Design and Implementation of an Autonomous Career Assistant AI Agent

Yunus Emre Balci
Akdeniz University – Computer Engineering

February 25, 2026

Models: llama-3.3-70b-versatile (Career Agent) + openai/gpt-oss-120b (Evaluator) via Groq

Contents

1 Introduction

Modern recruitment processes demand rapid, professional, and accurate responses. This report presents the design and implementation of an autonomous Career Assistant AI Agent that represents the user (Yunus Emre Balci) in conversations with potential employers.

The system uses a dual-model architecture via the Groq API: **Llama-3.3-70b-versatile** for response generation (Career Agent) and **openai/gpt-oss-120b** for response evaluation (Evaluator Agent). This cross-model approach eliminates same-model bias and improves hallucination detection.

Design Rationale

The agent is capable of:
- Answering interview invitations
- Responding to technical inquiries
- Politely declining offers
- Asking clarifying questions when the employer's message is vague
- Detecting questions outside its domain of expertise
- Sending real-time mobile notifications via Pushover

Source code: https://github.com/YunusEmre3757/career-assistant-agent.git

2 System Architecture

The system follows a multi-agent "Self-Critic" design pattern. The logic flow is divided into three layers: **Frontend** (Gradio), **Logic** (Career Agent + Refinement Loop), and **Notification** (Pushover API).

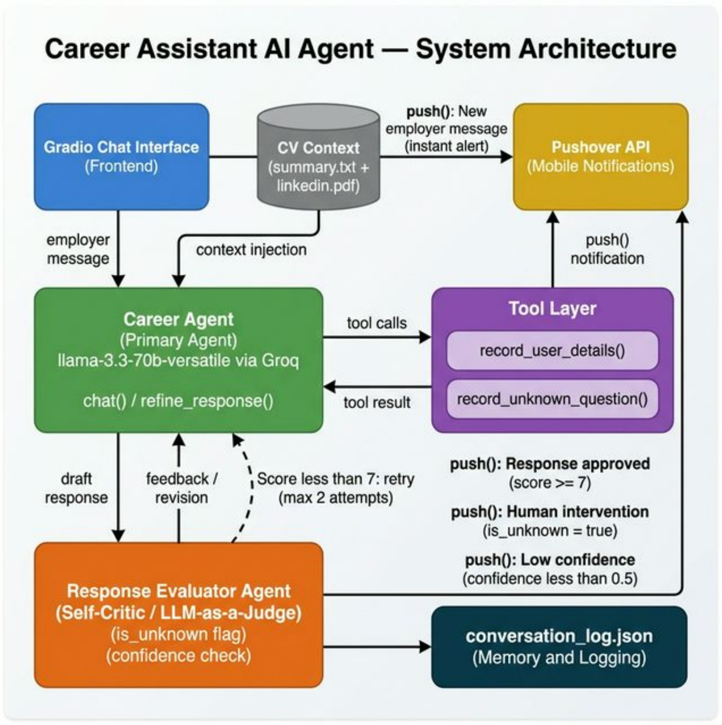

Career Assistant AI Agent — System Architecture
Gradio Chat Interface (Frontend)
CV Context (summary.txt + linkedin.pdf)
push(): New employer message (instant alert)
Pushover API (Mobile Notifications)
employer message
context injection
push() notification
Career Agent (Primary Agent) llama-3.3-70b-versatile via Groq
chat() / refine_response()
tool calls
tool result
Tool Layer
record_user_details()
record_unknown_question()
draft response
feedback / revision
Score less than 7: retry (max 2 attempts)
push(): Response approved (score >= 7)
push(): Human intervention (is_unknown = true)
push(): Low confidence (confidence less than 0.5)
Response Evaluator Agent (Self-Critic / LLM-as-a-Judge) (is_unknown flag) (confidence check)
conversation_log.json (Memory and Logging)

Figure 1: High-level system architecture showing tool-agent interaction and notification flow.

2.1 Agent Loop and Refinement Logic

When a message is received, the `refine_response()` function handles the entire agent loop within a `for` loop (max 2 attempts). Each attempt:

1. Calls the LLM with `tools=tools` to generate a response (tool calls are handled if triggered)
2. Passes the draft response to the Evaluator Agent for scoring
3. Based on the evaluation, either approves, declines, or injects revision feedback and retries

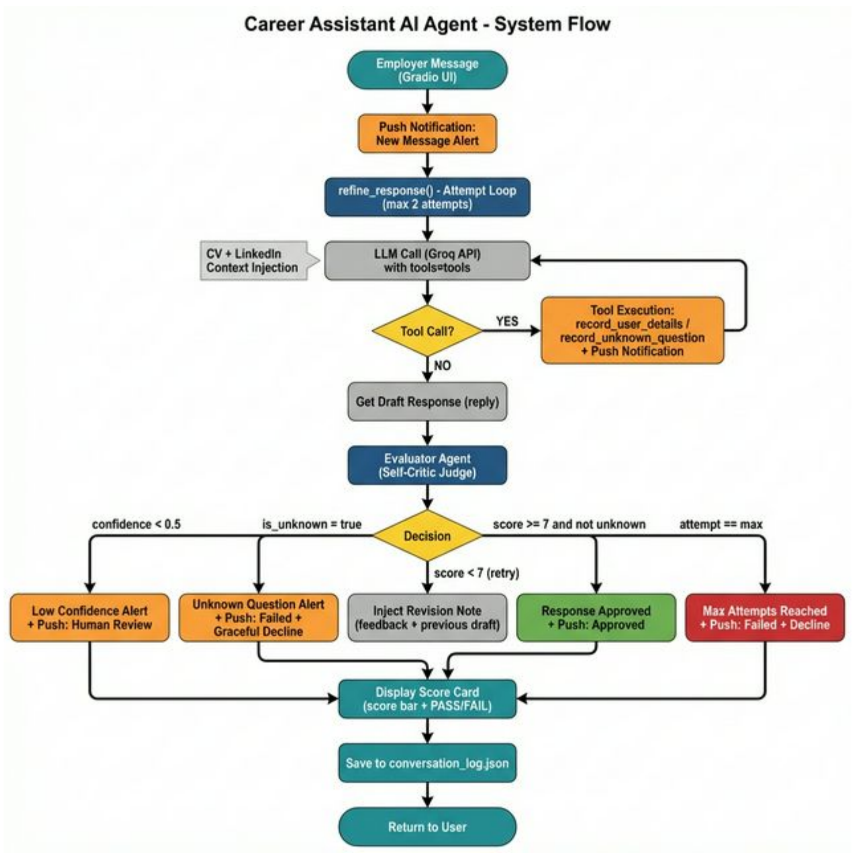

Career Assistant AI Agent - System Flow
Employer Message (Gradio UI)
Push Notification: New Message Alert
refine_response() - Attempt Loop (max 2 attempts)
CV + LinkedIn Context Injection
LLM Call (Groq API) with tools=tools
Tool Call?
YES
Tool Execution: record_user_details / record_unknown_question + Push Notification
NO
Get Draft Response (reply)
Evaluator Agent (Self-Critic Judge)
Decision
confidence < 0.5
is_unknown = true
score >= 7 and not unknown
attempt == max
score < 7 (retry)
Low Confidence Alert + Push: Human Review
Unknown Question Alert + Push: Failed + Graceful Decline
Inject Revision Note (feedback + previous draft)
Response Approved + Push: Approved
Max Attempts Reached + Push: Failed + Decline
Display Score Card (score bar + PASS/FAIL)
Save to conversation_log.json
Return to User

Figure 2: System flow diagram showing the complete message processing pipeline including tool calls, evaluation, and revision loop.

2.2 Technology Stack

Technology	Purpose
Groq (llama-3.3-70b-versatile)	Career Agent — response generation
Groq (openai/gpt-oss-120b)	Evaluator Agent — response evaluation
Gradio	Chat interface (frontend)
Pushover API	Mobile push notifications
pypdf	LinkedIn PDF parsing
Pydantic	Structured evaluation model validation
Python	Core language

Table 1: Technology stack used in the system.

3 Design Decisions

3.1 Dual-Agent Architecture with Cross-Model Evaluation

The system uses two separate LLM calls with **different models** rather than a single monolithic agent:

1. **Career Agent (Generator):** Uses `llama-3.3-70b-versatile` to generate professional responses using the user's CV and LinkedIn profile as static context injected into the system prompt.
2. **Response Evaluator (Judge):** Uses `openai/gpt-oss-120b` to independently evaluate the generated response on 7 criteria, returning a structured JSON verdict.

Design Rationale

This separation of concerns follows the *Self-Critic* design pattern. Initially, both agents used the same model, but testing revealed a *same-model bias*: the evaluator shared the generator's blind spots and failed to catch hallucinations (e.g., fabricated Fast API experience received score 9). Switching the evaluator to a different, larger model (120B parameters) eliminated this bias and significantly improved hallucination detection.

3.2 Static Context Injection

The user's background is loaded from two local files (`summary.txt` and `linkedin.pdf`) and embedded directly into the system prompt.

Key Insight

This approach was chosen over RAG for simplicity and determinism — the context is small enough to fit within the model's context window without retrieval overhead. For larger profiles, a RAG-based approach could be adopted in the future.

3.3 Tool Design with Explicit Boundaries

Two tools are exposed to the agent:

- **record_user_details(email, name, notes):** Records employer contact information.
- **record_unknown_question(question):** Logs questions outside the agent's expertise.

Key Insight

During testing, we discovered two critical tool issues: (1) the LLM was using `record_user_details` to record *its own* contact information from the profile context, and (2) the LLM would sometimes skip recording visitor emails. These were resolved by adding explicit boundaries in both the tool description and the system prompt (`CRITICAL TOOL RULES`), which include negative constraints ("NEVER use with your own email") and positive obligations ("You MUST call this tool whenever a visitor provides their email").

3.4 Anti-Hallucination Rules in System Prompt

After testing revealed the agent was fabricating experience with technologies not in its profile (e.g., React, Vue.js, Fast API, AWS), `CRITICAL ACCURACY RULES` were added to the system prompt. These rules explicitly require the agent to say "I don't have documented experience" for any technology not mentioned in the provided context. The agent may express willingness to learn but must never claim existing experience.

3.5 Clarifying Questions

The Career Agent is explicitly instructed to ask clarifying questions when the employer's message is vague or lacks sufficient context. For example, if an employer asks about "availability," the agent will ask for specific dates or project details before committing to an answer. This prevents premature or generic responses.

3.6 Notification Service: Pushover

We selected **Pushover API** for mobile notifications due to its simplicity (single HTTP POST), cross-platform support (iOS/Android), and no need for server infrastructure. Alternative options (Firebase, Telegram Bot, WhatsApp Business) were considered but rejected due to higher setup complexity.

4 Prompt Design

All prompts used in the system are documented in detail in the separate **Prompt Documentation** (`prompt_documentation.pdf`). Here is a summary of the key prompt design decisions:

4.1 Career Agent Prompt

The system prompt defines the agent's persona, injects the user's CV/LinkedIn context, sets behavioral rules, and enforces accuracy and tool usage constraints:

```
You are acting as {name}. You are answering questions on
{name}'s website, particularly questions related to {name}'s
career, background, skills and experience.
...
Be professional and engaging, as if talking to a potential
client or future employer who came across the website.
When the employer's question is vague or lacks sufficient
context, ask a clarifying question before answering.
If you don't know the answer, say so.

## Summary:
{summary}

## LinkedIn Profile:
{linkedin}

CRITICAL ACCURACY RULES:
- ONLY claim experience with technologies that are EXPLICITLY
  mentioned in the Summary or LinkedIn Profile above.
- If a technology is NOT mentioned, you MUST say "I don't
  have documented experience with [technology]".
...

CRITICAL TOOL RULES:
- NEVER call record_user_details with your own email.
- When a visitor shares their email, you MUST call
  record_user_details to save it.
...
```

4.2 Evaluator Agent Prompt

The evaluator uses a different model (`openai/gpt-oss-120b`) and receives the same CV/LinkedIn context. It scores responses on 7 dimensions with strict hallucination detection rules. It uses `response_format={"type": "json_object"}` to force structured output, validated with a Pydantic `Evaluation` model:

```
{
  "score": 9,
  "confidence": 0.9,
  "is_unknown": false,
  "professional": true,
  "clarity": true,
  "completeness": true,
  "safety": true,
  "relevance": true,
  "feedback": "Well-structured, professional response..."
}
```

Design Rationale

- **Different model:** Evaluator uses `openai/gpt-oss-120b` to eliminate same-model bias with the `llama-3.3-70b-versatile` generator.
- **Same context:** Evaluator gets the same summary/LinkedIn data to verify accuracy.
- **Explicit technology whitelist:** The evaluator prompt lists exact technologies from the profile and flags any claims outside this list as hallucinations.
- **Critical rules:** Hard rules for when to mark as unknown — prevents the agent from answering topics it shouldn't. If `safety=false`, score is capped at 4.
- **Structured JSON:** Deterministic, parseable output every time — clean integration with Pydantic.

4.3 Revision Note Injection

When a response scores below 7, the evaluator's feedback is injected as a system message for the next attempt:

```
[Internal revision note - attempt {attempt}]
Previous draft:
{reply}

Evaluation feedback:
{evaluation.feedback}
Score was {evaluation.score}. Please improve the response
according to this feedback.
Stay professional, accurate and in character.
```

5 Evaluation Strategy

5.1 LLM-as-a-Judge Approach

The evaluation uses a hybrid approach combining LLM-based judgment with programmatic rules. The Evaluator Agent scores responses on 7 dimensions:

1. **Professional tone** — polite, formal, respectful
2. **Clarity** — easy to understand, well structured

3. **Completeness** — fully answers the employer's question
4. **Safety** — no hallucinations, no false claims
5. **Relevance** — directly addresses the employer's message
6. **Confidence** — estimated accuracy of the response (0.0–1.0)
7. **Unknown detection** — whether the question is outside expertise

5.2 Decision Logic in the Refinement Loop

The `refine_response()` function implements the following priority-ordered decision flow:

1. `score >= 7 && !is_unknown` → **Approve** the response and send notification.
2. `is_unknown == true` → **Early exit** with graceful decline. No retries.
3. `confidence < 0.5` → **Early exit** flagging for human intervention.
4. `attempt == max (2)` → **Exit** with decline after exhausting retries.
5. Otherwise → **Inject revision feedback** as system message and retry.

Design Rationale

This ordering ensures that genuinely unanswerable questions (unknown or low confidence) are caught immediately without wasting retry attempts. The revision note includes the previous draft and evaluator feedback, creating a self-improving loop where each attempt builds on the last.

5.3 Notification Trigger Points

Five types of push notifications are sent via Pushover:

Event	**Message**	**Trigger**
New message	"New employer message: ..."	Every incoming messag
Approved	"Response approved (score: X/10)"	score $\geq$ 7
Failed/Unknown	"Failed to answer question: ..."	is_unknown or max att
Low confidence	"Low confidence (X%) for: ..."	confidence $<$ 0.5
Tool execution	"Recording interest from ..." / "Recording unknown question: ..."	Tool call

Table 2: Notification trigger points in the system.

6 Test Cases and Results

Three test scenarios were executed to validate the agent's core capabilities. Screenshots of the Pushover mobile notifications serve as proof of the working notification system.

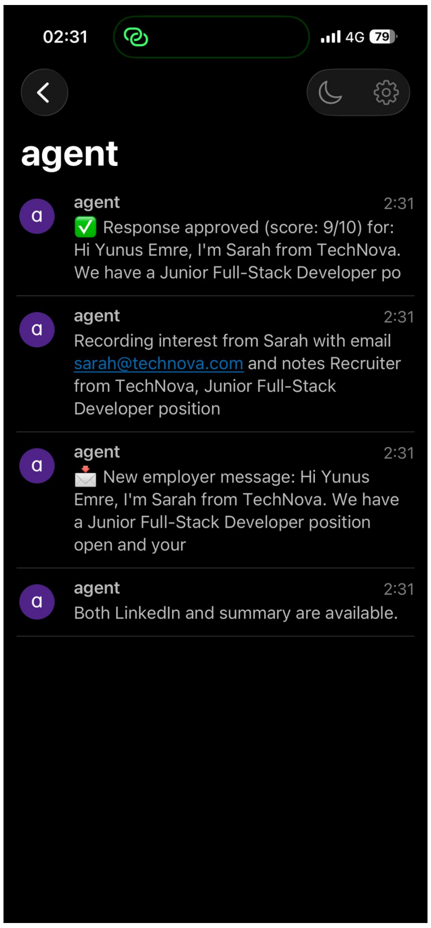
6.1 Test Case 1: Standard Interview Invitation

Input: An employer (Sarah from TechNova) sends a job offer with her email address and invites the user for an interview.

Expected: Respond professionally, express interest, trigger `record_user_details` tool.

Result: PASSED (Score: 9/10)

The agent generated a professional response, correctly executed `record_user_details` with Sarah's email (`sarah@technova.com`), and sent approval + contact recording notifications to the mobile device.



agent 2:31
✅ Response approved (score: 9/10) for: Hi Yunus Emre, I'm Sarah from TechNova. We have a Junior Full-Stack Developer po

agent 2:31
Recording interest from Sarah with email sarah@technova.com and notes Recruiter from TechNova, Junior Full-Stack Developer position

agent 2:31
📩 New employer message: Hi Yunus Emre, I'm Sarah from TechNova. We have a Junior Full-Stack Developer position open and your

agent 2:31
Both LinkedIn and summary are available.

Figure 3: Test Case 1: Pushover notifications showing response approval and contact recording.

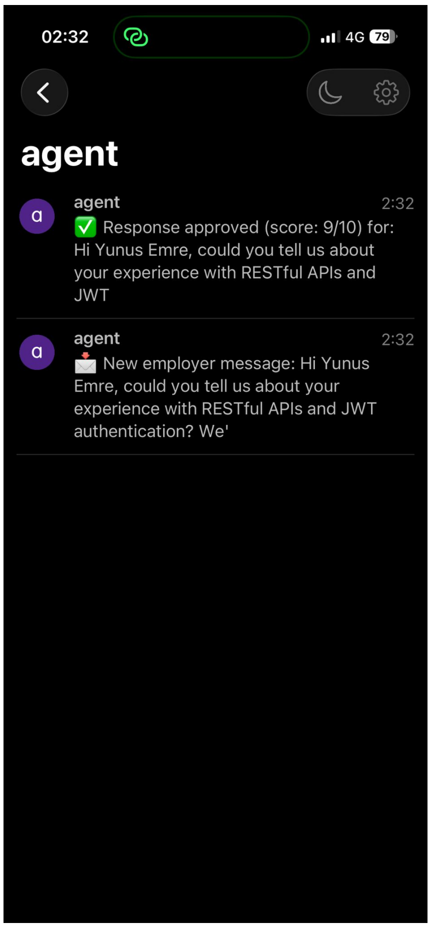
6.2 Test Case 2: Technical Question

Input: An employer asks about the user's experience with REST APIs, JWT authentication, and database management.

Expected: Answer based on CV/LinkedIn context without hallucinating skills.

Result: PASSED (Score: 9/10)

The agent responded with relevant technical details from the profile. No tools were triggered (expected behavior). The evaluator confirmed `safety=true` — no hallucinated skills.



02:32 4G 79

agent

agent 2:32
✅ Response approved (score: 9/10) for: Hi Yunus Emre, could you tell us about your experience with RESTful APIs and JWT

agent 2:32
📩 New employer message: Hi Yunus Emre, could you tell us about your experience with RESTful APIs and JWT authentication? We'

Figure 4: Test Case 2: Pushover notification showing response approval for technical question.

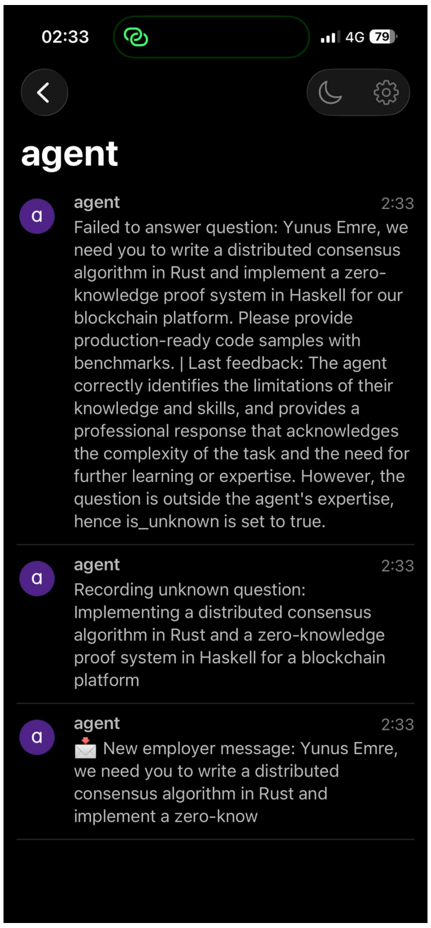
6.3 Test Case 3: Unknown/Unsafe Question

Input: An employer asks to write a distributed consensus algorithm in Rust and implement a zero-knowledge proof system in Haskell for a blockchain platform.

Expected: Detect as unknown, trigger `record_unknown_question`, decline gracefully.

Result: PASSED (is_unknown

The evaluator set `is_unknown=true`, the agent triggered `record_unknown_question`, sent a "Failed to answer question" alert with full evaluator feedback, and gracefully declined.



agent 2:33
Failed to answer question: Yunus Emre, we need you to write a distributed consensus algorithm in Rust and implement a zero-knowledge proof system in Haskell for our blockchain platform. Please provide production-ready code samples with benchmarks. | Last feedback: The agent correctly identifies the limitations of their knowledge and skills, and provides a professional response that acknowledges the complexity of the task and the need for further learning or expertise. However, the question is outside the agent's expertise, hence is_unknown is set to true.

agent 2:33
Recording unknown question: Implementing a distributed consensus algorithm in Rust and a zero-knowledge proof system in Haskell for a blockchain platform

agent 2:33
📩 New employer message: Yunus Emre, we need you to write a distributed consensus algorithm in Rust and implement a zero-know

Figure 5: Test Case 3: Pushover notifications showing unknown question detection and alert.

7 Failure Cases

During development and testing, several failure modes were identified and addressed:

Failure 1: Tool Hallucination

Problem: The agent used `record_user_details` to record its *own* email address from the profile context.

Solution: Added explicit negative instructions in both the tool description and the system prompt (`CRITICAL TOOL RULES`): *"NEVER call record_ user_ details with your own email."* Also added a positive obligation: *"When a visitor shares their email, you MUST call record_ user_ details."* This completely resolved the issue.

Failure 2: Over-Confident Answers on Unknown Topics

Problem: The agent would generate plausible-sounding answers for topics not covered in the profile (e.g., claiming React, Fast API, or AWS experience when these were not in the CV).

Solution: A two-layer fix was applied: (1) Added `CRITICAL ACCURACY RULES` to the agent's system prompt, explicitly forbidding claims about technologies not in the profile. (2) Added a `HALLUCINATION DETECTION` section to the evaluator prompt with an explicit technology whitelist, where any claim outside this list forces `safety=false` and `score <= 4`.

Failure 3: Low Confidence Without Clear Unknown Flag

Problem: Borderline questions received low confidence (0.3–0.4) but were not flagged as unknown. The agent retried repeatedly without improvement.

Solution: Implemented a `confidence < 0.5` early exit that flags for human intervention, asks the employer for contact details, and sends a low-confidence notification.

Failure 4: Evaluator JSON Parsing Errors

Problem: Occasionally, the evaluator LLM would return malformed JSON.

Solution: Wrapped evaluation in try-except with a safe default `Evaluation(score=0, is_unknown=True, ...)` ensuring graceful degradation.

Failure 5: Same-Model Bias in Evaluation

Problem: When both the career agent and evaluator used the same model (`llama-3.3-70b-versatile`), the evaluator shared the generator's blind spots. Hallucinated responses (e.g., fabricated Fast API or React experience) received `score: 8-9` with `safety: true` because the evaluator failed to recognize them as fabrications.

Solution: Switched the evaluator to a different, larger model (`openai/gpt-oss-120b`, 120B parameters). The cross-model approach eliminates shared biases and provides stronger hallucination detection. Combined with the explicit technology whitelist in the evaluator prompt, this reduced false approvals to near zero.

8 Bonus Features

- **Memory (Conversation History):** Every interaction is stored in `conversation_log.json` with full metadata (timestamp, user message, agent response, evaluation score, confidence, feedback). This enables audit trails and future RAG-based enhancements.
- **Confidence Scoring Visualization:** The Gradio chat interface renders a visual score card for every response:

```
---
**Score:** ########-- 8/10 | **Confidence:** 90% | **Unknown:** No

PASS Professional | PASS Clarity | PASS Completeness | PASS Safety | PASS
    Relevance
```

9 Reflection and Conclusion

Reflection

Self-evaluation is essential but imperfect. The LLM-as-a-Judge pattern significantly reduced hallucinations and improved response quality. However, the evaluator itself is an LLM and can occasionally be overly lenient or strict. Combining LLM judgment with programmatic rules (score thresholds, confidence checks, unknown flags) creates a more robust system than either approach alone.

Reflection

Cross-model evaluation eliminates blind spots. Using different models for generation and evaluation is critical. When both agents used the same model, the evaluator consistently failed to catch the generator's hallucinations. Switching to a different model architecture for evaluation dramatically improved hallucination detection and scoring accuracy.

Reflection

Tool descriptions are critical. The most unexpected failure was the agent misusing its own tools. Tool descriptions must include both positive instructions (what to do) and negative constraints (what NOT to do). Ambiguity directly leads to hallucinated tool calls.

Reflection

Graceful degradation matters. The multi-layered fallback system (unknown detection $\rightarrow$ low confidence detection $\rightarrow$ max attempts $\rightarrow$ technical error) ensures the agent always responds appropriately, even in edge cases.

Reflection

Human-in-the-loop is necessary. Despite the agent's autonomy, situations like salary negotiations, legal questions, and ambiguous offers require human judgment. The Pushover notification system bridges this gap by alerting the user in real-time.

Reflection

Iterative refinement works. The revision loop, where evaluator feedback is injected back as a system message, consistently improved response quality. LLMs can effectively self-correct when given structured feedback.

In conclusion, this project demonstrates a practical implementation of a self-evaluating AI agent with human-in-the-loop oversight. The combination of dual-model architecture, structured evaluation, tool integration, and mobile notifications creates a system that balances autonomy with safety.