

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Absolutely — the image you've shared is from a talk by Jerry Liu (LangChain co-founder) on how to go from *simple* to *production-grade* RAG systems. It presents a powerful gradient of capabilities across Table Stakes → Advanced Retrieval → Fine-tuning → Agentic Behavior.**

**Let’s do a deep dive into the two most powerful areas:**

**🔍 ADVANCED RETRIEVAL**

***Goal: Go beyond naive “retrieve top 3 chunks” logic and make retrieval smarter, more contextual, and more structured.***

**1. Re-ranking**

**Use a smarter model (e.g. BERT or a cross-encoder) to re-order retrieved documents based on the query.**

* **🔹 Why: Vector DB similarity isn’t always great at nuance.**
* **🔹 How: For top 20 results from FAISS, score them again using something like** [**Cohere ReRank**](https://docs.cohere.com/docs/rerank) **or OpenAI function calling.**

**Code example:**

**ranked\_docs = cohere\_rerank(query, retrieved\_docs)**

**top\_3 = ranked\_docs[:3]**

**Best for: Getting rid of junk results in medical, legal, or finance domains where precision matters.**

**2. Recursive Retrieval**

**If the first answer isn’t enough, generate follow-up queries or decompose the question.**

* **🔹 Why: Some questions are *composite* (e.g. "What’s the maternity leave policy and how does it compare to government guidelines?")**
* **🔹 How: Use an LLM to break a question into parts, retrieve for each, then aggregate.**

**Think:**

**queries = decompose(query)**

**docs = [retrieve(q) for q in queries]**

**3. Embedded Tables / Structured Indexes**

**Instead of treating all text the same, extract and separately index structured parts of documents (tables, lists, metadata).**

* **🔹 Why: Tables are dense with info but hard for chunking.**
* **🔹 How: Use PDF parsing tools like** [**Unstructured.io**](https://unstructured-io.github.io/) **+ structure-aware embeddings.**

**Example:  
Index company benefits table with a different chunker → tag with column\_name: maternity\_leave.**

**4. Small-to-Big Retrieval (Hierarchical)**

**Start with short, focused chunks, then expand outward if needed.**

* **🔹 Why: Reduces latency and irrelevant noise.**
* **🔹 How: Store small FAQ snippets separately; if query doesn't match them, then fallback to larger paragraphs or full docs.**

**Pattern:**

**if match\_in\_faq(query):**

**return top\_faq**

**else:**

**return search\_full\_docs(query)**

**🤖 AGENTIC BEHAVIOR**

***Goal: Let the system “think” about how to solve the query — like a mini assistant planning its own retrieval and response strategy.***

**1. Routing**

**Use different retrievers or models based on query type or user profile.**

* **🔹 Why: Not all queries are created equal. A benefits question ≠ a legal compliance question.**
* **🔹 How: Use a classifier to route queries to different retrievers (e.g. BM25 for dates, dense for intent, GraphRetriever for relationships).**

**if "compare" in query:**

**use\_legal\_comparison\_agent()**

**else:**

**use\_faq\_bot()**

**2. Query Planning (Tool-Using Agents)**

**Allow the system to *break down a question*, use tools or retrieval steps to collect evidence, and then generate a final answer.**

* **🔹 Why: Some queries require reasoning over multiple steps.**
* **🔹 How: Use a ReAct agent or LangGraph flow to:**
  1. **Retrieve**
  2. **Plan**
  3. **Call APIs or fetch more data**
  4. **Synthesize answer**

**Example Tools Used:**

* **Retriever**
* **Table summarizer**
* **Clause comparator**

**3. Multi-Document Agents**

**Let an LLM interleave reading and writing across many sources.**

* **🔹 Why: Answers may span multiple policies, laws, or manuals.**
* **🔹 How: Design a memory-aware agent (LangGraph, AutoGen, or CrewAI) that:**
  + **Tracks what it's seen**
  + **Cites sources**
  + **Cross-references documents**

**Think:  
Agent behaves like a research assistant with memory + access to a document corpus.**

**🧠 Summary Table**

| **Feature** | **Retrieval Layer** | **Why It Matters** |
| --- | --- | --- |
| **Re-ranking** | **Advanced** | **Improves relevance without retraining embeddings** |
| **Recursive Retrieval** | **Advanced** | **Handles complex, multi-hop queries** |
| **Embedded Tables** | **Advanced** | **Unlocks dense, structured knowledge** |
| **Routing** | **Agentic** | **Chooses best retrieval strategy** |
| **Query Planning** | **Agentic** | **Breaks down, sequences, and solves complex queries** |
| **Multi-doc Agent** | **Agentic** | **Synthesizes info across many documents with context and memory** |

**🛠 Suggested Side Project**

**🧠 “Policy Pulse Pro” (v2 of your app)  
Implement a multi-agent RAG system that:**

1. **Routes queries to different agents: HR policy, legal comparator, financial benefits**
2. **Each agent does:**
   * **Advanced retrieval (with re-ranking + table parsing)**
   * **Recursive clarification if needed**
3. **Final answer is traceable with cited steps and fallback logic.**

**Want me to help you scaffold this project using LangGraph or LangChain with Cohere or OpenAI as the reranker? Or do a walkthrough of how to integrate these in your current app setup?**

**You're asking all the right questions — these distinctions are subtle but crucial to designing RAG systems that scale in power, responsiveness, and trust.**

**Let's take your three points one by one and go deep in plain terms, grounding each in what's possible with naive RAG vs what requires an agentic structure.**

**🔁 1. Recursive Retrieval: Does It Require Agentic Structure?**

**Short answer: yes — recursive retrieval *can’t* really happen in naive RAG because it requires the system to “notice insufficiency” and “think again”.**

**✅ What naive RAG does:**

* **User asks a question →**
* **Embed it → Retrieve top-N chunks →**
* **Feed chunks + question to LLM →**
* **Return answer**

**💡 It’s stateless, single-shot, and LLM is downstream of retrieval.**

**🧠 What recursive retrieval does:**

**Step-by-step:**

1. **User asks: “What are the maternity benefits and how do they compare to statutory minimums?”**
2. **LLM (with no helpful context) recognizes it's a compound query.**
3. **It rewrites the query as:**
   * **"What are this company’s maternity benefits?"**
   * **"What are UK statutory maternity entitlements?"**
   * **"How do the two compare?"**
4. **Each sub-question is sent through the retriever independently.**
5. **Retrieved answers are then passed back to the LLM for synthesis.**

**✔️ This requires orchestration, not just LLM-as-output-machine.**

**🔧 Implementation options:**

* **In LangChain: use MultiQueryRetriever, ParentDocumentRetriever, or an AgentExecutor with tools.**
* **In Haystack: use Decomposition Nodes + PromptNode.**
* **Or roll your own: llm → sub-query planner → retrieval router → aggregator.**

**So yes — recursive retrieval needs a feedback loop, which *only exists in agentic setups or enhanced RAG chains*.**

**📏 2. Small-to-Big Retrieval: Hierarchical Indexing**

**You nailed the core idea — it’s about strategic layering of your retrieval indices.**

**📚 Concept:**

**You create two (or more) indices:**

| **Index** | **Content** | **Purpose** |
| --- | --- | --- |
| **faq\_index** | **Tiny, focused snippets (FAQs, headlines, definitions)** | **Low-latency, quick wins** |
| **main\_index** | **Larger, semantically chunked paragraphs** | **Richer, slower fallback** |
| **Optional: long\_context\_index** | **Whole docs or high-level summaries** | **For deep dives** |

**🧠 Retrieval strategy:**

**Scenario A: Fast happy path**

* **Query matches confidently in faq\_index**
* **Use top-1 result**
* **Fast and simple**

**Scenario B: Low match confidence or bad user feedback**

* **Retry query in main\_index or long\_context\_index**
* **Could even compare answers from both indices and choose best**
* **May route via agent or fallback logic**

**🔧 Implementation options:**

* **In LlamaIndex: use RetrieverRouter or AutoRetrieverSelector**
* **In LangChain: define multiple RetrieverTools and add logic or agent routing**
* **Manual: score with cosine similarity threshold before escalating to bigger chunks**

**So yes — you may need multiple indices (or at least multiple chunking strategies) to do this properly.**

**🤖 3. Multi-Document Agent vs Multi-Agent System**

**Great question. This is a terminology grey zone, but here's how they differ *conceptually*:**

**🤹 Multi-Document Agent**

**A single agent that works across multiple documents/sources, using memory, planning, and iterative reasoning.**

* **Retrieves from *many documents***
* **May maintain a “working memory” of what’s been read**
* **May re-query as new information is needed**
* **Synthesizes a final response *internally***

**✅ Focus is on breadth of input, depth of reasoning, and continuity of thought**

**👥 Multi-Agent System**

**Multiple specialized agents, each with their own role, tools, or domain knowledge, collaborating to answer a question.**

* **FAQ Agent → handles simple queries**
* **Policy Agent → scans HR manuals**
* **Comparison Agent → analyzes differences**
* **Root Agent → delegates + aggregates**

**✅ Focus is on division of labour and delegation logic**

**In your case (PolicyPulse):**

**If your app has a retrieval agent and a report generator agent, that’s already *multi-agent*.**

**If each agent handles multiple documents with memory of what’s been seen and selectively retrieves more, then you’re also using multi-document capabilities inside agents.**

**So it’s not either/or — you can have multi-agent systems where each agent is also a multi-document agent.**

**🔚 TL;DR**

| **Concept** | **Naive RAG?** | **Agentic Needed?** | **Key Mechanism** |
| --- | --- | --- | --- |
| **Recursive Retrieval** | **❌ Not feasible** | **✅ Yes** | **Sub-query planning, LLM reruns** |
| **Small-to-Big Retrieval** | **✅ Possible with config** | **✅ Better with control flow** | **Multi-index routing** |
| **Multi-Doc Agent** | **❌ Not naive RAG** | **✅ Yes** | **Memory + iterative read-summarize** |
| **Multi-Agent System** | **❌ Not naive RAG** | **✅ Yes** | **Specialized agents + orchestration** |

**Would you like help designing a LangGraph flow or AgentExecutor script that does recursive or small-to-big retrieval in PolicyPulse?**

**Or would a system diagram showing how these fit together help clarify this for your intern or team?**

**You're absolutely right to call this out — blank string responses from the orchestrator are a surprisingly persistent issue in multi-agent LLM apps, and they’re *not* automatically handled just by breaking down roles into orchestrator/reviewer/worker agents. Let’s break it down and propose targeted, realistic solutions.**

**🧠 Why the Orchestrator Sometimes Returns a Blank String**

**Root Causes:**

1. **Parsing or Routing Failure – It receives an empty or malformed result from a sub-agent and passes it along unprocessed.**
2. **Guardrail Misfire – Overly strict filters (like if not response: return "") intended to prevent hallucination end up killing legitimate-but-short answers.**
3. **LLM Misbehavior – The orchestrator prompt can be misunderstood, resulting in a null response due to excessive hedging or misaligned role logic.**
4. **System Prompt-User Prompt Mismatch – If your system prompt says “you don’t answer user queries,” and the user prompt is ambiguous, the LLM may simply... do nothing.**
5. **Token Limit Error – The orchestrator might hit token limits on very long context windows and simply truncate everything downstream.**

**✅ Real Solutions (Practical, Tested Fixes)**

**✅ 1. Post-Response Guard for Empty Output**

**Add a fallback validator after the orchestrator produces a response:**

**if not response or response.strip() == "":**

**# Try again or escalate**

**response = "⚠️ Something went wrong and I couldn’t generate a valid response. Please try rephrasing your question."**

**Or use a retry logic:**

**for i in range(2):**

**response = run\_orchestrator(prompt)**

**if response and response.strip() != "":**

**break**

**else:**

**response = "Apologies, I couldn't generate a valid response after two attempts."**

**This adds self-healing logic to catch silent failures.**

**✅ 2. Force Response Format in the Prompt**

**In the root agent system prompt, add a hard constraint like:**

**You must return a JSON object in the format: {"response": "your final answer to the user"}.**

**Never return an empty string. If no information is available, say so explicitly.**

**This forces the model to avoid empty replies and gives you something structured to parse.**

**✅ 3. Insert a Self-Reflection Step**

**Have the orchestrator double-check itself before returning:**

**Before returning your response, reflect: "Have I produced a non-empty, clear answer that fulfills the user request?"**

**If not, re-invoke the appropriate sub-agent.**

**This reflection/retry step has been proven effective in frameworks like Reflexion and AutoGPT.**

**✅ 4. Use a default fallback agent**

**You could optionally introduce a lightweight "fallback explainer" agent:**

**if root\_agent\_response.strip() == "":**

**response = fallback\_agent.generate("Explain why the system failed to respond to this user input.")**

**This gives transparency to the user instead of silence.**

**✅ 5. Log, Notify, Retry**

**In production, implement structured logging like:**

**if root\_agent\_output == "":**

**logger.error("Empty response from orchestrator")**

**notify\_admin("Orchestrator failed to respond to prompt: {user\_input}")**

**You can even use this to auto-trigger fine-tuning later or flag edge cases.**

**🚫 Don’t Try This Alone:**

**Relying *only* on system prompt rules like “do not return blank” is not reliable. Even the most carefully crafted prompt will occasionally be misunderstood by the LLM, so you need runtime logic as a safety net.**

**✅ Summary: How to Handle Blank Responses in Root Agent**

| **Strategy** | **Purpose** |
| --- | --- |
| **if response == "": retry** | **Basic fallback** |
| **Enforce JSON structure** | **Avoid total blank output** |
| **Add self-reflection in prompt** | **Raise LLM awareness** |
| **Use fallback explainer agent** | **Graceful error handling** |
| **Log and flag issues** | **Monitoring + long-term fix** |
| **Force qualitative exit criteria** | **Avoid premature returns** |

**Would you like me to draft a root\_agent.py skeleton that implements retry + fallback logic and tightly binds to your existing report\_writer and faq\_agent setup?**

**Below is a practical pattern you can drop into your Google-ADK (or LangChain/Autogen) stack to eliminate the “blank-string” problem without letting yet another agent run wild.**

**1 │ What *is* the fallback\_agent?**

| **Aspect** | **Behaviour** |
| --- | --- |
| **Purpose** | **Produce *some* intelligible, policy-compliant text whenever the root\_agent returns "", None, or obvious garbage.** |
| **Scope** | **Very small: it never performs research or cites documents; it only explains, apologises, or guides the user to re-phrase.** |
| **Hierarchy** | **It is a sub-agent of the root\_agent (i.e. root\_agent can call it, but it never calls the user directly).** |
| **Output contract** | **Always returns a short, user-facing message – never blank, never more than ~50 words.** |
| **LLM** | **A cheaper / faster model is fine (e.g. Gemini Flash or GPT-3.5). No context window worries.** |

**2 │ Minimal system prompt for the fallback\_agent**

**### SYSTEM (fallback\_agent)**

**Role: FAILURE\_HANDLER**

**You are invoked \*\*only\*\* when the main orchestrator (root\_agent) fails to produce a**

**valid answer or returns an empty string.**

**Your job is to give the user a brief, polite, and helpful message:**

**1. Apologise for the hiccup (1 short sentence).**

**2. Suggest a concrete next step (e.g. "Please rephrase", "Try again in a minute").**

**3. Never claim new facts; never cite documents; never exceed 50 words.**

**4. Never return an empty string. If uncertain, default to:**

**"Sorry, something went wrong. Please try re-phrasing your request."**

**Return plain text only.**

**Why so strict? We want deterministic, safe behaviour; no citations, no extra hallucination vectors.**

**3 │ How the root-agent calls it (pseudo-code)**

**def safe\_root\_orchestrator(user\_prompt):**

**# 1. normal orchestration flow -----------------------------**

**result = root\_agent.run(user\_prompt) # may call FAQ / REPORT agents**

**# 2. minimal validation ------------------------------------**

**if not result or result.strip() == "":**

**# blank or None → delegate to fallback\_agent**

**result = fallback\_agent.run(**

**"Orchestrator returned blank for user prompt:\n"**

**f"{user\_prompt}\n"**

**"Provide user-facing fallback now."**

**)**

**# 3. last-ditch guard --------------------------------------**

**if not result or result.strip() == "":**

**# Should never happen, but prevent silence**

**result = (**

**"⚠️ Sorry, something went wrong and I couldn't generate an answer. "**

**"Please try again shortly."**

**)**

**return result**

**4 │ Will the fallback agent “talk over” the orchestrator?**

**No. The call chain is:**

**User ⇄ root\_agent ⇄ fallback\_agent (only if needed)**

**The user never sees raw fallback output unless the root agent decides to pass it along. In practice, safe\_root\_orchestrator() simply returns whatever fallback produced.**

**5 │ Qualitative exit + retry loop (optional)**

**Add one retry before falling back:**

**for attempt in range(2): # first normal run, then one retry**

**response = root\_agent.run(user\_prompt)**

**if response and response.strip():**

**break**

**else:**

**response = fallback\_agent.run("…")**

**That covers transient LLM hiccups while still guaranteeing a non-blank final message.**

**6 │ Extra hardening tips**

| **Trick** | **Why** |
| --- | --- |
| **JSON schema** | **Force orchestrator to wrap its output in `{"final": "text", "status": "ok** |
| **Token sanity check** | **len(response.split()) < 3 ⇒ treat as invalid.** |
| **Logging & alerting** | **Count fallback hits; alert if > X per hour for monitoring.** |
| **Unit tests** | **Inject dummy prompts that historically caused blanks; assert non-empty results.** |

**🟢 Result**

**With this micro-agent pattern:**

1. **The user never sees silence.**
2. **The root agent remains the single public face of the system.**
3. **You get analytics on how often the primary flow fails.**

**Feel free to copy-paste the system prompt and integration snippet directly.  
If you need a full Google-ADK AgentTool wrapper for fallback\_agent, just let me know!**

**What “Deep Research” really is**

**Across vendors the label means “an LLM wrapped inside an autonomous, tool-using loop that (a) decomposes the user task, (b) calls external tools – mainly web search/read, sometimes code or math, (c) iterates until a quality or time budget is reached, then (d) synthesises a citation-rich report.”**

**1 OpenAI Deep Research (o-series agent)**

| **Layer** | **What happens under the hood** | **Evidence** |
| --- | --- | --- |
| **Planner pass 0** | **o3 Reasoner receives the user query, produces a JSON-like “plan” (sub-questions + search keywords).** | **API doc shows a plan object returned before the first tool call (**[**cookbook.openai.com**](https://cookbook.openai.com/examples/deep_research_api/introduction_to_deep_research_api?utm_source=chatgpt.com)**)** |
| **Tool loop** | **A hard-wired “browser\_search→browser\_read→analyze” chain runs for dozens of iterations. After each read, the model updates a scratchpad and may spawn new search queries.** | **OpenAI blog: “find, analyse and pivot as needed” (**[**openai.com**](https://openai.com/index/introducing-deep-research/?utm_source=chatgpt.com)**)** |
| **Critic / self-reflection** | **After n steps (≈ 30 s compute cap) the same o3 model reviews its own draft for missing angles & citation gaps, then issues follow-up searches if needed.** | **VentureBeat write-up mentions iterative refinement and gap-filling (**[**venturebeat.com**](https://venturebeat.com/ai/openais-surprise-new-o3-powered-deep-research-shows-the-power-of-the-ai-agent-era/?utm_source=chatgpt.com)**)** |
| **Fallback rules** | ***If a search tool fails or returns 0 results* → agent retries with broader keywords; *if time budget reached* → returns partial answer plus “Further work needed.”** | **Shown in OpenAI cookbook example ("status":"partial","next\_steps":[…]) (**[**cookbook.openai.com**](https://cookbook.openai.com/examples/deep_research_api/introduction_to_deep_research_api?utm_source=chatgpt.com)**)** |
| **Output stage** | **Final answer + inline links and end-note citations; JSON envelope includes tool\_use\_trace for audit.** |  |

**Single model vs. many?  
OpenAI runs one o-series model but in two modes: “planner/critic” and “writer.” The tool calls (search, scrape, code) are handled by lightweight micro-services, not extra LLMs.**

**2 Perplexity Deep Research (Sonar-based agent)**

| **Step** | **What is different** |
| --- | --- |
| **Underlying model** | **Sonar-Large (Llama 3.3 + PPLX fine-tune).** |
| **Phase 0** | **Generates a search map, then fires parallel web searches (dozens of URLs) rather than sequential.** |
| **Phase 1** | **Uses an internal code-executor tool to run Python for stats or chart extraction (**[**usaii.org**](https://www.usaii.org/ai-insights/what-is-perplexity-deep-research-a-detailed-overview?utm_source=chatgpt.com)**).** |
| **Phase 2** | **Merges sources, then calls the model again for a “graduate-level” narrative report (≈ 1 600 words).** |
| **Guard-rails** | **Perplexity discards an answer if fewer than X unique domains are cited (anti-hallucination).** |

**3 Claude 4 “Deep Research” (Enterprise preview)**

**Anthropic hasn’t opened docs, but demo logs shared by customers show:**

* **Initial planning – Claude 4 Opus produces a task tree (“research plan”).**
* **Long context (200 k tokens) lets it paste entire papers → fewer external tool calls.**
* **Two-model hand-off – Opus drafts; cheaper Sonnet validates citations for “hallucination score.”**

**4 Common agentic building blocks you can copy**

1. **Planning prompt: “Break the task into ordered steps and decide which tool to call first.”**
2. **Tool routing: if step.tool == "search": run\_search(); elif …**
3. **Self-critique loop (max 2–3 passes):**
4. **Reflect: does the draft fully answer the initial question?**
5. **If not, list missing pieces and new search terms.**
6. **Timeout + fallback: after X seconds or Y tool-calls → stop and return partial answer with TODOs.**
7. **Final review: secondary LLM pass to check empty sections, citation format, and “no-hallucination” rules.**

**5 Why agentic “Deep Research” improves quality**

| **Problem** | **Agentic mitigation** |
| --- | --- |
| **Shallow single-shot answers** | **Planner forces multi-hop information gathering.** |
| **Source hallucination** | **Tool trace + critic step rejects uncited claims.** |
| **Blank or partial drafts** | **Loop retries until budget exhausted; fallback message if still empty.** |
| **Latency** | **Parallel searches (Perplexity) or long-context ingestion (Claude) reduce iterations.** |

**Take-away**

**“Deep Research” isn’t magic—just a scripted chain:**

1. **Plan → 2. Search/read/code in a loop → 3. Self-review → 4. Fallback/exit.**

**OpenAI’s version uses a single o-series model in two roles plus hard tools; Perplexity adds a code executor; Anthropic pairs Opus + Sonnet. You can replicate most of the pattern with your own agents by adding a planner node, a search tool, a critic loop, and a final citation checker.**

**"inputs": {"question": "What accommodations must we provide for pregnant employees?"},**

**"outputs": {"ground\_truth": "Under Title VII and the Pregnancy Discrimination Act, employers must provide reasonable accommodations for pregnancy-related conditions."}**

**},**

**{**

**"inputs": {"question": "Can we ask about pregnancy plans during hiring interviews?"},**

**"outputs": {"ground\_truth": "No, asking about pregnancy plans during hiring is illegal under pregnancy discrimination laws."}**

**},**

**{**

**"inputs": {"question": "What lactation accommodation is required under the PUMP Act?"},**

**"outputs": {"ground\_truth": "The PUMP Act requires employers to provide reasonable break time and private space for pumping breast milk."}**

**},**

**{**

**"inputs": {"question": "Are employers required to cover IVF treatments?"},**

**"outputs": {"ground\_truth": "IVF coverage requirements vary by state - some mandate coverage while others don't."}**

**},**

**{**

**"inputs": {"question": "Do fertility benefits need to include same-sex couples?"},**

**"outputs": {"ground\_truth": "Yes, fertility benefits should be inclusive and cover same-sex couples equally."}**

**}**

**3. Better Auto-Start Alternative**

**Instead of auto-starting in init\_session\_state(), add this to your chat\_interface() function:**

**def chat\_interface():**

**"""Main chat interface"""**

**st.title("🏥 Policy Pulse Agent")**

**# Auto-start conversation here (won't cause re-run loop)**

**if not st.session\_state.current\_session\_id:**

**if st.button("🆕 Start New Conversation", use\_container\_width=True):**

**start\_new\_conversation()**

**st.rerun()**

**st.info("👈 Click 'Start New Conversation' to begin!")**

**return**

**# Rest of your chat interface...**

**For Streamlit, add streaming display:**

**python**

***# In your chat\_interface function, replace the response section:***

**with st.chat\_message("assistant"):**

**message\_placeholder = st.empty()**

**full\_response = ""**

**async for event in runner.run\_async(...):**

**if hasattr(event, 'content') and hasattr(event.content, 'parts'):**

**for part in event.content.parts:**

**if hasattr(part, 'text') and part.text is not None:**

**full\_response += part.text**

**message\_placeholder.markdown(full\_response + "▌") *# Show typing cursor***

**message\_placeholder.markdown(full\_response) *# Final response***

**Streamlit has several async handling issues that can cause slowness. Here are specific optimizations:**

**1. Use Streamlit's Built-in Async Support**

**Replace your current async handling with Streamlit's native async:**

**# Instead of asyncio.run(), use st.write with async**

**async def get\_agent\_response\_streaming(user\_message):**

**"""Stream response from agent"""**

**message\_content = types.Content(**

**role='user',**

**parts=[types.Part(text=user\_message)]**

**)**

**# Return the async generator directly**

**return runner.run\_async(**

**user\_id=st.session\_state.user\_id,**

**session\_id=st.session\_state.current\_session\_id,**

**new\_message=message\_content**

**)**

**# In your chat interface:**

**if prompt := st.chat\_input("Ask about reproductive & fertility health policies..."):**

**# Add user message immediately**

**st.session\_state.messages.append({"role": "user", "content": prompt})**

**with st.chat\_message("user"):**

**st.write(prompt)**

**# Stream assistant response**

**with st.chat\_message("assistant"):**

**message\_placeholder = st.empty()**

**full\_response = ""**

**# Use async with proper context**

**async def stream\_response():**

**async for event in runner.run\_async(**

**user\_id=st.session\_state.user\_id,**

**session\_id=st.session\_state.current\_session\_id,**

**new\_message=types.Content(role='user', parts=[types.Part(text=prompt)])**

**):**

**if hasattr(event, 'content') and hasattr(event.content, 'parts'):**

**for part in event.content.parts:**

**if hasattr(part, 'text') and part.text:**

**yield part.text**

**# Stream the response**

**for chunk in asyncio.run(stream\_response()):**

**full\_response += chunk**

**message\_placeholder.markdown(full\_response + "▌")**

**message\_placeholder.markdown(full\_response)**

**2. Use Connection Pooling**

**Create a session pool to avoid recreating connections:**

**# At the top of your streamlit\_app.py**

**@st.cache\_resource**

**def get\_agent\_runner():**

**"""Cache the runner to avoid recreating connections"""**

**return runner**

**# Use the cached runner**

**cached\_runner = get\_agent\_runner()**

**3. Optimize with Threading**

**Use threading for the async calls:**

**import threading**

**from concurrent.futures import ThreadPoolExecutor**

**def run\_agent\_sync(user\_message):**

**"""Run agent in thread pool"""**

**async def \_run():**

**message\_content = types.Content(**

**role='user',**

**parts=[types.Part(text=user\_message)]**

**)**

**response\_text = ""**

**async for event in runner.run\_async(**

**user\_id=st.session\_state.user\_id,**

**session\_id=st.session\_state.current\_session\_id,**

**new\_message=message\_content**

**):**

**if hasattr(event, 'content') and hasattr(event.content, 'parts'):**

**for part in event.content.parts:**

**if hasattr(part, 'text') and part.text:**

**response\_text += part.text**

**return response\_text**

**return asyncio.run(\_run())**

**# In your chat interface:**

**with st.spinner("Thinking..."):**

**response = run\_agent\_sync(prompt)**

**4. Pre-load Heavy Components**

**# Cache expensive imports and initializations**

**@st.cache\_resource**

**def initialize\_agent\_components():**

**"""Pre-load agent components"""**

**# This runs once and stays in memory**

**return {**

**'runner': runner,**

**'session\_service': session\_service,**

**'agent': root\_agent**

**}**

**components = initialize\_agent\_components()**

**5. Optimize Pinecone Calls**

**# Add connection pooling for Pinecone**

**@st.cache\_data(ttl=60) # Cache for 1 minute**

**def cached\_retrieve\_context(query: str) -> str:**

**"""Cache Pinecone results briefly"""**

**return \_retrieve\_context(query)**

**Most Effective: Use st.spinner and Threading**

**The simplest and most effective approach:**

**def chat\_interface():**

**# ... existing code ...**

**if prompt := st.chat\_input("Ask about reproductive & fertility health policies..."):**

**# Add user message**

**st.session\_state.messages.append({"role": "user", "content": prompt})**

**# Show all messages**

**for message in st.session\_state.messages:**

**with st.chat\_message(message["role"]):**

**st.write(message["content"])**

**# Get response with spinner**

**with st.chat\_message("assistant"):**

**with st.spinner("🤔 Analyzing policies..."):**

**response = asyncio.run(get\_agent\_response(prompt))**

**st.write(response)**

**# Add response to messages**

**st.session\_state.messages.append({"role": "assistant", "content": response})**

**# Auto-save and refresh**

**save\_conversation(...)**

**st.rerun()**

**The key is using st.spinner() to show progress and avoiding complex async patterns that Streamlit doesn't handle well.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

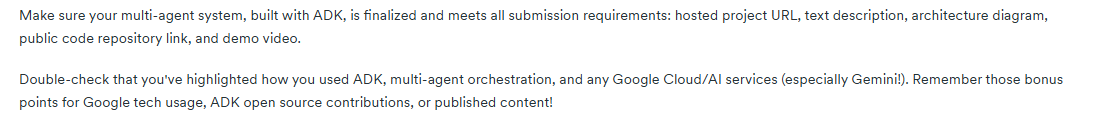
**POLICYPULSE AI AGENT**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A white background with black text

AI-generated content may be incorrect.**

****

1. streamlit\_app.py file – this is what I will run when hosting on the server. It will activate the root\_agent file which is basic\_agents/step01/policy\_pulse\_agent/agent.py . It will also provide a chat UI that shows the current conversation, but also previous conversations in the left-hand side that users can activate. The previous conversation would have been downloaded from the Supabase database. Once the agent responds to a user the message will be saved ie the user will not need to click any button in order for the conversation to be saved The prior conversations will have natural language titles or names rather than uuid ones.
2. Login / sign-up script: This will allow a user to either sign up or login to his profile to see his own chat history and also to start a new conversation. I think we can have it at the front of the streamlit page. Note that we are not doing Google authentication or Auth0. A potential user will just be asked to sign up with his email address, userid and password. These will be logged in an appropriate place
3. Session Manager – I’m not sure this is needed but it would just be to support the other 2 scripts

OPENROUTER\_API\_KEY=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

# Supabase DB

SUPABASE\_PASSWD=HRRWhrsJHiNtOKYD

SUPABASE\_URL=https://heagdxjxnimnbfrhxrgr.supabase.co

SUPABASE\_API=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImhlYWdkeGp4bmltbmJmcmh4cmdyIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTA1MDg5ODksImV4cCI6MjA2NjA4NDk4OX0.5Lo0ejdhGbLByLclHxYU4K7ijwsS6AUnNuvLO8Ecth0

#DATABASE\_URL=postgresql+pg8000://postgres:HRRWhrsJHiNtOKYD@db.heagdxjxnimnbfrhxrgr.supabase.co:5432/postgres # this doesn't work wth IPv4

# - this below is the session pooler type of connection

DATABASE\_URL=postgresql://postgres.heagdxjxnimnbfrhxrgr:HRRWhrsJHiNtOKYD@aws-0-eu-west-1.pooler.supabase.com:5432/postgres

Here are some thoughts for additional code files I need. They are rough ideas. Try to come up with native and elegant ways of achieving these things that do not require hundreds of lines of code

**3. Regarding Text Splitters**

The RecursiveCharacterTextSplitter is indeed one of the most advanced text splitters currently available in the LangChain ecosystem. However, there are some others worth considering:

1. **MarkdownHeaderTextSplitter**: Specifically for markdown content, splits based on header structure
2. **TokenTextSplitter**: Splits by token count rather than character count (better for LLM context limitations)
3. **SemanticChunker**: Uses embeddings to create semantically coherent chunks (available in some newer libraries)
4. **NLTKTextSplitter**: Uses NLTK to split on linguistic boundaries
5. **SpacyTextSplitter**: Uses spaCy for more linguistically-aware splitting

For a compliance bot, some newer approaches that might be worth exploring include:

* **Hierarchical chunking**: Creating chunks at multiple levels of granularity (paragraph, section, document)
* **Semantic chunking**: Using embeddings to determine natural semantic boundaries
* **Context-aware chunking**: Preserving sections that should remain together based on content analysis

If you're planning to stick with RecursiveCharacterTextSplitter, one enhancement could be to customize the separators list to better respect the structure of compliance documents, for example by prioritizing splitting at section boundaries or after complete regulatory citations.

The RecursiveCharacterTextSplitter is definitely a strong choice for your use case, especially since it attempts to respect natural document boundaries during splitting.

Questions:

explain the impact of mental stress on reproductive health

what is reproductive health?

you didn't include the effects on men

A diagram of a diagram

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a document

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a white and black page

AI-generated content may be incorrect.

A screenshot of a document

AI-generated content may be incorrect.

Your question: what benefits do companies typically provide for women undergoing IVF?

Retrieving relevant chunks for query: what benefits do companies typically provide for women undergoing IVF?

Search response: {'result': {'hits': [{'\_id': 'Module 2 Lesson 3 Chat GPT Prompt\_chunk\_1',

'\_score': 0.5471951961517334,

'fields': {'text': 'Here are case studies and worked '

'examples of how companies in '

'different regions (UK, Europe, US, '

'LATAM, and Africa) have successfully '

'implemented reproductive '

'health-inclusive workplace '

'policies.\n'

'\n'

'Case Study 1: United Kingdom – '

'Fertility and Family-Forming '

'Benefits\n'

'Company: NatWest Group (Financial '

'Services, UK)\n'

'Policy Highlights:\n'

'Offers fertility benefits, including '

'IVF support, egg freezing, and '

'adoption assistance.\n'

'Provides up to 7 days of paid leave '

'for fertility treatments.\n'

'Has a dedicated employee network for '

'fertility and baby loss support.\n'

'Implementation Insights:\n'

'Worked with HR and insurance '

'providers to introduce inclusive '

'fertility policies.\n'

'Conducted internal awareness '

'campaigns to normalize fertility '

'conversations.\n'

'Partnered with external fertility '

'experts to provide coaching and '

'digital resources.'}},

{'\_id': 'v3 Script Module 2 lesson 3\_chunk\_13',

'\_score': 0.5098229050636292,

'fields': {'text': 'Workplace Adjustments & Support\n'

'Quiet spaces for employees '

'undergoing treatment.\n'

'Adjusted workloads during '

'challenging times.\n'

'Mental health and peer support '

'groups.\n'

'(With We Are Eden’s Online Course & '

'Support Services for Companies)\n'

'Case Study 1: United Kingdom – '

'Fertility and Family-Forming '

'Benefits\n'

'Company: NatWest Group (Financial '

'Services, UK)\n'

'Policy Highlights:\n'

'Offers fertility benefits, including '

'IVF support, egg freezing, and '

'adoption assistance.\n'

'Provides up to 7 days of paid leave '

'for fertility treatments.\n'

'Has a dedicated employee network for '

'fertility and baby loss support.\n'

'How We Are Eden Can Help:\n'

'✅ Online Course: Helps HR teams '

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'✅ Consulting & Training: We Are Eden '

'provides custom policy audits and '

'workshops on best practices for '

'introducing fertility-related '

'benefits.'}},

{'\_id': 'v2 Script and Lesson outline\_chunk\_6',

'\_score': 0.5087902545928955,

'fields': {'text': 'Section 2bii. Company Policies & '

'Benefits\n'

'Detail leave policies (fertility '

'leave, miscarriage leave, menstrual '

'health accommodations).\n'

'Flexible working options for medical '

'treatments or symptom management.\n'

'Health insurance and financial '

'support for fertility treatments.\n'

'2biii Creating a Reproductive '

'Health-Inclusive Workplace (10-15 '

'min)\n'

'Key Components of a Strong Workplace '

'Policy\n'

'Fertility and family-forming '

'benefits (IVF, surrogacy, egg '

'freezing support).\n'

'Menstrual, menopause, and '

'reproductive health leave.\n'

'Access to expert support (coaching, '

'peer groups, digital tools).\n'

'How Leaders Can Drive Change\n'

'Embedding reproductive health into '

'DEI & Wellbeing strategy.\n'

'Setting the tone for psychological '

'safety.\n'

'Allocating budget and measuring '

'ROI.'}},

{'\_id': 'v2 Script and Lesson outline Module 2 Creating a '

'Reproductive and Fertility Health at work '

'guide\_chunk\_6',

'\_score': 0.5087902545928955,

'fields': {'text': 'Section 2bii. Company Policies & '

'Benefits\n'

'Detail leave policies (fertility '

'leave, miscarriage leave, menstrual '

'health accommodations).\n'

'Flexible working options for medical '

'treatments or symptom management.\n'

'Health insurance and financial '

'support for fertility treatments.\n'

'2biii Creating a Reproductive '

'Health-Inclusive Workplace (10-15 '

'min)\n'

'Key Components of a Strong Workplace '

'Policy\n'

'Fertility and family-forming '

'benefits (IVF, surrogacy, egg '

'freezing support).\n'

'Menstrual, menopause, and '

'reproductive health leave.\n'

'Access to expert support (coaching, '

'peer groups, digital tools).\n'

'How Leaders Can Drive Change\n'

'Embedding reproductive health into '

'DEI & Wellbeing strategy.\n'

'Setting the tone for psychological '

'safety.\n'

'Allocating budget and measuring '

'ROI.'}},

{'\_id': 'Module 2 lesson 3 script inc case studies and '

'WAE edge\_chunk\_13',

'\_score': 0.5017654895782471,

'fields': {'text': 'Workplace Adjustments & Support\n'

'Quiet spaces for employees '

'undergoing treatment.\n'

'Adjusted workloads during '

'challenging times.\n'

'Mental health and peer support '

'groups.\n'

'(With We Are Eden’s Online Course & '

'Support Services for Companies)\n'

'Case Study 1: United Kingdom – '

'Fertility and Family-Forming '

'Benefits\n'

'Company: NatWest Group (Financial '

'Services, UK)\n'

'Policy Highlights:\n'

'Offers fertility benefits, including '

'IVF support, egg freezing, and '

'adoption assistance.\n'

'Provides up to 7 days of paid leave '

'for fertility treatments.\n'

'Has a dedicated employee network for '

'fertility and baby loss support.\n'

'How We Are Eden Can Help:\n'

'✅ Online Course: Helps HR teams '

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'Company: NatWest Group (Financial '

'Services, UK)\n'

'Policy Highlights:\n'

'Offers fertility benefits, including '

'IVF support, egg freezing, and '

'adoption assistance.\n'

'Provides up to 7 days of paid leave '

'for fertility treatments.\n'

'Has a dedicated employee network for '

'fertility and baby loss support.\n'

'How We Are Eden Can Help:\n'

'✅ Online Course: Helps HR teams '

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'Services, UK)\n'

'Policy Highlights:\n'

'Offers fertility benefits, including '

'IVF support, egg freezing, and '

'adoption assistance.\n'

'Provides up to 7 days of paid leave '

'for fertility treatments.\n'

'Has a dedicated employee network for '

'fertility and baby loss support.\n'

'How We Are Eden Can Help:\n'

'✅ Online Course: Helps HR teams '

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'fertility and baby loss support.\n'

'How We Are Eden Can Help:\n'

'✅ Online Course: Helps HR teams '

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'structure fertility benefits that '

'meet employee needs while complying '

'with UK workplace regulations.\n'

'with UK workplace regulations.\n'

'✅ Consulting & Training: We Are Eden '

'provides custom policy audits and '

'workshops on best practices for '

'introducing fertility-related '

'benefits.'}}]},

'usage': {'embed\_total\_tokens': 15, 'read\_units': 6}}

Found 5 hits

Retrieved 5 relevant chunks

Querying SONAR API...

Compliance Bot Response:

Companies typically provide a range of benefits to support women undergoing IVF, focusing on both financial assistance and workplace accommodations. Here are some common benefits:

'usage': {'embed\_total\_tokens': 15, 'read\_units': 6}}

Found 5 hits

Retrieved 5 relevant chunks

Querying SONAR API...

Compliance Bot Response:

Companies typically provide a range of benefits to support women undergoing IVF, focusing on both financial assistance and workplace accommodations. Here are some common benefits:

Retrieved 5 relevant chunks

Querying SONAR API...

Compliance Bot Response:

Companies typically provide a range of benefits to support women undergoing IVF, focusing on both financial assistance and workplace accommodations. Here are some common benefits:

Compliance Bot Response:

Companies typically provide a range of benefits to support women undergoing IVF, focusing on both financial assistance and workplace accommodations. Here are some common benefits:

Companies typically provide a range of benefits to support women undergoing IVF, focusing on both financial assistance and workplace accommodations. Here are some common benefits:

## Financial Support

- \*\*IVF Coverage\*\*: Many companies offer financial support for IVF treatments. For example, companies like A&E provide a $50,000 lifetime maximum for IVF treatment, while Bank of America offers unlimited IVF coverage after an infertility diagnosis[5].

- \*\*Egg Freezing and Storage\*\*: Some companies, such as Adobe and Amazon, offer benefits for egg freezing and storage, which can be part of broader fertility support packages[5].

- \*\*Adoption and Surrogacy Assistance\*\*: Companies like American Express provide benefits for adoption and surrogacy, recognizing the diverse paths to family formation[5].

## Workplace Accommodations

- \*\*Paid Leave\*\*: Companies often provide paid leave specifically for fertility treatments. For instance, NatWest Group offers up to 7 days of paid leave for fertility treatments[Document 1][Document 2].

- \*\*Flexible Working Options\*\*: Flexible work schedules can help employees manage medical appointments and treatments without impacting their work[Document 3][Document 4].

- \*\*Mental Health Support\*\*: Many companies offer mental health support, including peer groups and counseling services, to help employees cope with the emotional aspects of fertility treatments[Document 2][Document 5].

## Access to Expert Support

- \*\*Digital Resources and Coaching\*\*: Companies may partner with fertility experts to provide coaching and digital resources, enhancing employee support during fertility journeys[Document 1][Document 5].

These benefits not only support employees' health and well-being but also contribute to a more inclusive and supportive work environment.

Your question:

Recent financial penalties incurred by companies related to reproductive health primarily stem from violations of privacy laws, particularly the Health Insurance Portability and Accountability Act (HIPAA) in the United States, as well as from workplace discrimination and failure to accommodate reproductive health conditions in the UK and EU.

## Financial Penalties Related to Reproductive Health Privacy Violations (HIPAA)

In the U.S., the most notable financial penalties have been imposed due to violations of HIPAA rules concerning the privacy and security of protected health information (PHI), including reproductive health information.

- \*\*Holy Redeemer Family Medicine Settlement (2024):\*\* A Pennsylvania hospital paid a settlement of $35,581 after it was found to have impermissibly disclosed a female patient’s reproductive health information to her prospective employer without proper authorization. The disclosure included sensitive details such as surgical, gynecological, and obstetric history beyond the authorized test result. The hospital also agreed to implement a corrective action plan monitored by the Office for Civil Rights (OCR) for two years to prevent future violations[3].

- \*\*HIPAA Enforcement Trends:\*\* The OCR has been increasingly active in enforcing HIPAA compliance, with 2018 being a record year for fines totaling over $28 million. In 2019 and 2020, enforcement continued strongly, with 19 settlements in 2020 alone totaling $13.5 million. Many penalties in 2020 were related to failures in risk analysis, poor risk management, lack of policies, impermissible disclosures, and inadequate safeguards, which often include reproductive health information[1][5].

- \*\*Regulatory Developments and Legal Challenges:\*\* New HIPAA regulations proposed in 2025 aim to strengthen protections for reproductive health care privacy. However, these have faced legal challenges from several states, reflecting the contentious nature of reproductive health privacy protections. Lawsuits have been filed by anti-abortion groups against the 2024 HIPAA Privacy Rule designed to support reproductive health care privacy, and some states have sued the Department of Health and Human Services (HHS) to drop these regulations[2][4].

## Financial Penalties and Legal Cases Related to Workplace Reproductive Health Compliance (UK and EU)

In the UK and EU, companies have faced significant financial penalties and reputational damage due to non-compliance with reproductive health-related workplace laws, particularly concerning discrimination and failure to accommodate reproductive health conditions.

- \*\*Failure to Accommodate Reproductive Health Conditions:\*\* A UK employment tribunal awarded £31,707.34 to an employee with endometriosis who was unfairly dismissed after her employer withdrew a previously agreed work-from-home arrangement. This case highlights the financial risks companies face when failing to accommodate reproductive health conditions such as endometriosis[1][3].

- \*\*Discrimination Related to Fertility Treatments:\*\* Another UK case involved Mrs. Benton, who was dismissed after undergoing IVF treatment and suffering a miscarriage. The tribunal found that her dismissal was due to harassment and victimization related to her fertility treatment, resulting in a successful claim against the employer. Such cases underscore the importance of non-discriminatory practices and support for employees undergoing fertility treatments[2].

- \*\*EU Directives and National Laws:\*\* The EU Work-Life Balance Directive, Pregnant Workers Directive, and Equal Treatment Directive set minimum standards for parental leave, flexible working, and protection against discrimination based on sex, pregnancy, and maternity. Non-compliance with these directives and national laws implementing them can lead to legal challenges and financial penalties, although specific recent penalty amounts were not detailed in the documents[4].

## Summary

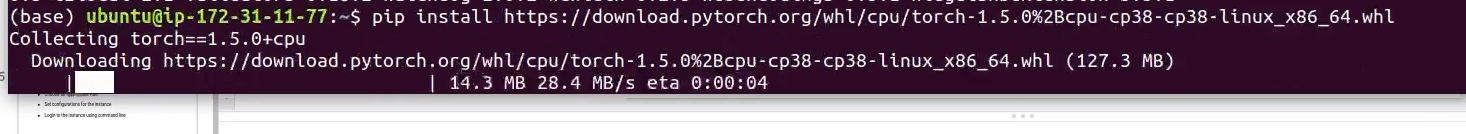
Recent financial penalties related to reproductive health fall into two main categories:

1. \*\*Privacy Violations under HIPAA in the U.S.:\*\* These include settlements for unauthorized disclosure of reproductive health information, with penalties ranging from tens of thousands to millions of dollars annually across multiple cases. The Holy Redeemer case is a recent example with a $35,581 settlement. Enforcement is increasing, with new regulations and legal challenges ongoing[1][3][5].

2. \*\*Workplace Discrimination and Failure to Accommodate in the UK and EU:\*\* Financial penalties and tribunal awards have been made for unfair dismissal and discrimination related to reproductive health conditions like endometriosis and fertility treatments. Awards can be substantial, such as the £31,707.34 awarded in the endometriosis case, reflecting the legal and financial risks of non-compliance[1][2][3][4].

These cases collectively demonstrate the growing legal and financial consequences for companies that fail to protect reproductive health privacy or accommodate reproductive health needs in the workplace. Organizations are advised to maintain up-to-date policies, provide necessary accommodations, and ensure compliance with evolving legal standards to mitigate these risks.

How to install cpu version of torch



A computer screen with white text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Auth0 sample user

{

"sub": "auth0|6850069ef6be34807d374301",

"nickname": "nnamdi.odozi",

"name": "nnamdi.odozi@gmail.com",

"picture": "https://s.gravatar.com/avatar/caeaf0ec7296e9def1d0e4258c371286?s=480&r=pg&d=https%3A%2F%2Fcdn.auth0.com%2Favatars%2Fnn.png",

"updated\_at": "2025-06-16T11:57:48.905Z"

}