamically.
2. Process the retrieved data
3. Final response is generated after tools output
The Agent generates the final response.
Optimize rresponses based on feedback loop

Comparison of Agentic Designs

(1) ReAct (Reason + Act)

ReAct integrates reasoning and action in a loop, allowing the agent to think step-by-step while interacting with tools. This approach is transparent since it verbalizes its reasoning before making a decision.

- Reasoning Steps:
  - Breaks down the query
  - Thinks through possible actions
  - Calls external tools when necessary
  - Iterates based on feedback
- Tool Use: Selectively queries databases, search engines, or APIs while justifying each step.

(2) Toolformer

Toolformer enhances traditional LLMs by incorporating tool usage without explicit reasoning. The model is trained to predict when and how to call external APIs without needing human-like step-by-step thought processes.

- Reasoning Steps:
  - Identifies tool use points within the query
  - Automatically executes API calls
  - Compiles results directly into responses
- Tool Use: Efficiently integrates APIs without over-reasoning.

(3) Chain of Tools

This approach sequences multiple tools logically, ensuring that dependencies between steps are handled efficiently. It follows a structured pipeline where each tool feeds into the next.

- Reasoning Steps:
  - Identifies the sequence of tasks
  - Calls relevant tools in a structured order
  - Aggregates results into a coherent response
- Tool Use: Highly structured, ensuring that data from one tool informs subsequent steps.

(4) Language Agent Tree Search (LATS)

LATS explores multiple solution paths in parallel, evaluating different possibilities before selecting the best outcome. It mimics a tree search algorithm, ensuring optimal decision-making.

- Reasoning Steps:
    - Generates multiple solution paths
    - Evaluates different results in parallel
    - Picks the most optimal result based on set criteria
- Tool Use: Uses a mix of exploratory search and validation tools.

Contrast methodologies

ReAct follows a step-by-step reasoning approach, making it ideal for tasks like customer support where the agent needs to think through responses. It selectively calls APIs when needed, maintaining a moderate speed.

Toolformer uses minimal reasoning but excels in automatic API integration, making it the fastest option for quick lookups, such as checking product prices.

Chain of Tools structures multi-tool reasoning, executing APIs in a sequential manner. This makes it effective for handling complex queries, like assembling an outfit based on an occasion, though its speed remains moderate.

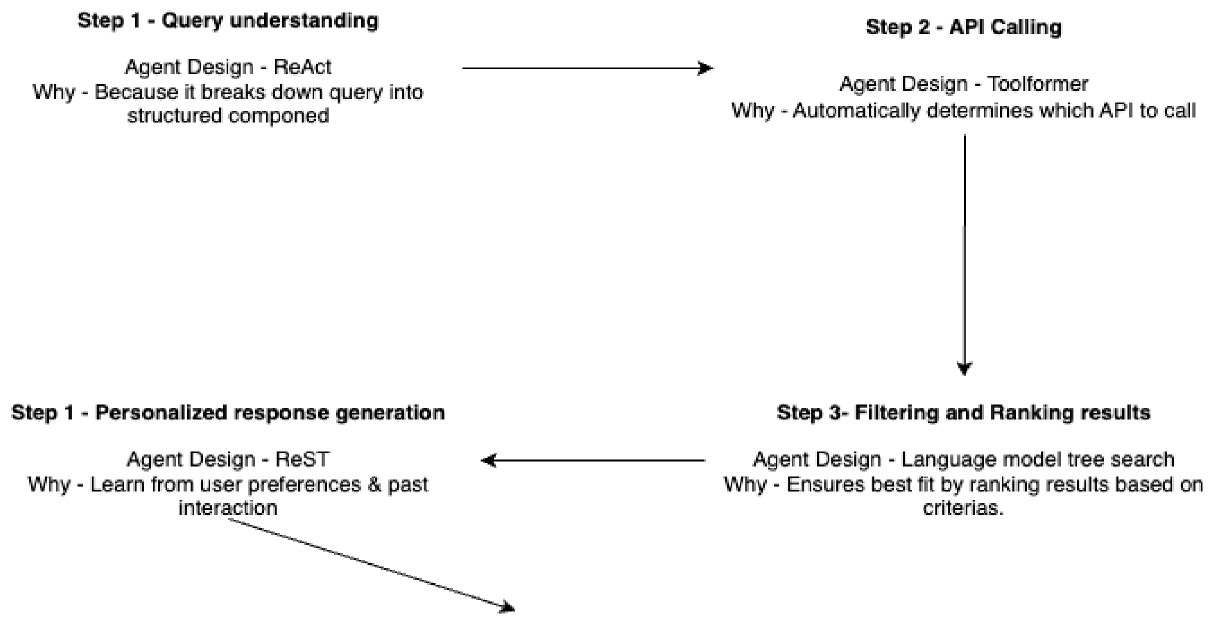
ReST enhances **step-by-step reasoning** by integrating structured API calls using RESTful services. This allows agents to **seamlessly interact with multiple APIs** while maintaining a logical flow.

LATS takes a parallel exploration approach, evaluating multiple paths simultaneously. It is the slowest but best suited for personalized recommendations, such as style matching based on user preferences.

Real world application

We will discuss a real world example for shoppin and how we can use different system to improve efficiency

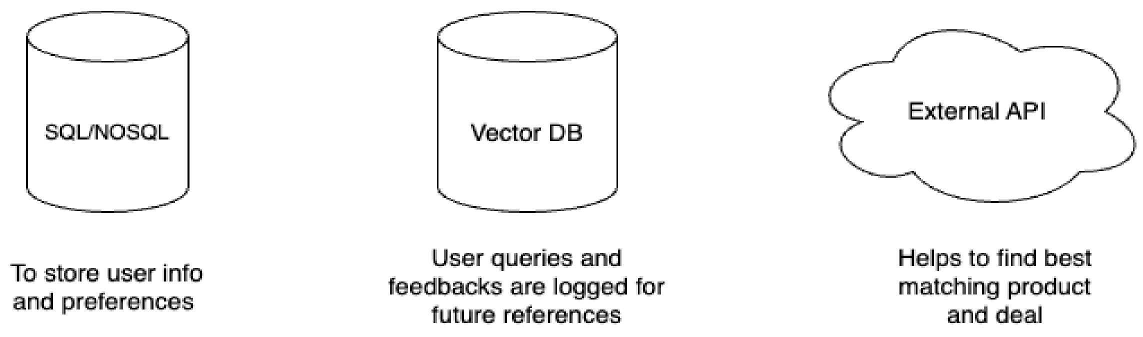
Query - Find me a pastel blue dress $50 that matches my skin tone and suggest accesoried



Step 1 - Query understanding
Agent Design - ReAct
Why - Because it breaks down query into structured componed
Step 2 - API Calling
Agent Design - Toolformer
Why - Automatically determines which API to call
Step 3- Filtering and Ranking results
Agent Design - Language model tree search
Why - Ensures best fit by ranking results based on criterias.
Step 1 - Personalized response generation
Agent Design - ReST
Why - Learn from user preferences & past interaction

"Based on your preferences, here are 3 pastel blue dresses under $50 that complement your skin tone. We've also picked accessories that match!"

Data sources involved



SQL/NOSQL
To store user info and preferences
Vector DB
User queries and feedbacks are logged for future references
External API
Helps to find best matching product and deal

Deployment Challenges

1. Scalability Issues

Challenge: Handling millions of users simultaneously while making multiple API calls for personalized recommendations.

2. Adaptability to Dynamic Inventory

Challenge: Real-time stock updates, price fluctuations, and product availability affect recommendations.

3. Handling Errors in Multi-Step Reasoning

Challenge: Incorrect API calls, missing data, or broken reasoning chains leading to poor user experience.

4. Complex Integration with External APIs

Challenge: Different APIs have varied response formats, authentication mechanisms, and rate limits.

Future Implementation

Users interact with AI in diverse ways, ranging from **simple fact-based queries** to **complex, multi-step reasoning tasks**. However, relying on a **single large, complex AI model** to handle all queries is inefficient and costly.

- **Basic queries** (e.g., *“Show me trending sneakers”*) do not require complex reasoning.
- **Advanced queries** (e.g., *“Find the best running shoes under ₹3000, considering user reviews and return policies”*) involve multiple API calls, logical reasoning, and external data aggregation.
- **High-frequency usage** results in **expensive LLM API calls**, increasing operational costs and slowing down responses.

Solution: A Hybrid AI Model Approach

We implement a **two-tier AI system**:1️⃣**A smaller, fine-tuned model** for **frequent and structured queries**2️⃣**A large LLM agent** for **complex, multi-step tasks**

This setup ensures **speed, scalability, and cost reduction**, while still providing **high-quality responses**.

Implementation Steps

1- Data Collection from Large LLM Responses

Before training a smaller model, we need **high-quality training data** from interactions with the large LLM. We collect:

- **User Queries & Responses** – Store how the LLM responds to different types of queries.
- **User Behavior & Feedback** – Track which recommendations users engage with (clicks, selections, edits).
- **API Call Patterns** – Log external data sources used by the LLM.

2-Training the Smaller Model (Fine-Tuning & Distillation)

Instead of continuously relying on expensive LLM calls, we train a **lightweight model** using **Knowledge Distillation (KD)**:

Steps:

1. **Extract LLM-generated responses** to create a **training dataset**.
2. **Fine-tune a small model** (e.g., DistilBERT, TinyBERT) to **mimic LLM outputs**.
3. **Apply Reinforcement Learning from User Feedback (RLHF)** to improve accuracy.
4. **Use Retrieval-Augmented Learning (RAG)** to allow the small model to fetch past responses instead of always querying the LLM.

3-Real-Time Query Handling: When to Use Each Model

Once trained, we deploy the small model as the **first layer of response generation**. The AI system follows this decision tree:

Query Categorization & Execution:

- **Basic Queries** → The **small model** provides instant responses.
- **Moderate Queries** → The **small model retrieves past responses** or **fetches lightweight API data**.
- **Advanced Queries** → The **complex LLM agent is called**, using multiple API integrations.

Benefits of This Approach

Reduced Costs – Lower LLM API usage saves money.

Faster Responses – Small models handle 70-80% of queries instantly.

Scalability – Can support **millions of users simultaneously**.

Personalization – Adapts to individual user preferences over time.