Research for Local LLM Approach

**Introduction**

The Epic Retirement Institute requires an AI-powered tool to deliver conversational, compliant retirement guidance. While rule-based systems avoid hallucinations, they lack flexibility. A locally hosted Large Language Model (LLM) offers a middle ground: leveraging generative AI’s natural language capabilities while retaining control over data sources, compliance, and privacy. This report evaluates the feasibility of a local LLM solution trained/fine-tuned on curated datasets from trusted sources (ATO, ASIC, client content).

**Core Components of a Local LLM Solution**

Model Selection & Customization

Local LLM Options:

* Llama 2 (7B/13B parameter variants): Open-source, commercially usable, balances performance and resource needs.
* Mistral-7B: Efficient for smaller hardware, strong reasoning for its size.
* Falcon-40B: High accuracy but requires more computational power.

Fine-Tuning

Train models on the client’s books, articles, and structured data from ATO/Services Australia to align responses with their voice and compliance requirements.

Use LoRA (Low-Rank Adaptation) for cost-effective fine-tuning without full model retraining.

Retrieval-Augmented Generation (RAG)

Purpose: Ground the LLM in factual, up-to-date data while minimizing hallucinations.

Implementation:

Build a vector database (e.g., FAISS or ChromaDB) of trusted sources: ATO rules, ASIC guides, client-authored content. For each query, retrieve relevant snippets and prompt the LLM to synthesize answers only from these snippets.

Example: A user asks, “What’s the concessional cap for super?” → RAG retrieves the ATO’s latest cap ($27,500) and the LLM formats it conversationally.

Strict Prompt Engineering

System Prompts: Enforce compliance and tone:

“

You are a factual assistant. Answer only using data from the provided documents. Never provide personal advice. Always cite sources.

”

Response Guardrails

* Block responses that imply recommendations (e.g., “You should…”).
* Flag uncertain answers for human review (e.g., “I’m unsure; consult Services Australia for…”).

Local Deployment & Privacy

On-Premises Hosting: Run the LLM on local servers or private cloud infrastructure to ensure user data never leaves Australia.

Hardware Requirements

Llama 2-7B: ~10GB VRAM (1x NVIDIA RTX 3090/4090).

Quantized Models: Use tools like llama.cpp or GPTQ to reduce GPU memory demands.

Compliance and Auditability

Logging: Track all user interactions and sources used for responses.

Escalation: Escalate complex queries to moderators (e.g., via a “Flag for Review” button).

**Benefits of Local LLM vs. Cloud-Based LLMs**

Data Privacy: No third-party API risks; user data stays on-premises.

Customization: Fine-tune models to the client’s voice and compliance needs.

Cost Control: Avoid per-query fees from cloud providers (e.g., OpenAI).

Transparency: Audit model behavior and training data directly.

**Challenges & Mitigations**

| **Challenge** | **Mitigation** |
| --- | --- |
| **Hallucinations** | Strict RAG pipelines + prompt engineering to restrict answers to sourced data. |
| **Hardware Costs** | Use quantized models (e.g., 4-bit Llama) or smaller variants (Mistral-7B). |
| **Regulatory Compliance** | Pre-approve all training data; regular audits with legal/financial experts. |
| **Response Latency** | Optimize inference with tools like **vLLM** or **TGI** (TensorRT-LLM). |

**Comparison to Non-LLM (Rule-Based) Approach**

| **Factor** | **Local LLM** | **Rule-Based** |
| --- | --- | --- |
| **Flexibility** | High (handles varied phrasing) | Low (rigid keyword matching) |
| **Development Cost** | Moderate (fine-tuning, RAG setup) | Low (simple regex/databases) |
| **Compliance Risk** | Moderate (requires strict RAG/prompt control) | Low (fully deterministic) |
| **User Experience** | Natural, conversational | Menu-driven, transactional |

Conclusion

A local LLM with RAG offers clear benefits such as natural language understanding, scalability, and alignment with the brand, but it is not a practical option for this project. The associated costs, including the need for enterprise-grade hardware or cloud infrastructure and the specialist expertise required for fine-tuning and pipeline development, are significant. The technical complexity of implementing safeguards like RAG, prompt engineering, and auditing, along with integrating compliance workflows, would exceed the team's current capacity and stretch the timeline. With only one year to deliver and a focus on rapid prototyping, developing and validating an LLM-based solution would consume too many resources. A rule-based chatbot, as outlined in the previous Non-LLM report, remains the most suitable choice. It allows for quick deployment, keeps costs low, and ensures predictable, auditable outputs that meet the client's compliance and risk management needs.

**Non-LLM Solution Research**

To design a conversational, non-LLM-based chatbot that helps users find retirement information (without offering financial advice), a rule-based architecture can be effectively implemented. Rule-based chatbots rely on manually created decision trees, intent recognition models, and pattern-matching techniques rather than generative AI, making them predictable, transparent, and compliant — essential for financial information use cases.

## **Core Components for a Rule-Based Retirement Chatbot**

1. **Intent Recognition via Keyword or Phrase Matching** The bot must identify the user's intent based on keywords and common sentence patterns. This can be achieved through:  
   * **Pattern libraries**: Regular expressions (Regex) to detect key intents such as "superannuation rules", "retirement age", "pension eligibility", "contribution caps", "early retirement penalties", etc.
   * **Synonym mapping**: Incorporating synonyms and alternate phrasing (e.g., "super" = "superannuation") to expand detection.
   * **Fuzzy matching**: Using basic Levenshtein Distance (edit distance) algorithms to tolerate small typos like "suprannuation" instead of "superannuation".
2. **Decision Trees and State Machines** A finite state machine (FSM) could be used to handle structured conversations. The bot moves between predefined states based on user input. Example:  
   * State 1: Greeting
   * State 2: Identify financial topic
   * State 3: Ask clarification questions if intent unclear
   * State 4: Output informational database links or factual summaries
   * State 5: Offer next steps (e.g., "Would you like to know more about eligibility?")
   * State 6: Goodbye/loop
3. **Database Integration for Reliable Information** Instead of generating content, the bot should pull directly from verified financial data sources (e.g., ATO retirement rules, ASIC MoneySmart guidelines, Centrelink policies).  
   * Simple retrieval can be built using **keyword-to-snippet mapping**, where certain intents retrieve pre-written FAQ-style answers linked to databases.
   * An internal knowledge base could be indexed (e.g., SQLite or a lightweight document database) for fast lookups.
4. **Handling Incorrect or Ambiguous Inputs** The bot should:  
   * Detect when user input doesn't match any intent ("I'm sorry, I didn't quite understand. Are you asking about retirement age, superannuation, or pension eligibility?").
   * Offer clarification menus if confidence is low.
   * Allow users to restart or rephrase easily ("You can type 'start over' at any time").
5. **Context Awareness (Session Memory)** Basic session memory is needed to remember the topic in progress.  
    E.g., if a user starts asking about "early withdrawal", the bot remembers this context even if the user later says "how much can I take out?"
6. **Fallback and Escalation Strategies** Since the bot cannot offer advice:  
   * It must consistently disclaim ("Please note: This is general information only and not personal financial advice.")
   * If a query exceeds its capabilities, escalate with: "For personalised advice, we recommend contacting a licensed financial advisor."

## **Suitable Technologies and Tools (Non-LLM)**

| **Technology** | **Purpose** |
| --- | --- |
| **Rasa Open Source** | Intent recognition and dialogue management via rule-based pipelines (non-LLM mode). Highly customizable and open source. |
| **Dialogflow CX (without ML)** | Google’s Dialogflow has a non-ML mode allowing you to build rule-based flows via intent matching and explicit training phrases. |
| **Botpress** | Open-source conversational AI platform with strong rule-based flow builder — can handle complex trees without LLMs. |
| **Snips NLU** | Lightweight natural language understanding tool, designed for rule-based intents without cloud-based models. |
| **Regex + Flask/Node.js Custom Backend** | A custom lightweight solution with regex matching for intents, simple backend APIs for knowledge base querying. |
| **Finite State Machine Libraries (e.g., transitions in Python)** | Manage chatbot conversational states effectively and predictably. |

## **Benefits of Rule-Based over LLM for this Use Case**

* **Control and Compliance**: Ensures outputs only contain vetted information; avoids hallucinations or unauthorised advice.
* **Transparency**: Every answer path is visible and editable, critical for auditing.
* **Predictable Responses**: Important when dealing with sensitive user financial queries.
* **Lightweight and Efficient**: Lower computational needs compared to LLMs; faster responses even on limited infrastructure.

A rule-based chatbot using intent recognition, keyword mapping, FSMs, and verified database querying is a strong solution for building a retirement information assistant. Platforms like Rasa, Botpress, or custom-built regex-based bots offer practical, compliant alternatives without relying on large language models. While slightly more rigid than AI-based bots, their predictability and safety are key advantages when dealing with sensitive financial information where offering advice is legally restricted.

Retrieval-Augmented Generation (RAG)

Wikipedia Definition:

Retrieval-augmented generation (RAG) is a technique that enables generative artificial intelligence

(Gen AI) models to retrieve and incorporate new information. It modifies interactions with a large

language model (LLM) so that the model responds to user queries with reference to a specified set of

documents, using this information to supplement information from its pre-existing training data.

Independent Interpretation:

RAG is a method that helps AI models give better answers by letting them look things up before

responding. Instead of relying only on what the model has memorized, RAG searches a database for

relevant information and then uses that information to generate a more accurate and helpful

response.

It’s like giving the model a mini search engine and a writing assistant at the same time: first it finds

useful facts, then it writes an answer based on them.

Key Concepts

• Retriever: Finds relevant documents from a large corpus.

• Generator: Uses those documents + original query to generate a response.

• Query -> Retriever -> Docs -> Generator -> Answer

Pros

• More factual, access to up-to-date or domain-specific data.

• Less hallucination as it doesn’t rely solely on LLM’s internal memory.

• Flexible knowledge updates: just update the database, not the model.

• Smaller model size possible, since external memory does the heavy lifting.

Cons

• Slower: Retrieval adds latency.

• Bad documents = bad generation.

• Complex, two components (retriever + generator) to manage and tune.

Implementation Overview

1. Retriever

• Dense (DPR, FAISS) or sparse (BM25) models

• Input: Query -> Output: relevant docs

2. Generator

• Usually seq2seq models like BART, T5, or GPT variants

• Takes query + docs -> Generates answer

3. Common Flow:

User Query

↓

Dense Retriever (e.g., DPR)

↓

Docs from Knowledge Base

↓

Seq2Seq Generator (e.g., BART)

↓

Generated Answer

Use Cases

• Open-domain QA (e.g., Chatbots, Search Assistants)

• Enterprise AI (access private knowledge bases)

• Scientific/Legal document querying

**Exploration into OpenAI API Platform**

As part of our research into implementing potential solutions for our retirement chatbot, I explored the OpenAI API, specifically its flexibility, range of parameters and cost to consider what can meet the functionality efficiently.

Inside the API

The OpenAI API gives many functions and options to customise your LLM. Furthermore, it has the option to test functionality with the chosen parameters. This most useful of these options for building the chat bot is the ability to add data files for the LLM to reference. This would allow us to import critical information sites (such as the ATO) for the bot to reference.

Parameters

| **Parameter** | **Purpose** | **Typical Range/Options** |
| --- | --- | --- |
| temperature | Controls randomness | 0.0 to 1.0 (e.g., 0.2–0.8) |
| max\_tokens | Limits the number of tokens in the output | Any positive integer (e.g., 100– 2000) |
| top\_p | Alternative to temperature | 0.0 to 1.0 (common around  0.9) |
| frequency\_penalty | Reduces the likelihood of repeating the same lines/phrases | 0.0 to 2.0 |
| presence\_penalty | Encourages the model to talk about new topics | 0.0 to 2.0 |
| stop | Specifies one or more stopping sequences to end the output | List of strings (e.g., ["\n", "End of message"]) |
| system\_prompt | Sets overall behavior and tone for the model | Custom string |
| logit\_bias | Adjusts probability of specific tokens appearing | Map of token IDs to bias values |

Analysis of price

| **Provider** | **Model** | **Input Cost (per 1M tokens)** | **Output Cost (per 1M tokens)** | **Notes** |
| --- | --- | --- | --- | --- |
| OpenAI | GPT-3.5 Turbo | $3.00 | $6.00 | Cost-effective for general-purpose tasks. |
| OpenAI | GPT-4 | $30.00 | $60.00 | High performance for complex tasks. |
| OpenAI | GPT-4o | $2.50 | $10.00 | Optimized for speed and cost. |
| OpenAI | GPT-4o Mini | $0.15 | $0.60 | Suitable for lightweight tasks. |
| DeepSeek | DeepSeekR1 (cache  hit) | $0.14 | $2.19 | Advanced reasoning capabilities. |
| DeepSeek | DeepSeekR1 (cache miss) | $0.55 | $2.19 | Higher cost when cache is missed. |
| DeepSeek | DeepSeekV3 | $0.27 | $1.10 | Balanced performance and cost. |
| DeepSeek | DeepSeek Chat 8B | $0.20 | $0.60 | Suitable for general chat applications. |
| DeepSeek | DeepSeek Chat 67B | $1.00 | $3.00 | High-capacity model for complex tasks. |

Price of Average Interaction

GPT-4o Mini has the cheapest cost out of all the compared models, with 1000 interacts costing approximately 50 cents. Even with such a small cost per interaction, it should be considered what reach the model will have where a larger user base could build up cost over time. This should be considered respect to the industry partners goals and business model.

| Model | Input (500) | Output (750) | Total |
| --- | --- | --- | --- |
| OpenAI GPT-3.5 Turbo | $0.0015 | $0.0045 | $0.006 |
| OpenAI GPT-4 | $0.015 | $0.045 | $0.06 |
| OpenAI GPT-4o Mini | $0.000075 | $0.00045 | $0.000525 |
| DeepSeek-R1 | $0.00007 | $0.0016425 | $0.0017125 |
| DeepSeek-V3 | $0.000135 | $0.000825 | $0.00096 |

Advantages and Disadvantages

| **Advantages** | **Details** |
| --- | --- |
| **Faster**  **Development** | APIs provide out of the box pre-built models which reduces development time and complexity compared to training an LLM. |
| **Customisable** | Built in functionality to add data sources and parameters can help customise the bot as needed. |
| **Language Support** | Built-in language capabilities can reduce development time for accessibility. |

| **Disadvantages** | **Details** |
| --- | --- |
| **Cost Over Time** | API usage fees can be expensive. |
| **Latency Issues** | Each API request has network and processing delay which can delay inputs to the system. |
| **Privacy Concerns** | Sending data externally could risk security of sensitive data. |
| **Accuracy /**  **Hallucinogenic** | LLMs could be provide misleading data or destroy data integrity. |

Conclusion

Overall, the use of an external API for use in the retirement chat bot is promising as it provides. It provides customisation, can be low-cost and offers a quick development. The use of an API will be considered among our options and will be explored further in the future.

References

OpenAI. *Pricing*. OpenAI. Retrieved April 28, 2025, fro[m https://openai.com/api/pricing/](https://openai.com/api/pricing/)

DeepSeek. *Pricing*. DeepSeek. Retrieved April 28, 2025, fro[m https://apidocs.deepseek.com/quick\_start/pricing/](https://api-docs.deepseek.com/quick_start/pricing/)