140509_22.md

README

22. Multi-Step Research Assistant Agent

Summary: Build an autonomous AI agent that plans, executes, and synthesizes multi-step research tasks by coordinating web searches, document analysis, and report generation.

Problem Statement: Complex research tasks require coordination across multiple information sources and analytical steps, which is time-consuming and error-prone. Your task is to create an agentic AI system that decomposes research queries into sub-tasks, retrieves information from diverse sources (web, documents, databases), analyzes findings, and generates comprehensive reports with source attribution. The system must demonstrate reasoning, adaptive execution, and maintain factual accuracy with credibility assessments.

Steps: - Design task decomposition and planning algorithms for research workflows. - Implement multi-source information retrieval (web search, database queries, document analysis). - Create reasoning chains for connecting disparate information. - Build synthesis and report generation with source attribution. - Develop fact-checking and credibility assessment mechanisms. - Include interactive refinement and follow-up question capabilities.

Suggested Data Requirements: - Sample research questions across domains (e.g., science, history). - Diverse information sources (web pages, academic papers, databases). - Fact-checking datasets (e.g., FEVER) and credibility scoring criteria. - Report templates and evaluation criteria.

Themes: Agentic AI, Agents & APIs

The steps and data requirements outlined above are intended solely as reference points to assist you in conceptualizing your solution.

PRD (Product Requirements Document)

Product Vision and Goals

The Multi-Step Research Assistant Agent aims to streamline complex research by automating task planning, information retrieval, and synthesis, reducing research time by 60% while ensuring high factual accuracy and transparency. Goals include supporting diverse domains (e.g., academia, journalism), delivering reports with traceable reasoning, and enabling users to refine outputs interactively, fostering trust and utility in professional and educational settings.

Target Audience and Stakeholders

- **Primary Users**: Students, researchers, journalists, policy analysts.
- **Stakeholders**: Content providers (for source access), educators (for teaching tools), AI ethicists (for fact-checking).
- **Personas**:
  - A graduate student researching climate policy impacts across countries.
  - A journalist investigating corporate financial scandals with source verification.

Key Features and Functionality

- **Task Decomposition**: Break down complex queries into actionable sub-tasks.
- **Multi-Source Retrieval**: Access web (via APIs), documents (PDFs), and databases.
- **Reasoning Chains**: Connect findings using chain-of-thought (CoT) reasoning.
- **Report Synthesis**: Generate structured reports with citations and summaries.
- **Fact-Checking**: Verify claims against reliable sources with credibility scores.
- **Interactive Refinement**: Allow users to adjust queries or focus areas dynamically.
- **Analytics**: Track popular research paths and user engagement metrics.

Business Requirements

- Integration with APIs (e.g., Google Search, Arxiv, Wikipedia).

- Freemium model: Basic research free, premium for advanced sources (e.g., paywalled journals).
- Export reports in PDF/Markdown for integration with tools like Notion or Overleaf.

Success Metrics

- **Accuracy**: >90% fact verification rate against FEVER dataset.
- **Engagement**: Average session time >10 minutes, >5 sub-tasks per query.
- **User Satisfaction**: NPS >70.
- **Efficiency**: Full research cycle <8 minutes for typical queries.

Assumptions, Risks, and Dependencies

- **Assumptions**: Access to open LLMs (e.g., Llama) and search APIs.
- **Risks**: Inconsistent source reliability; mitigate with credibility scoring and fallback sources.
- **Dependencies**: Public datasets (FEVER, MultiWOZ), APIs (Google, SerpAPI).

Out of Scope

- Real-time collaboration with human researchers.
- Multilingual source processing initially.

FRD (Functional Requirements Document)

System Modules and Requirements

1. **Task Decomposition Module (FR-001)**:
   - **Input**: Natural language research query (e.g., â€œImpact of renewable energy policies in Europeâ€).
   - **Functionality**: Use LLM to break into sub-tasks (e.g., â€œsearch policies,â€ â€œanalyze adoption ratesâ€) via prompt engineering (â€œDecompose query into steps: {query}â€).
   - **Output**: List of sub-tasks with priorities and dependencies.
   - **Validation**: Ensure sub-tasks cover query intent; validate via user feedback.
2. **Retrieval Module (FR-002)**:
   - **Input**: Sub-task (e.g., â€œsearch policiesâ€).
   - **Functionality**: Execute web searches (SerpAPI), parse PDFs (PyPDF2), query DBs (SQL/NoSQL); rank results by relevance.
   - **Output**: Structured data (e.g., JSON with text snippets, URLs, metadata).
   - **Validation**: Filter low-relevance results using embeddings (e.g., Sentence Transformers).
3. **Reasoning Chain Module (FR-003)**:
   - **Input**: Retrieved data across sub-tasks.
   - **Functionality**: Apply CoT prompting in LLM to link findings (e.g., â€œPolicy X led to Y% increase in solar adoptionâ€).
   - **Output**: Reasoning paths as text or JSON (e.g., {â€œstepâ€: â€œpolicy Xâ€, â€œeffectâ€: â€œY% increaseâ€}).
   - **Validation**: Check logical consistency using rule-based checks.
4. **Synthesis and Report Module (FR-004)**:
   - **Input**: Reasoning paths, raw data.
   - **Functionality**: Generate formatted report with sections (intro, findings, conclusion), include citations (APA/ML format).
   - **Output**: Markdown/PDF report with embedded sources.
   - **Validation**: Ensure all claims are sourced; flag uncited claims.
5. **Fact-Checking Module (FR-005)**:
   - **Input**: Report claims.
   - **Functionality**: Cross-verify against reliable sources (e.g., Wikipedia, FEVER dataset) using embeddings; assign credibility scores (0-1 based on source rank).
   - **Output**: Annotated report with credibility flags (e.g., â€œClaim X: 0.9 confidenceâ€).
   - **Validation**: Manual override for flagged low-confidence claims.
6. **Interactive Refinement Module (FR-006)**:
   - **Input**: User feedback (e.g., â€œfocus on Germanyâ€).
   - **Functionality**: Adjust sub-tasks or re-run retrieval; update report dynamically.
   - **Output**: Revised report and session state.
   - **Validation**: Track feedback loops to prevent infinite iterations.

Interfaces and Integrations

- **UI**: Web-based conversational interface (React, chat-style) for query input, feedback, and report viewing.
- **API**: RESTful endpoints (e.g., POST /research, POST /refine) with JSON payloads.
- **Data Flow**: Query -> Decompose -> Retrieve -> Reason -> Synthesize -> Fact-check -> Output/Refine.
- **Integrations**: SerpAPI for web, Arxiv API for papers, MongoDB for caching results.

Error Handling and Validation

- **Invalid Query**: Return clarification prompt (e.g., â€œPlease specify regionâ€).
- **Unreliable Source**: Flag and seek alternatives (e.g., skip low-ranked sites).
- **Tests**: Unit tests for decomposition (90% coverage), E2E for full cycle.

NFRD (Non-Functional Requirements Document)

Performance Requirements

- **Latency**: Full research cycle <8 minutes for queries with <10 sub-tasks.
- **Throughput**: Handle 100 concurrent users on standard cloud infra (16GB RAM, 4 vCPUs).

Scalability and Availability

- **Scalability**: Containerized (Docker) with Kubernetes for load balancing.
- **Availability**: 99.9% uptime; redundant API servers.

Security and Privacy

- **Data Privacy**: Avoid storing sensitive queries; encrypt cached results.
- **Authentication**: OAuth2 for user access; API key for external integrations.
- **Compliance**: GDPR for user data, audit logs for query tracking.

Reliability and Maintainability

- **Error Rate**: <1% failure rate for retrieval tasks.
- **Code Quality**: Modular design, 85% test coverage, CI/CD with GitHub Actions.
- **Monitoring**: Prometheus for API latency, retrieval success rates.

Usability and Accessibility

- **UI/UX**: Responsive design, WCAG 2.1 AA compliance (e.g., keyboard navigation).
- **Documentation**: API docs via Swagger, user guides with examples.

Environmental Constraints

- **Deployment**: Cloud-agnostic (AWS/GCP) or on-prem.
- **Cost**: Optimize for <0.05 USD per research query.

AD (Architecture Diagram)

Pseudocode

```python class ResearchAgent: def **init**(self): self.llm = LangChainLLM(â€œmeta-llama/Llama-3-8bâ€) self.retriever = Retriever(serpapi_key, mongo_uri) self.fact_checker = FactChecker(fever_dataset) self.state = RedisClient()

```
def decompose(self, query):
    prompt = f"Decompose into sub-tasks: {query}"
    sub_tasks = self.llm.generate_json(prompt)
    return sorted(sub_tasks, key=lambda x: x["priority"])

def retrieve(self, task):
```
```

```
    if "search" in task:
        results = self.retriever.web_search(task["query"])
    elif "analyze" in task:
        results = self.retriever.doc_extract(task["file"])
    else:
        results = self.retriever.db_query(task["query"])
    ranked = rank_results(results, embeddings_model)
    return ranked[:5]  # Top 5 results

def reason(self, findings):
    prompt = f"CoT: Given facts {findings}, reason to answer {task}"
    reasoning = self.llm.generate_json(prompt)
    return reasoning

def synthesize(self, reasoning, findings):
    report = "# Research Report\n"
    for step in reasoning:
        report += f"## {step['step']}\n{step['evidence']}\n"
    return report

def fact_check(self, report):
    claims = extract_claims(report)
    for claim in claims:
        score = self.fact_checker.verify(claim)
        report = annotate(report, claim, score)
    return report

def refine(self, report, feedback):
    state = self.state.load(report_id)
    state["focus"] = feedback
    new_tasks = self.decompose(f"Refined query: {state['query']} with {feedback}")
    return self.run(new_tasks, state)

def run(self, query):
    tasks = self.decompose(query)
    findings = []
    for task in tasks:
        results = self.retrieve(task)
        findings.append(results)
    reasoning = self.reason(findings)
    report = self.synthesize(reasoning, findings)
    checked_report = self.fact_check(report)
    state_id = self.state.save({"query": query, "report": checked_report})
    analytics.log_path(state_id, tasks)
    return checked_report, state_id
```