**✍️ AI Agent Assignment Submission**

**Candidate:** Abhigyan Singh Jagwan  
**Use Case Title:** *AI Agent to Extract Important Questions from Previous Year Exam Papers*

**🧾 SECTION 1: BASIC DETAILS**

* **Name:** Abhigyan Singh Jagwan
* **AI Agent Title / Use Case:** AI Agent to Extract Important Questions from Previous Year Exam Papers

**🧠 SECTION 2: PROBLEM FRAMING**

**2.1 What problem does your AI Agent solve?**

Students often have multiple years of exam papers but don’t know which questions are important or repeated. This agent reads uploaded papers, extracts all questions, and highlights the ones that appear frequently across different years.

**2.2 Why is this agent useful?**

It saves time and effort by helping students focus only on high-priority questions that are more likely to be repeated in upcoming exams. It reduces exam stress and improves revision efficiency.

**2.3 Who is the target user?**

College students preparing for semester exams, especially those who want to revise smartly using past year trends.

**2.4 What not to include?**

* No answer key generation for questions
* No explanation of answers
* No prediction using machine learning — only pattern-based analysis
* No live web scraping — user must upload the papers manually

**🧱 SECTION 3: 4-LAYER PROMPT DESIGN**

**🔹 3.1 INPUT UNDERSTANDING**

* **Prompt:**  
  “The user uploads previous year papers as text files.  
  Extract all question-like sentences (e.g., those ending in ? or starting with keywords like ‘define’, ‘explain’, ‘what’, etc.).  
  Return a clean list of questions found in the text.”
* **Responsible for:**  
  Parsing raw exam paper content and extracting valid question patterns.
* **Example Input:**
* 1. Define normalization in DBMS.
* 2. What is 2-phase locking?
* 3. Explain concurrency control.
* **Example Output:**
  + "Define normalization in DBMS."
  + "What is 2-phase locking?"
  + "Explain concurrency control."

**🔹 3.2 STATE TRACKER**

* **Prompt:**  
  “Now keep track of all extracted questions across multiple years.  
  Store them in memory along with how often they occurred and the year(s) they appeared.”
* **How does this help “remember”?**  
  The agent persists information across file uploads and tracks frequency + source years for each question.
* **Simulated Memory Used:**  
  Yes — dictionary object simulates memory with structure:  
  {question: {years: [...], frequency: n}}

**🔹 3.3 TASK PLANNER**

* **Prompt:**  
  “From the tracked questions:
  + Filter those that appear in 2 or more years
  + Sort by frequency (descending)
  + Optionally apply filters (e.g., theory-only)
  + Return top N repeated questions.”
* **Steps taken:**
  + Filter questions
  + Sort by frequency
  + Apply optional filters
  + Slice top-N
* **Complexity Management:**  
  Chaining is used. Each step is modular and sequential for clarity.

**🔹 3.4 OUTPUT GENERATOR**

* **Prompt:**  
  “Display the final results in student-friendly Markdown.  
  Use numbering, include appearance years in italics.  
  If no questions are repeated, show a fallback message.”
* **Output Goals:**
  + Numbered list
  + Markdown formatting
  + Friendly tone
  + Supports fallback when no result is found

**🧪 SECTION 4: CHATGPT EXPLORATION LOG**

| **Attempt #** | **Prompt Variant / Action** | **What Happened** | **What You Changed** | **Why You Changed It** |
| --- | --- | --- | --- | --- |
| 1 | Extract only with ? | Missed instructions like “Define” or “Explain” | Included instruction keywords | Captured more valid academic questions |
| 2 | Stored all questions + tracked frequency | Crashed on new keys | Added safe initialization logic | Prevented key errors and improved memory |
| 3 | Showed question + year list | Duplicated years appeared | Deduplicated year list in output | Cleaner and clearer presentation for students |

**🧪 SECTION 5: OUTPUT TESTS**

**✅ Test 1: Normal Input**

* **Input:** 3 past year papers (2021–2023) with standard questions
* **Expected Output:**
* 1. Define normalization in DBMS. \_[2021, 2022, 2023]\_
* 2. Explain 2-phase locking protocol. \_[2021, 2023]\_
* 3. Describe concurrency control. \_[2022, 2023]\_

**✅ Test 2: Vague Input**

* **Input:** One small paper with few lines, not many questions
* **Output:**  
  ⚠️ No repeated questions found. Upload more papers for better analysis.

**✅ Test 3: Invalid Input**

* **Input:** File with no questions, only instructions
* **Output:**  
  ⚠️ Could not detect valid questions. Please upload proper exam paper text.

**🧠 SECTION 6: REFLECTION**

**6.1 Hardest Part**

Breaking the problem into four modular prompt layers — understanding vs. memory vs. logic vs. output — was tricky. Simulating memory without code was also a creative challenge.

**6.2 Most Enjoyable Part**

Designing logic to detect repeated questions across years and formatting it cleanly for users was very rewarding.

**6.3 If I Had More Time**

I’d add NLP similarity detection (to catch rephrased questions), and a front-end UI for uploads + results.

**6.4 What I Learned**

Prompt design is like modular programming. Giving ChatGPT clear roles for each layer helps create better behavior.

**6.5 When I Felt Stuck**

Initially mapping system design to prompt layers was unclear. I tackled it by breaking down the process, iterating on prompts, and asking ChatGPT better follow-up questions.

**🧠 SECTION 7: HACK VALUE (Optional)**

* ✅ **Went Beyond Brief:**  
  Designed each layer as an independent modular agent using Python logic.
* ✅ **Edge Cases Covered:**  
  Tested for empty files, low-data uploads, and duplicate handling.
* ✅ **Chaining + Fallbacks Used:**  
  Fallback message for empty results. Sorting + filtering steps chained clearly.
* ✅ **Explainability:**  
  Output shows **which years** each question appeared, making the reasoning traceable.