Optimierung des IMAP Agenten-Systems: Informationsfluss und Effizienz

1. Einleitung

Ziel dieser Optimierungsstrategie ist es, das IMAP Democratic Agent System effizienter und robuster zu gestalten. Kernprobleme wie hohe Token-Nutzung und das Überschreiten von Rate-Limits (besonders bei Modellen wie Claude Sonnet) sollen adressiert werden. Dies erreichen wir durch die konsequente Umsetzung eines selektiven Informationsflusses ("Need-to-Know"-Prinzip), die Verfeinerung der Aufgabenbearbeitung und eine modularere Systemarchitektur, gestützt durch aktuelle Forschungsergebnisse.

2. Grundprinzip: Selektiver Kontext und "Need-to-Know"

Das aktuelle Verhalten, bei dem potenziell alle Agenten Zugriff auf einen sehr breiten Kontext haben, wird zugunsten eines strikten "Need-to-Know"-Prinzips aufgegeben.

- **Vorteile:**
  - **Reduzierte Token-Nutzung:** Agenten erhalten nur die für ihre aktuelle Aufgabe relevanten Informationen, was die Größe der Prompts und damit die API-Kosten und die Gefahr von Rate-Limiting signifikant senkt.
  - **Fokussiertere Agenten:** Ein kleinerer, relevanter Kontext hilft den LLMs, sich auf die Kernaufgabe zu konzentrieren und präzisere Ergebnisse zu liefern.
  - **Verbesserte Sicherheit und Datenintegrität:** Agenten können nicht versehentlich auf sensible oder für sie irrelevante Daten zugreifen oder diese modifizieren.
- **Umsetzung:** Erfolgt primär durch Anpassungen im ProjectWorkflowManager und in der Definition der Task-Objekte in CrewAI.

3. Strategische Umstrukturierung: Modularisierung der Agenten-Definitionen

Um die Optimierung und das Testen einzelner Agenten zu erleichtern, wird eine Modularisierung der Agenten-Definitionen vorgenommen.

- **Ziel:** Jeder Agent wird in einer eigenen Python-Datei (.py) definiert.
- **Vorteile:** Verbesserte Maintainability, unabhängiges Testen, fokussierte Optimierung, klarere Verantwortlichkeiten.
- **Strukturvorschlag:** Ein neuer Ordner (z.B. agent_definitions) mit separaten Dateien pro

Agent.
- **Vorgehen:** Nach der Modularisierung können einzelne Agenten gezielt auf Robustheit und Effizienz optimiert werden.

4. Verbesserungen im ProjectWorkflowManager und der Task-Definition

Der ProjectWorkflowManager (PM) wird zur zentralen Instanz für die Steuerung des Informationsflusses und der Kontextbereitstellung.

4.1. Kontextübergabe an Agenten: Dynamisches Kontextmanagement

- **PM als "Scratchpad-Orchestrator" und "semantischer Filter":**
  - Der PM lädt zu Beginn eines Planungsschritts oder eines neuen Entwicklungs-Steps die *relevanten* übergeordneten Informationen.
  - **Kernaufgabe (basierend auf Recherche Teil 1, Abschnitt I):** Der PM konstruiert aktiv den minimalen, semantisch reichen Kontext für andere Agenten. Dies beinhaltet:
    - **Identifizierung des Informationsbedarfs** der nächsten Aufgabe/des nächsten Agenten.
    - **Selektiver Abruf/Auswahl relevanter Daten** (z.B. mittels RAG-Techniken auf Task-Ebene, um spezifische Projektdetails oder Code-Snippets zu laden).
    - **Intelligente Zusammenfassung und Synthese** der ausgewählten Informationen (Nutzung des text_summarization_tool mit klarem summary_focus).
    - **Formatierung und Bereitstellung** des Kontexts in einer für den Zielagenten optimalen Form.
  - Der PM nutzt "Scratchpad"-ähnliche Operationen (Daten laden, verarbeiten, extrahieren) zur Kontexterstellung.
- **Strategische Nutzung des CrewAI-Gedächtnissystems (Recherche Teil 1, Abschnitt I.D.2):**
  - **ShortTermMemory (STM):** Aktivieren (memory=True in der Crew-Definition) für den automatischen Fluss aktueller Interaktionskontexte zwischen sequenziellen Tasks.
  - **EntityMemory (EM):** Evaluieren für wiederkehrende Schlüsselkonzepte, Dateien oder Anforderungen, um ein konsistentes Verständnis über Aufgaben hinweg zu gewährleisten. Der PM könnte Tasks erhalten, die explizit EM-Einträge aktualisieren oder abfragen.
- **Explizite Kontextdefinition in Tasks:**
  - Jede Task in CrewAI muss so definiert werden, dass sie explizit nur die für den Agenten notwendigen Informationen als Input erhält (z.B. über description oder den context-Parameter der Task, falls für strukturierte Datenübermittlung geeignet).

4.2. Strikte Implementierung der files_access.json: "Need-to-Know"-Dateizugriff

- **Mechanismus im ProjectWorkflowManager (basierend auf Recherche Teil 1, Abschnitt IV.D.2 & IV.D.3):**
    - Der PM agiert als **Policy Decision Point (PDP)**. Er lädt und interpretiert die files_access.json für den aktuellen Step.
    - Er stellt eine Schnittstelle bereit, über die Berechtigungen abgefragt werden können.
- **Implementierung eines Dateizugriffs-Proxy/Fassade (Recherche Teil 1, Abschnitt IV.D.1):**
    - **Empfehlung:** Refaktorierung von file_operations_tool.py. Alle Dateizugriffsfunktionen (read_file_tool, write_file_tool etc.) leiten ihre Aufrufe über eine interne Proxy-Schicht.
    - Diese Proxy-Schicht prüft vor jeder Dateioperation die Berechtigung des aufrufenden Agenten (Kontext muss übergeben werden) gegen die vom PM bereitgestellten Regeln.
    - Bei Bedarf kann die Untersuchung von PyCasbin für eine komplexere Richtlinienverwaltung in Betracht gezogen werden (Recherche Teil 1, Abschnitt IV.D.4).
- **Protokollierung:** Alle Dateizugriffsversuche (erlaubt und verweigert) sollten protokolliert werden.

4.3. Detailliertere, atomare Tasks: Insbesondere für Code-Generierung

- **Aufbrechen großer Aufgaben (basierend auf Recherche Teil 1, Abschnitt II.D.1 & II.D.3):**
    - Der PM zerlegt Features/User Stories in eine Sequenz atomarer (Code-Generierungs-)Aufgaben.
    - Jede atomare Aufgabe im plan.md spezifiziert: singuläres Ziel, erforderliche Inputs (z.B. Output der vorherigen atomaren Aufgabe), erwartetes Output-Format.
    - **Frontend-Struktur als Leitfaden:** Zerlegung nach dem Muster HTML -> CSS -> JS pro Komponente/Sektion.
- **SCoT (Structured Chain-of-Thought) Prompting für Developer (Recherche Teil 1, Abschnitt II.D.2):**
    - Prompts für den Developer Agent sollten ihn anleiten, vor der Code-Generierung die Logik mittels Sequenz-, Verzweigungs- und Schleifenkonstrukten zu planen.
- **Abhängigkeitsmanagement (Recherche Teil 1, Abschnitt II.D.4):**
    - Der Plan des PM muss die Sequenz klar definieren. Die Ausgabe einer atomaren Aufgabe wird explizit als Kontext an die nächste Task übergeben (via Task.context in CrewAI).

5. Agenten-spezifische Anpassungen zur Kontextreduktion

5.1. Project Manager (PM) - Gemini 2.5 Pro

- **Strategie für "Full Repo Load" (präzisiert):** Fokussiert sich auf Planungs-, Statusartefakte und spezifische Rechercheergebnisse.
- **Kernaufgabe (verstärkt durch Recherche):** Aktive Kontextkonstruktion durch Aggregation, Filterung, semantische Relevanzbewertung, RAG und Zusammenfassung. Formuliert spezifische Recherchefragen (siehe Abschnitt 9).

5.2. Developer Agent (Claude Sonnet / alternatives LLM) & Debug Agent (Codestral)

- **Input-Kontext:** Erhält primär den plan.md des aktuellen (Sub-)Tasks mit SCoT-Anweisungen und nur die Code-Dateien, die für diesen Task relevant sind.
- **Arbeitsweise:** Implementiert Features modular. Schreibt Code in kleineren Chunks, ggf. durch mehrere write_file_tool Aufrufe mit append-Logik für dieselbe Datei (siehe Abschnitt 6).

5.3. Researcher Agent (Gemini 1.5 Flash)

- **Input-Kontext:** Erhält sehr spezifische Rechercheaufträge vom PM (siehe Abschnitt 9).
- **Output:** Liefert *immer* prägnante Zusammenfassungen, die direkt die gestellten Fragen beantworten (Nutzung text_summarization_tool mit klarem summary_focus).

5.4. Tester Agent (Mistral Medium)

- **Input-Kontext:** Erhält den plan.md des zu testenden Steps/Features und die relevanten Code-Dateien.

6. Tool-Optimierungen und Nutzungshinweise

- **text_summarization_tool (basierend auf Recherche Teil 1, Abschnitt V.D):**
  - **Proaktive und fokussierte Nutzung:** Alle Agenten müssen angewiesen werden, dieses Tool mit einem klaren summary_focus zu verwenden, der auf die Bedürfnisse der aktuellen Aufgabe oder des empfangenden Agenten zugeschnitten ist.
  - **Dynamische Längen-/Detailkontrolle:** Der aufrufende Agent sollte das gewünschte Ausgabeformat/Detaillierungsgrad spezifizieren können.
  - **Systematische Integration:** PM nutzt es zur Kontextvorbereitung; Researcher zur Ergebnisdestillation.
  - **Fortgeschrittene Optionen (optional):** Evaluation einer internen Multi-LLM-Strategie und rekursiver Zusammenfassung für sehr große Inputs innerhalb des Tools.

- **Write File Tool & Co.:**
    - Agenten (insbesondere Developer) anweisen, atomare Änderungen vorzunehmen.
    - **Empfehlung:** Evaluation und ggf. Implementierung eines append_to_file_tool oder einer Modifikation des write_file_tool mit einem append=True Modus, um inkrementelles Schreiben zu erleichtern.
- **Tool-Beschreibungen und Pydantic-Modelle (basierend auf Recherche Teil 1, Abschnitt III.D.2 & III.D.4):**
    - System-Prompt: Tool-Beschreibungen so prägnant wie möglich halten.
    - **Pydantic Field Descriptions als "Mikro-Prompts":** Für alle Custom Tools detaillierte, LLM-leitende description-Strings für jedes Field in den Pydantic Input-Modellen verfassen, um korrekte Tool-Aufrufe zu fördern.
- **Task-Level Tool Scoping (Recherche Teil 1, Abschnitt III.D.1):**
    - Bei der Definition von Task-Objekten in CrewAI sollte jeder Task nur die minimal notwendige Teilmenge von Tools über den tools-Parameter zugewiesen bekommen.
- **Zusammenfassung von Tool-Beobachtungen (Recherche Teil 1, Abschnitt III.D.3):**
    - Agenten anweisen, Ausgaben von informationsintensiven Tools (z.B. scrape_website_content_tool, read_file_tool) systematisch mit dem text_summarization_tool zu verdichten, bevor sie weiterverarbeitet werden.

7. LLM-Auswahl und Rate-Limiting-Strategien

- **Dynamische LLM-Zuweisung (fortgeschritten):** Prüfung, ob für token-intensive Tasks temporär auf LLMs mit höheren Limits gewechselt werden kann.
- **Kleinere Batches für API-Calls:** Interne Optimierung von LiteLLM/LLM-Wrappern.

8. Überarbeitung der Agenten-System-Prompts und Goals

- **Beibehaltung der Backstories:** Die ausführlichen Backstories bleiben erhalten, um den Charakter der Agenten zu wahren.
- **Anweisungen in goal oder Task-Beschreibungen integrieren:** Explizite Anweisungen zur Token-Sparsamkeit, zur Nutzung von Zusammenfassungs-Tools und zum "Need-to-Know"-Prinzip werden in die goal-Definitionen der Agenten oder in die Beschreibungen spezifischer Tasks aufgenommen.

9. Wissensmanagement und Recherchesteuerung

- **Formulierung spezifischer Recherchefragen:**
    - Der PM formuliert detaillierte und kontextualisierte Recherchefragen für den Researcher Agent.
    - **Format-Vorschlag:**
      [Prägnante Beschreibung des Gesamt-Projekts oder des aktuellen Problems]

Frage X: [Konkrete Frage zum Problem Y]
[Background für Frage X: IST-Zustand (wie machen wir es?), SOLL-Zustand (was wollen wir erreichen?), BEGRÜNDUNG/HYPOTHESE (warum glauben wir, dass SOLL besser ist?). Welche Daten/Best Practices/Alternativen gibt es für den Übergang von IST zu SOLL unter Berücksichtigung von Aspekt A und B?]

    - Dies stellt sicher, dass der Researcher Agent hochrelevante und fokussierte Informationen liefert.
- **Knowledge Graphs (Langfristige Vision):**
    - Evaluation der Integration eines Knowledge Graphs als zentrales, lernendes Gedächtnis des Systems.
    - Ziel: Beziehungen zwischen Anforderungen, Entscheidungen, Code-Modulen, Agenten und Tasks explizit machen, um das Situationsbewusstsein und die Konsistenz zu verbessern.
    - Dies ist ein fortgeschrittenes Thema für eine spätere Optimierungsphase (Recherche Teil 2).

10. Nächste Schritte und Prioritäten

1. **Modularisierung der Agenten-Definitionen:** Aufteilung der agents.py in separate Dateien pro Agent.
2. **Anpassung des ProjectWorkflowManager:** Implementierung der Logik zur Erstellung und Weitergabe von aufgabenspezifischen, reduzierten Kontexten (basierend auf Ergebnissen von Recherche Frage 1).
3. **Durchsetzung der files_access.json:** Implementierung eines Proxy/Facade-Musters in file_operations_tool.py (basierend auf Ergebnissen von Recherche Frage 4).
4. **Überarbeitung der Task-Definitionen:** Aufbrechen bestehender und zukünftiger Tasks in kleinere, atomare Einheiten, insbesondere für Code-Generierung unter Nutzung von SCoT-Prinzipien (basierend auf Ergebnissen von Recherche Frage 2).
5. **Optimierung der Tool-Nutzung:** Implementierung von Task-Level Tool Scoping und Verfeinerung der Pydantic-Modelle für Tools (basierend auf Ergebnissen von Recherche Frage 3).
6. **Systematische Integration der Zusammenfassung:** Anpassung der Agenten-Workflows und Prompts zur konsequenten Nutzung des text_summarization_tool mit summary_focus (basierend auf Ergebnissen von Recherche Frage 5).
7. **Anpassung der Agenten-Goals/Task-Beschreibungen:** Integration von Anweisungen zur Token-Effizienz.
8. **Durchführung der Recherche Teil 2 (Knowledge Graphs).**
9. **Iterative Implementierung der Optimierungen** basierend auf den Rechercheergebnissen und Tests der modularisierten Agenten.

Durch die konsequente Umsetzung dieser Maßnahmen sollte das System deutlich effizienter im Umgang mit Tokens werden, die Fehleranfälligkeit reduzieren und die Performance,

insbesondere bei LLMs mit strengeren Rate Limits, verbessern. Die vorgeschlagenen Tabellen aus dem Recherchedokument (z.B. zu Kontextmanagement-Techniken, Aufgabenzerlegung, Tool-Interaktion, Dateizugriffsmuster, Prompting für Zusammenfassungen) sollten als interne Referenzen und Leitfäden für die Implementierung dieser Optimierungen dienen.