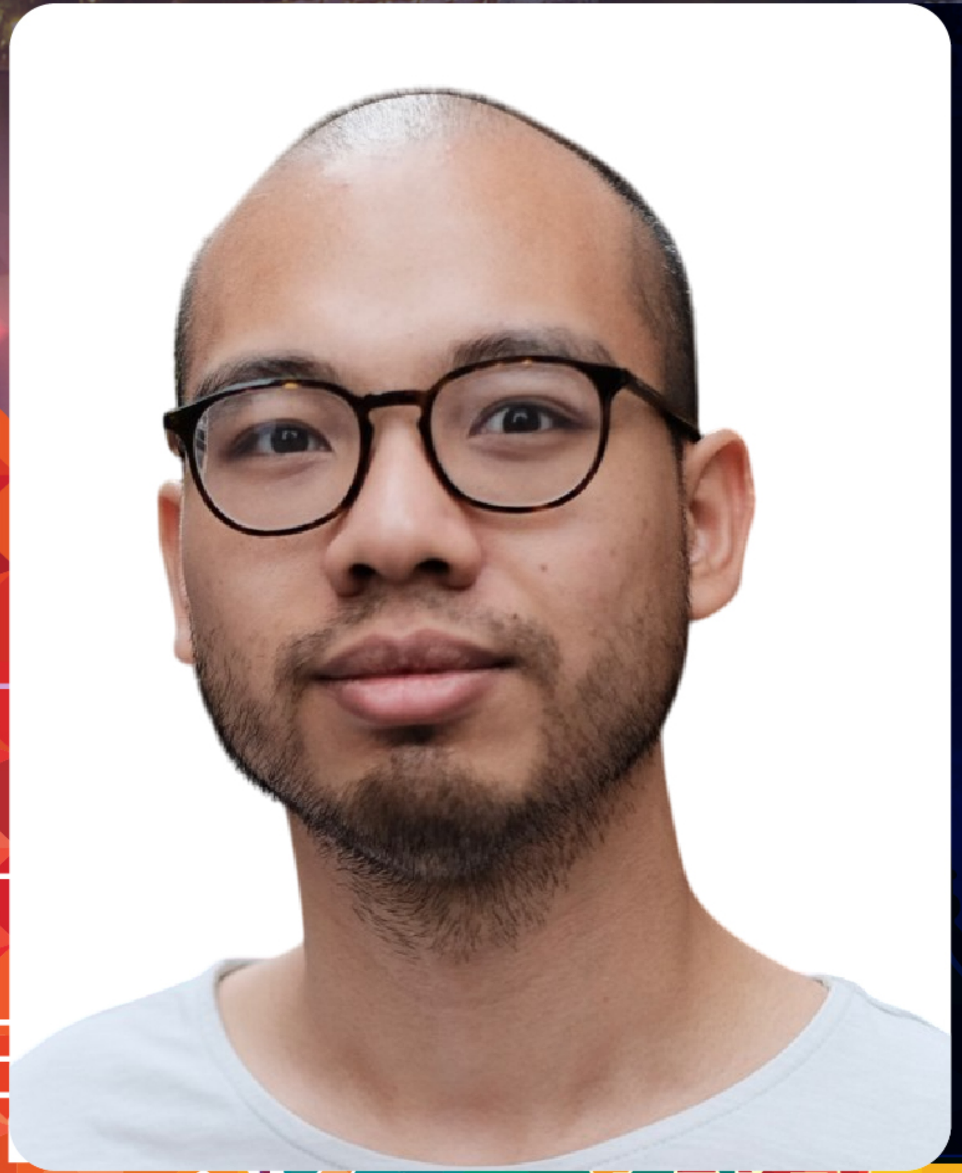
Asian Actuarial Conference 2025 Bangkok

Artificial Intelligence (AI) Agents in Asset and Liability Management (ALM)

12 Nov | 16.00-16.40



Duc Hien VU

Fellow of the Institut des Actuaires (France), Solvencii

Platinum sponsors

Gold sponsors

Silver sponsors

Exhibitor sponsors

Media & Supporting event partners

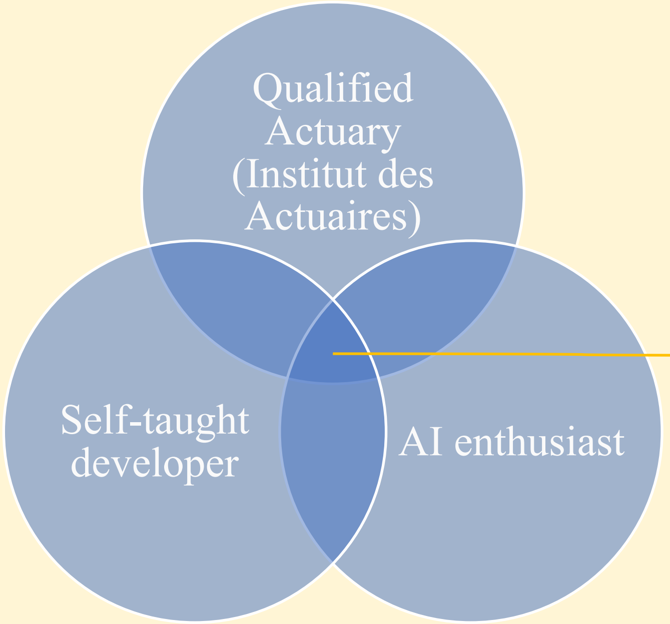
Asian Actuarial Conference 2025

Agentic AI in Asset and Liability Management (ALM)

Duc Hien VU

SOLVENCII | Founder & CEO

About Me



Me building Solvencii

I'm not alone. I have 4 AI assistants working full-time @**Solvencii Copilot**.

Reach out to us directly at: **solvencii.fr** !!!

Theoretical Foundations of LLMs

The chatbot era powered by LLMs

The launch of **ChatGPT** by OpenAI in **November 2022** marked the beginning…

Data collection and cleaning

- Raw data collection
- Preprocessing (remove noise, filter out irrelevant or harmful content)
- Standardize formats

Pre-training

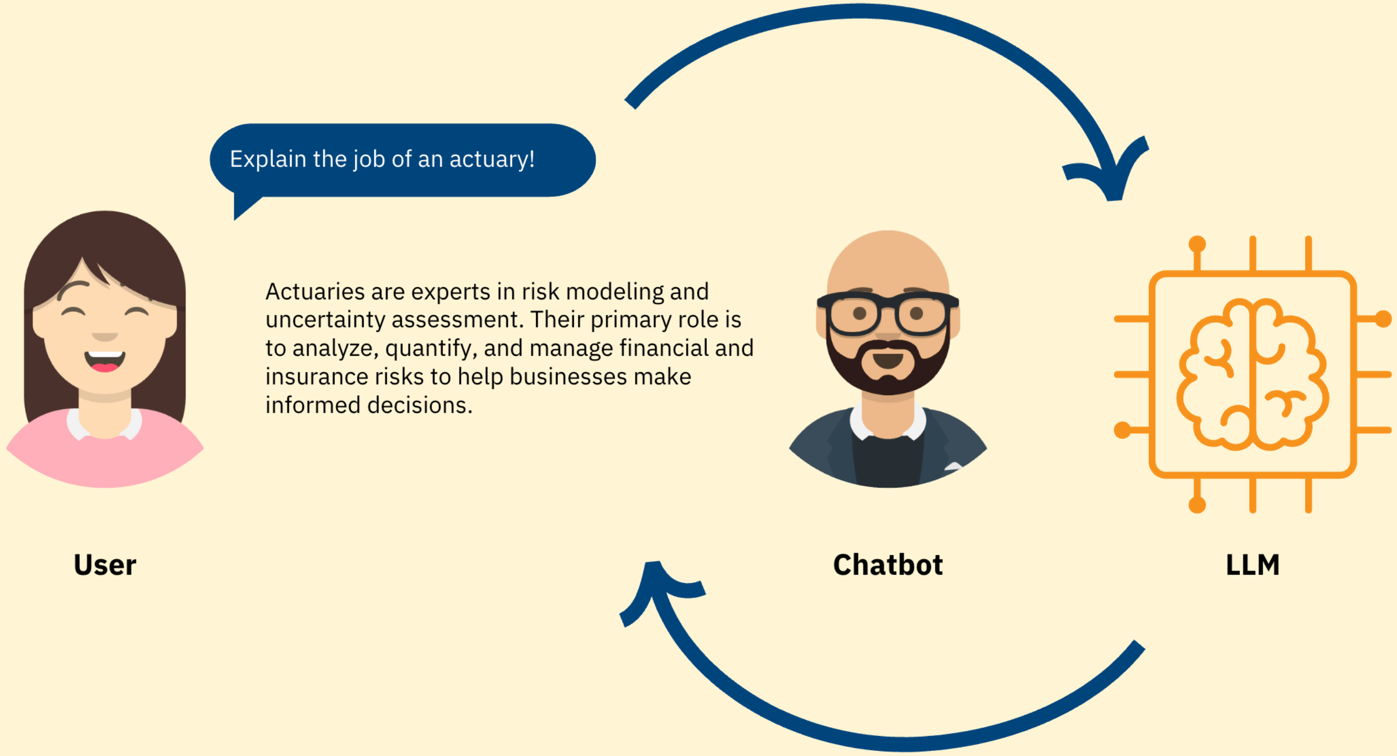
- Learn a foundation of linguistic knowledge in a self-supervised manner
- Massive computational resources and large-scale datasets

Fine-tuning

- Use labeled data to adapt the model to specific tasks or domains
- Supervised learning / Reinforcement Learning from Human Feedback

Post-training

- Prompt Engineering
- Temperature tuning
- Retrieval-augmented generation (RAG)
- Chain-of-thought reasoning

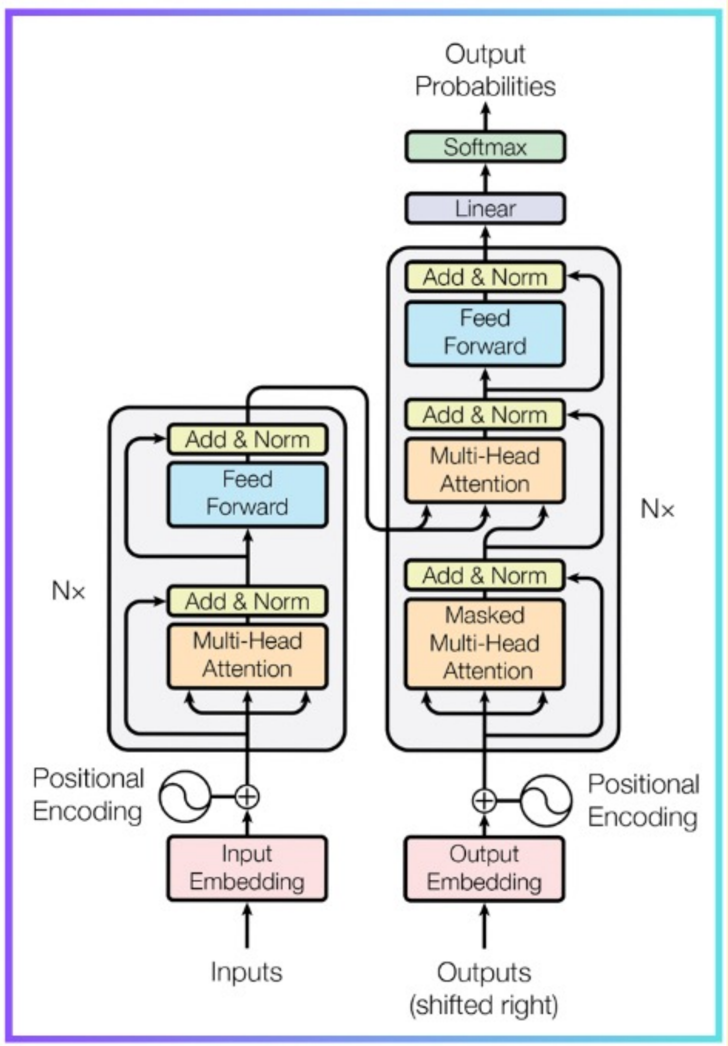


Transformer Architecture

A deep neural network framework that leverages self-attention mechanisms to process sequential data without relying on recurrent structures.

It consists of stacked encoder and decoder layers, where:

- each encoder layer computes self-attention over input tokens to capture contextual relationships, and
- each decoder layer performs both self-attention and encoder-decoder attention to generate outputs.

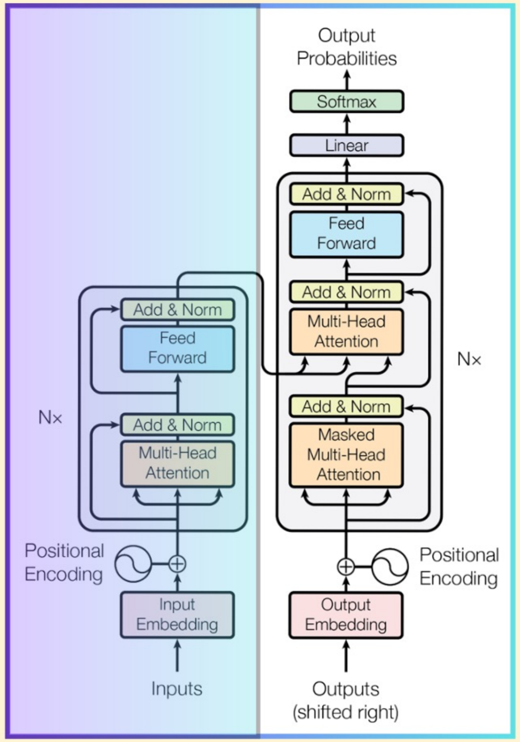


Attention is All You Need (Vaswani et al., 2017)

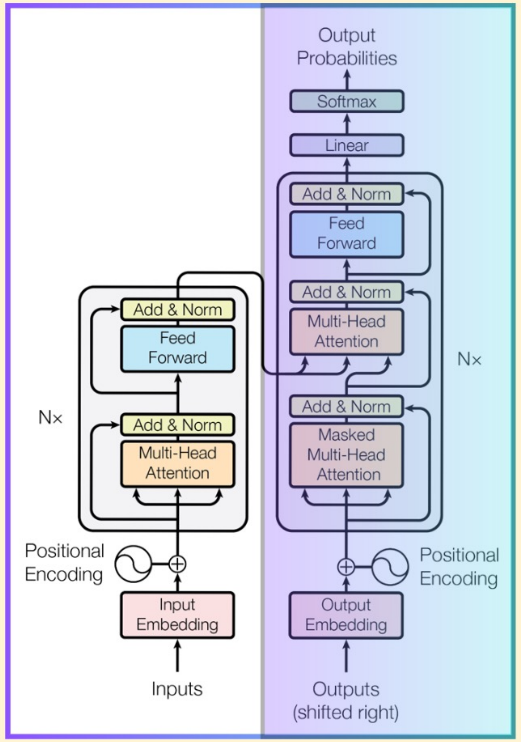
Variations of Transformer Architecture

- Encoder-only models focus on bidirectional context representation, ideal for understanding and classification tasks.
- Decoder-only models, are optimized for generative tasks by predicting subsequent tokens in a sequence, operating in a unidirectional manner.
- Encoder-decoder models combine both approaches to handle sequence-to-sequence tasks such as translation, summarization, and question answering
- Hybrid architectures that integrate elements of both self-attention and other neural structures have emerged.

encoder-only
(eg. BERT - Google)

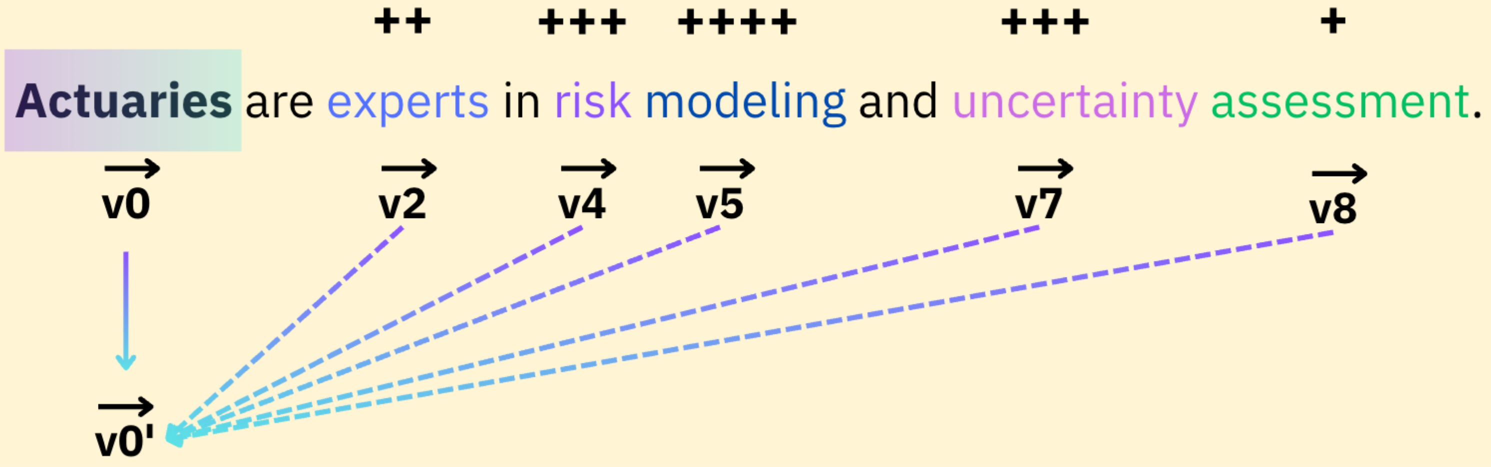


decoder-only
(eg. GPT - OpenAI)



Self-Attention mechanism

- Allows the model to capture the contextual meaning of the word **"Actuaries"** by attending to all other words in the sentence. First, the input words are converted into embeddings, which are projected into **query** Q, **key** K, and **value** V vectors.



- For the word "Actuaries" the model computes a similarity score between its query vector $Q_{Actuaries}$ and the key vectors of all other words in the sentence using a dot product: $score(Q_{Actuaries}, K_i)$. These scores are passed through the **softmax** function to calculate attention weights (how much focus to give to each word). The final respresentation of “Actuaries” is computed as a weighted sum of the value vectors.

Next Token Prediction

Tokenization breaks text into smaller, more manageable sub-word units

→reduces the overall vocabulary size

→allow the model to handle rare or out-of-vocabulary words more effectively

→captures morphological nuances that full words might miss.

For a sequence of tokens $t_1, t_2, \dots, t_n$, the **probability of a token $\boldsymbol{t_i}$ given its preceding context** is modeled as:

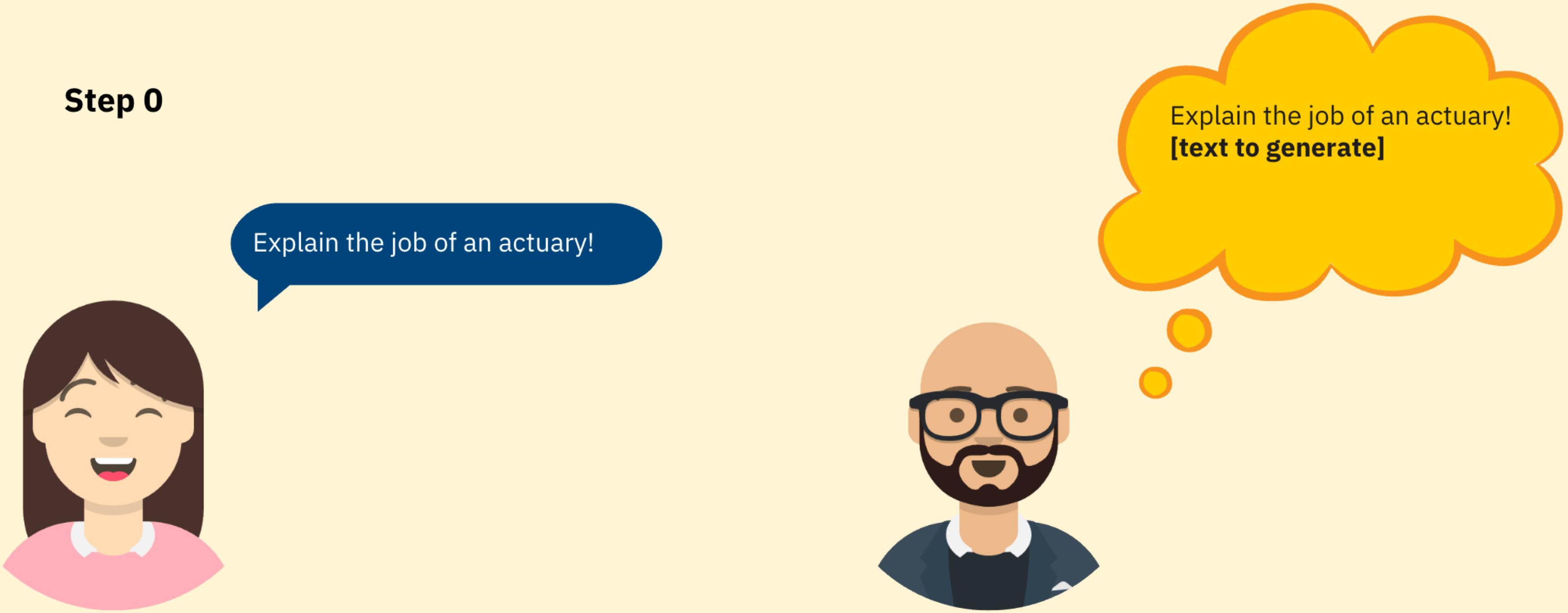
$$P(t_i | t_{1:i-1}) = \frac{\exp(score(t_{1:i-1}, t_i))}{\sum_{t' \in V} \exp(score(t_{1:i-1}, t'))}$$

where V is the vocabulary and $score(.)$ is a function learned by the network.

Each generated token is appended to the existing sequence and fed back into the model as part of the input context to predict the next token. This process is repeated iteratively until the model generates a stopping token or reaches the maximum token limit, completing the response.

Next Token Prediction

Step 0

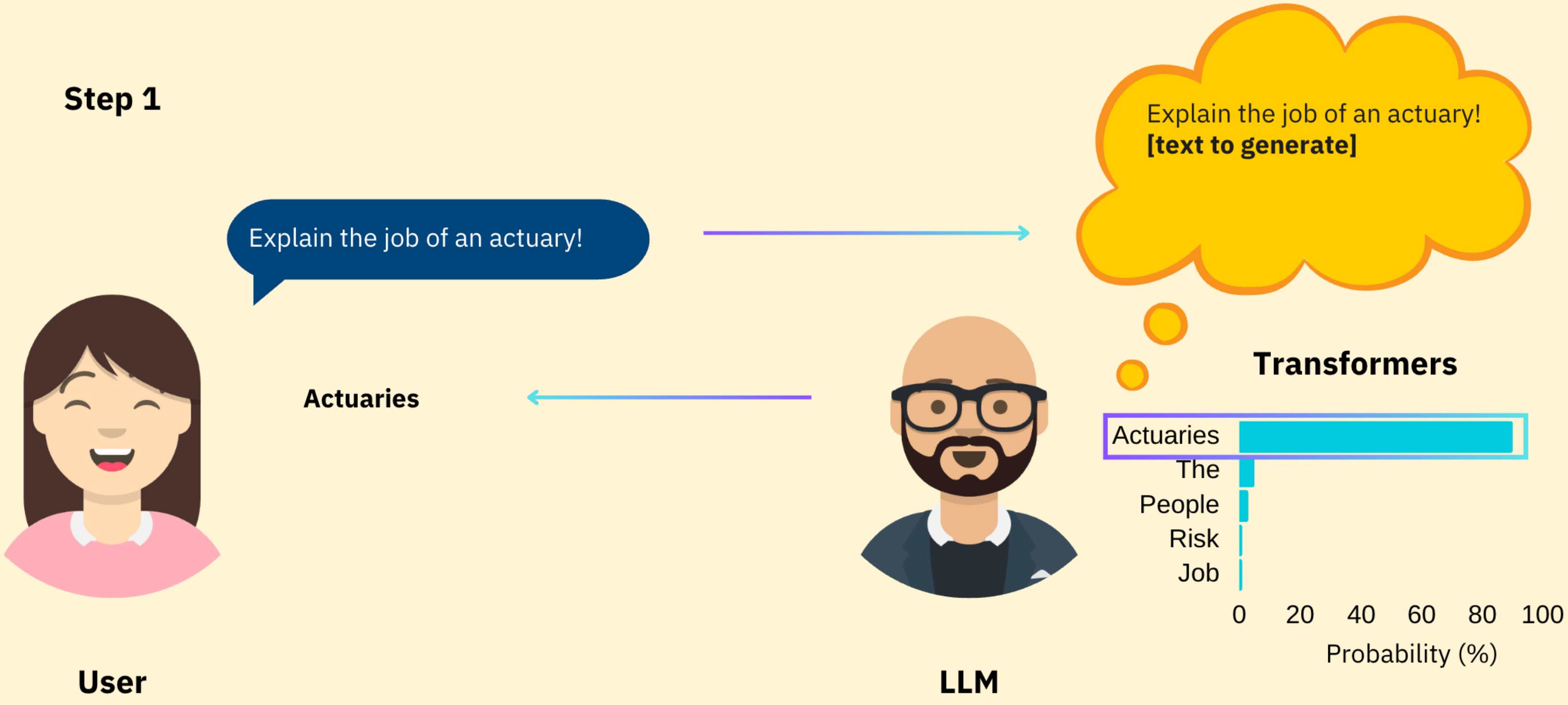


User

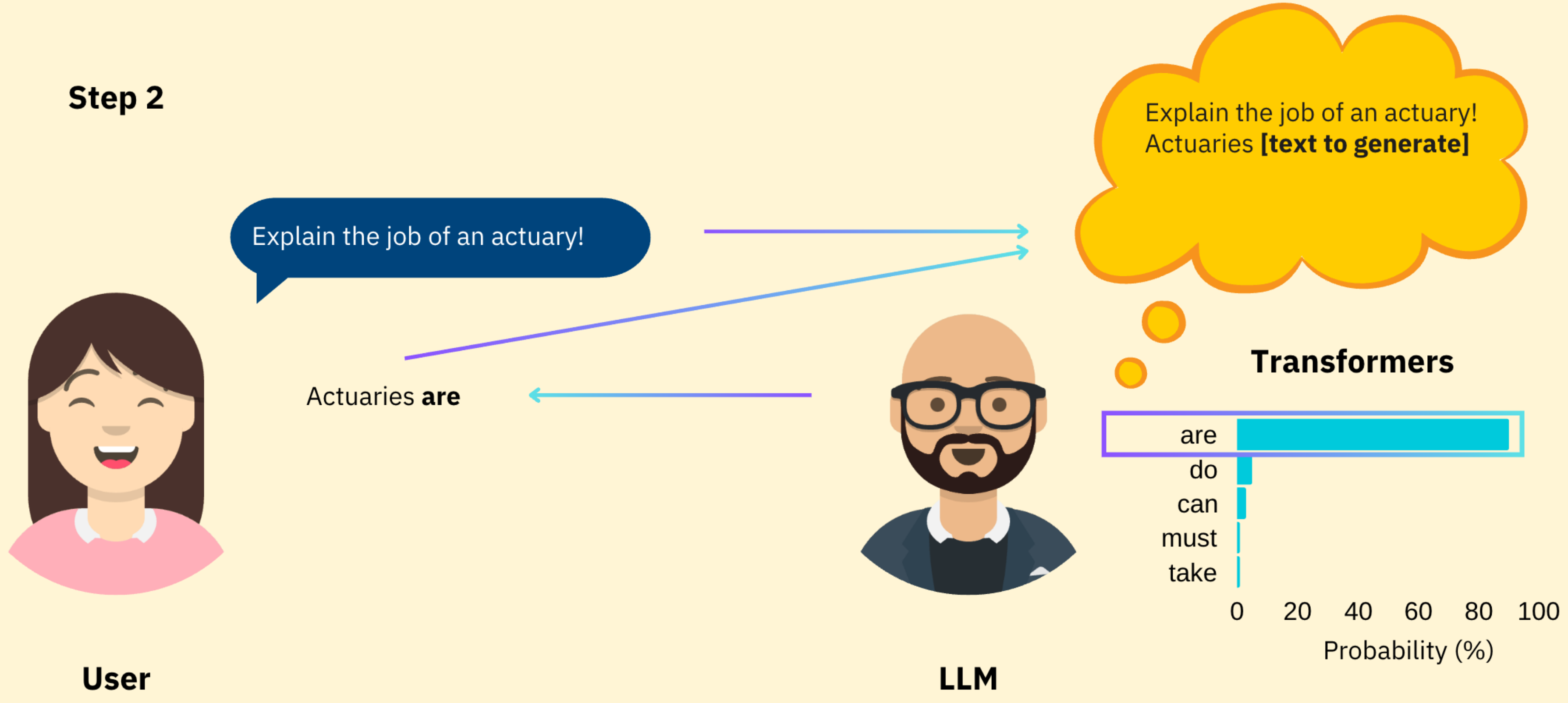
LLM

Next Token Prediction

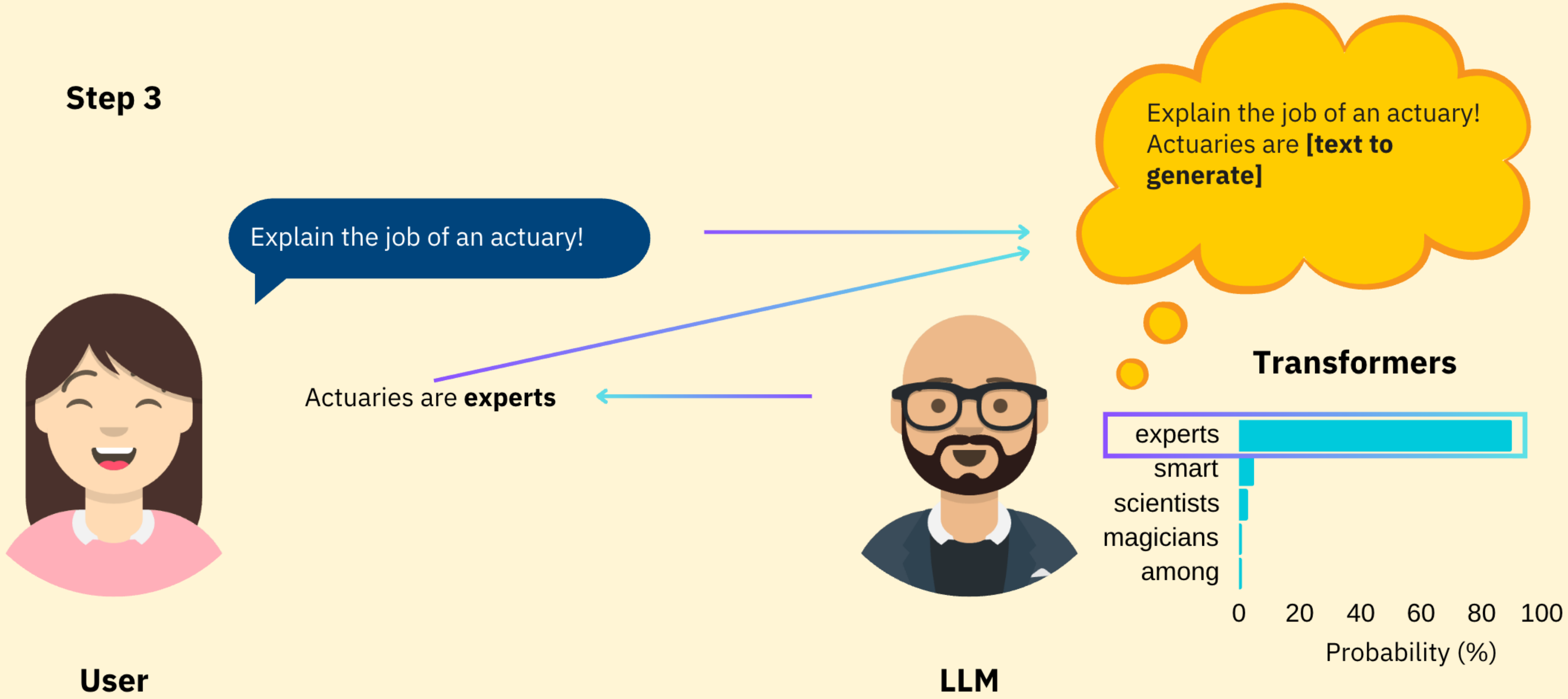
Step 1



Next Token Prediction



Next Token Prediction



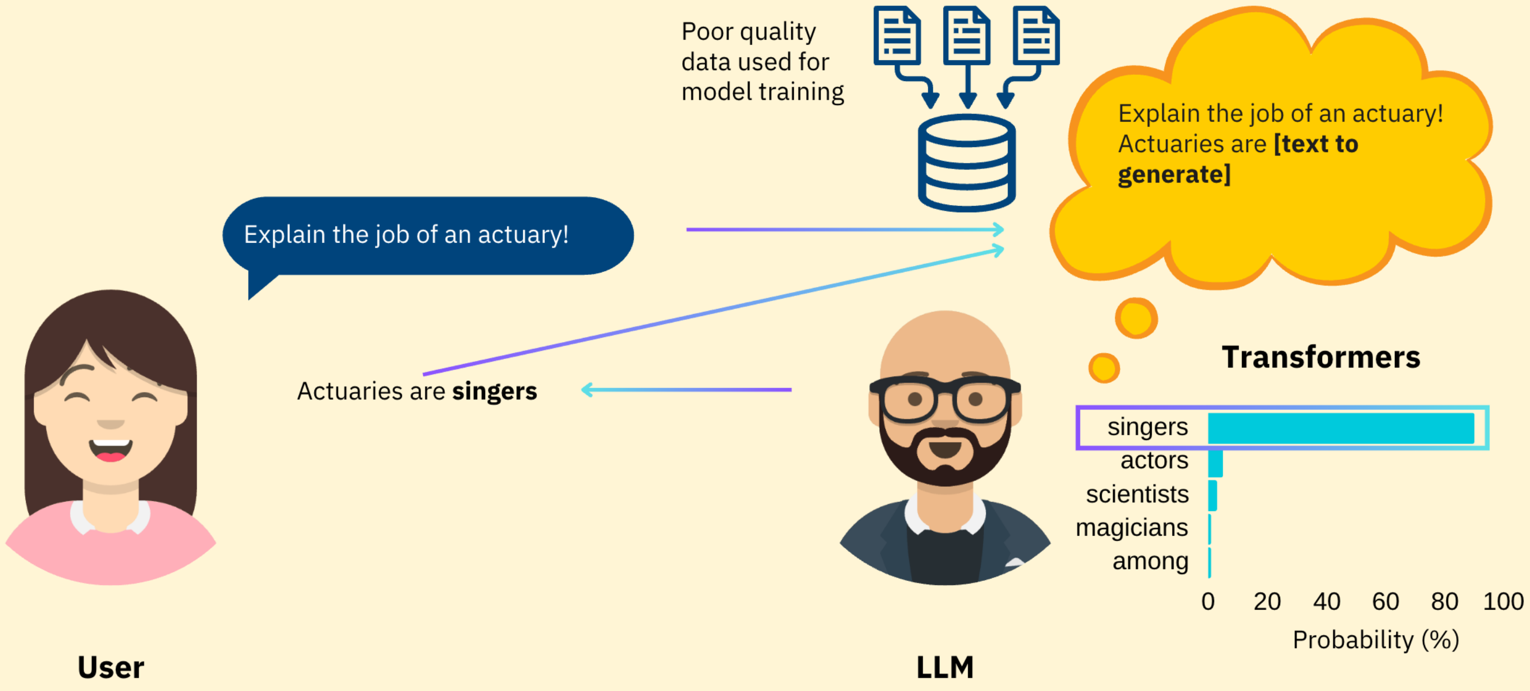
Hallucination

- **Hallucination** in large language models (LLMs) refers to the phenomenon where the model generates factually incorrect, misleading, or nonsensical information that appears plausible.

- This occurs because **LLMs are trained to predict the likely next token based on patterns in the training data** rather than verifying the factual accuracy of the output.
- Hallucination is particularly problematic in professional fields like finance and actuarial science, where accuracy and reliability are crucial.

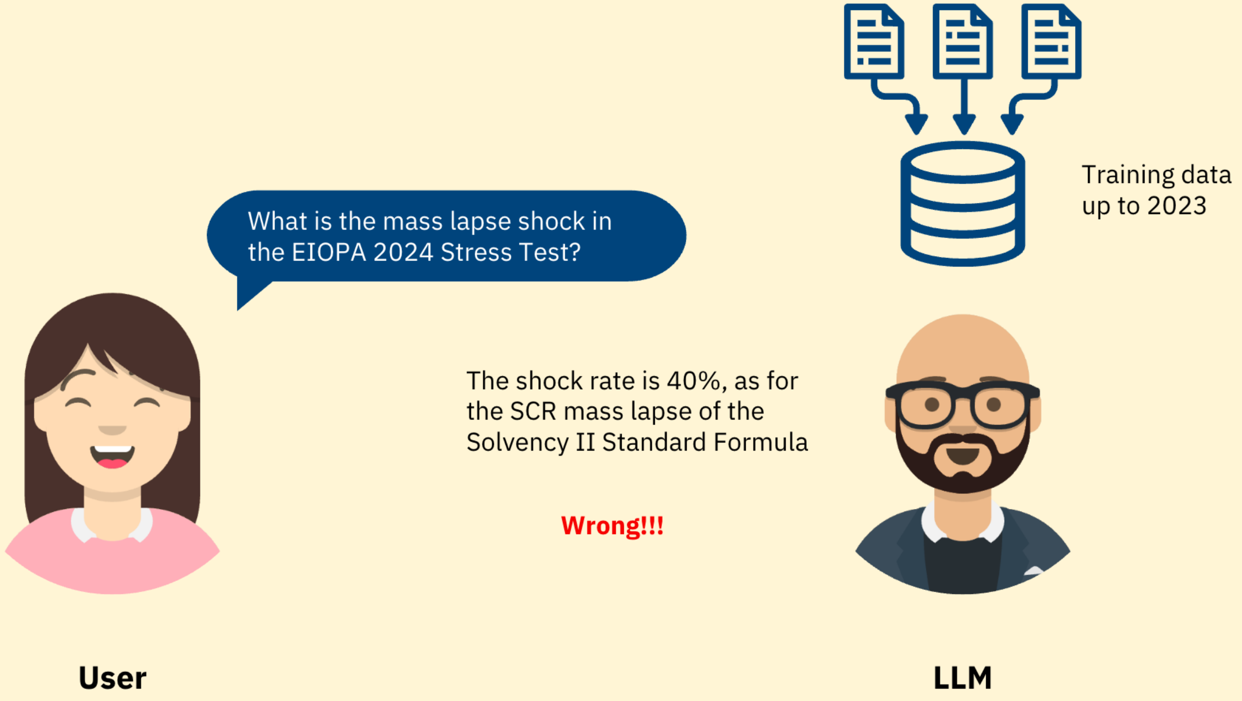
- The root causes of hallucination include **insufficient or biased training data, overgeneralization, lack of real-world grounding, and limitations in the model's ability to access or retrieve updated information**.

Hallucination



Hallucination due to poor quality data used for model training

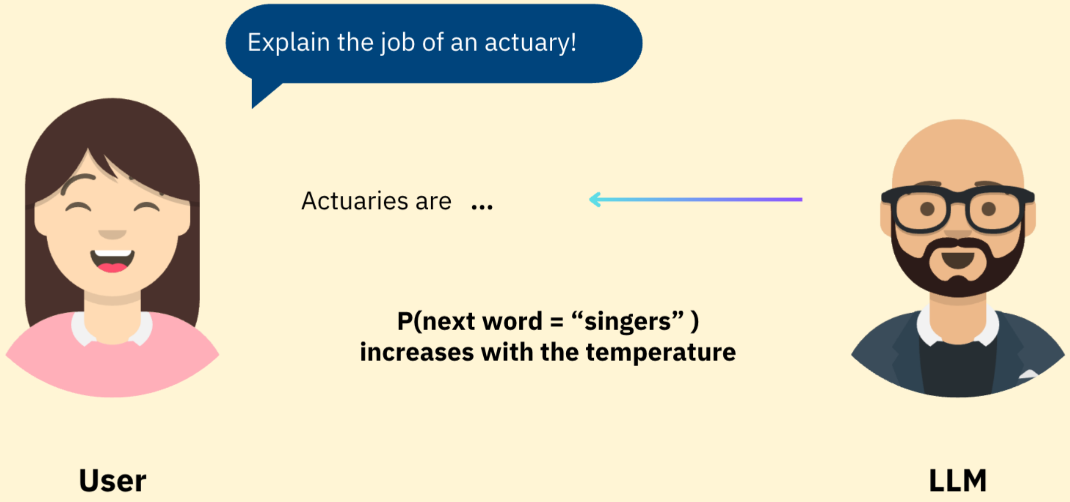
Hallucination

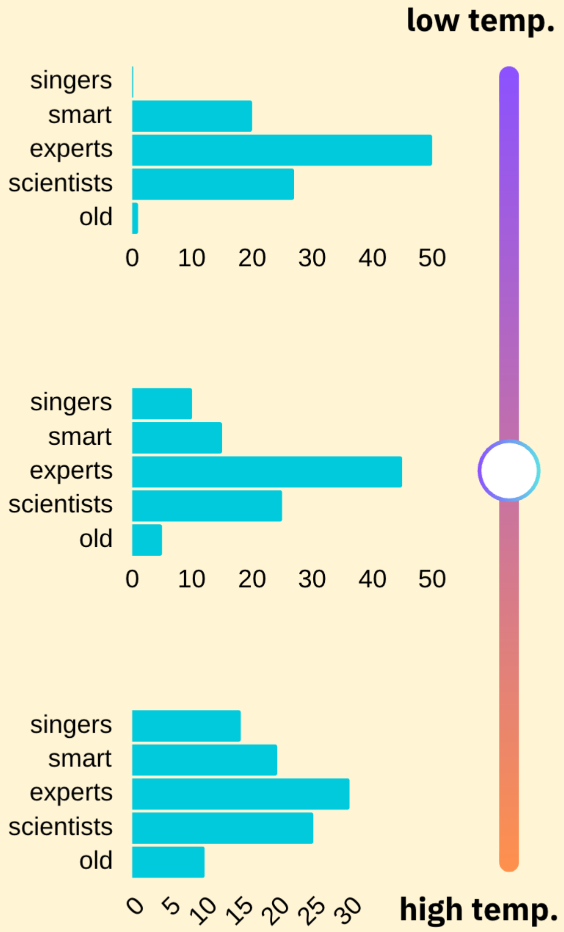


Hallucination due to limitations in the model's ability to access or retrieve updated information.

Temperature

Adjusting the temperature parameter can boost the quality of model outputs depending on the desired behavior.



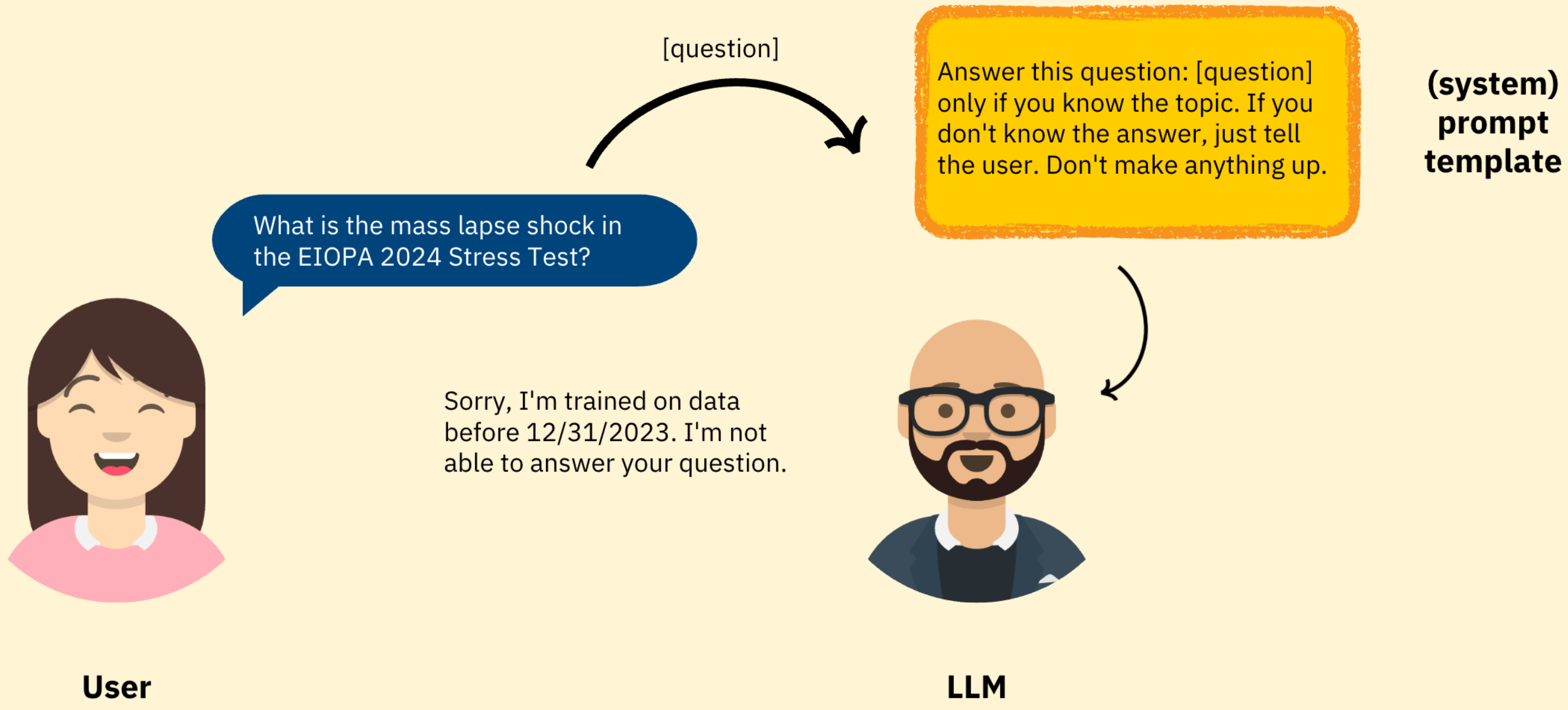


- A lower temperature sharpens the probability distribution, making the model more confident in its highest probability tokens and thus more deterministic in its output.
- A higher temperature flattens the distribution, leading to more diverse and random outputs.

Prompt Engineering

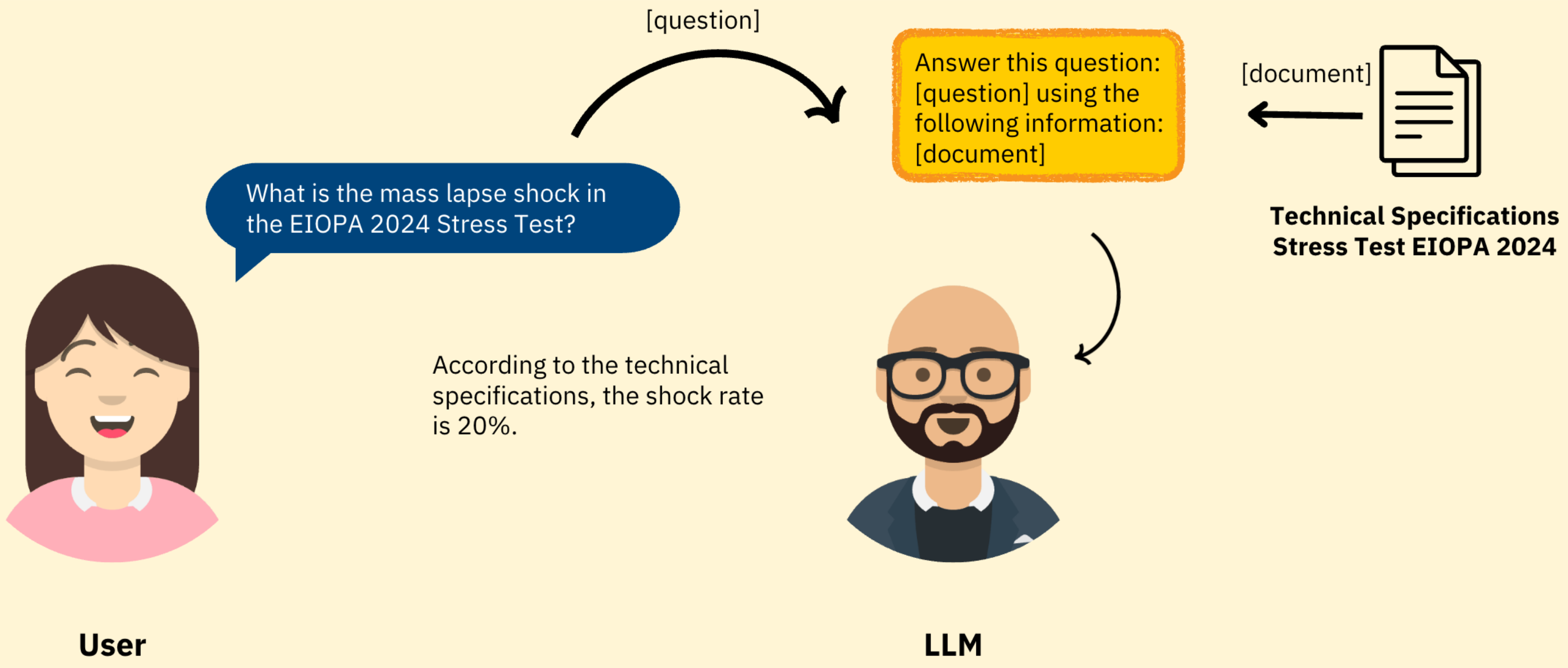
A common and effective way to reduce model hallucination is through **prompt engineering**. When a user submits a question, the input prompt is combined with a **system prompt template** that includes specific instructions designed to guide the model's behavior and improve response accuracy.

Returning to the example of hallucination caused by the model's limited access to updated information, one straightforward approach is to **explicitly instruct the model not to generate information it is unsure about**.



Augmented Data & Context Window

If we still want the LLM to provide a meaningful answer despite its limited internal knowledge, one approach is to **inject augmented data into the system prompt** to give the LLM a more complete context.

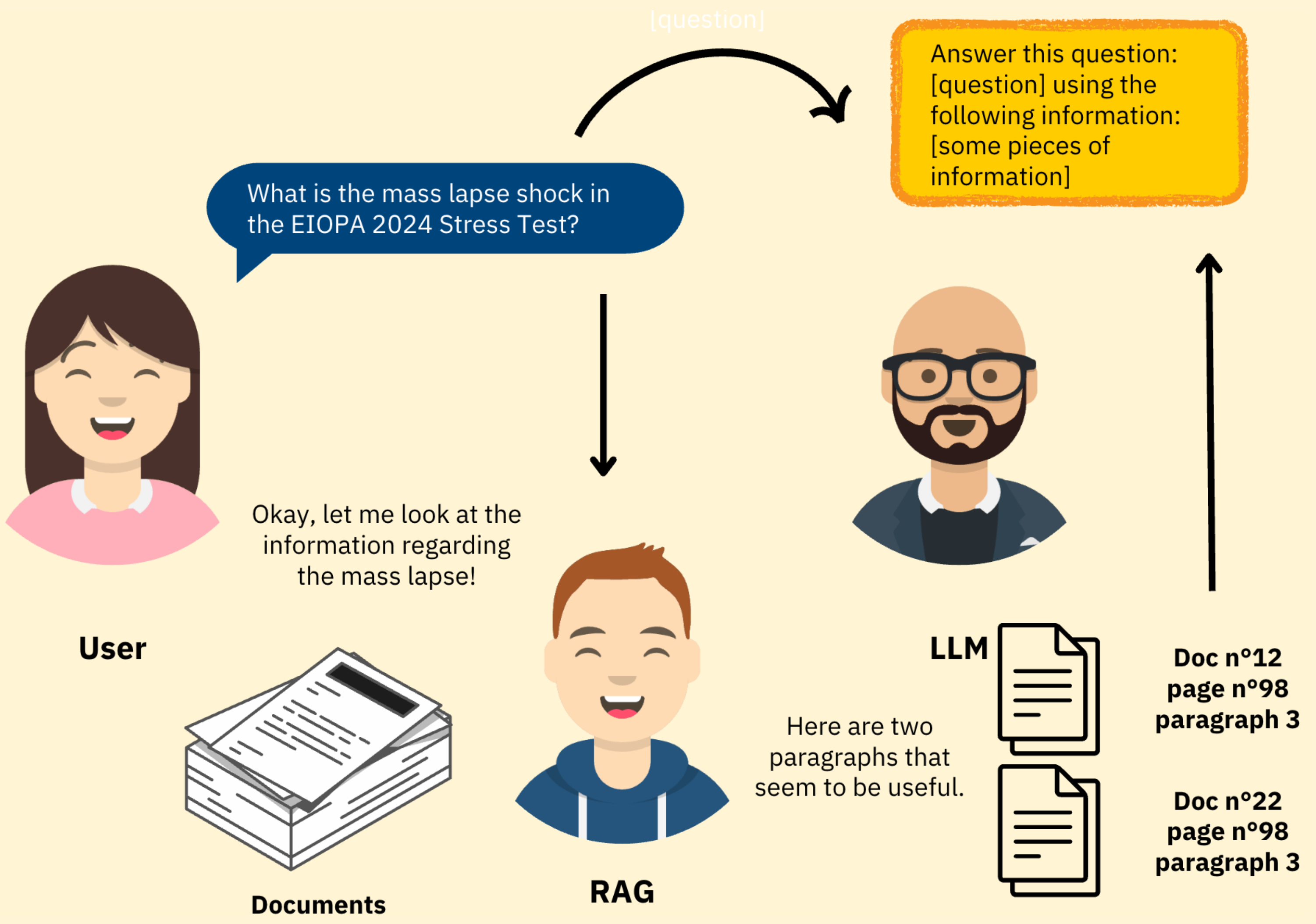


However, this is technically challenging because augmented data often resides in large documents, and the **context window** - which defines the maximum length of the prompt - is limited.

Retrieval-Augmented Generation (RAG)

Even though context windows are expanding, filling the prompt with large amounts of raw data is often inefficient and costly.

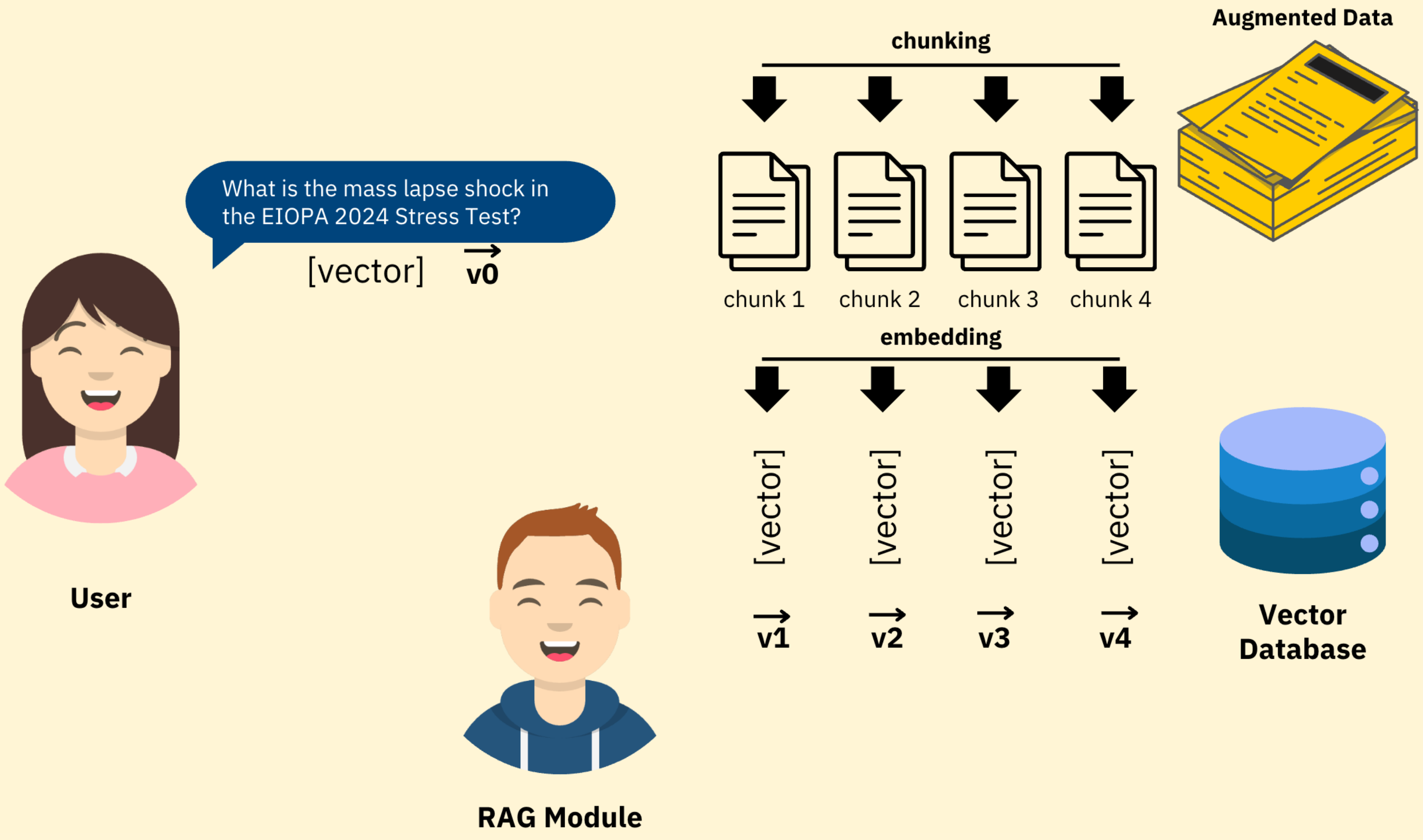
A smarter and more cost-effective solution is to **extract only the most relevant and useful pieces of information** and supply them to the model as context.



Retrieval-Augmented Generation (RAG)

To enable efficient searching, text data is first converted into numerical representations called **embeddings**.

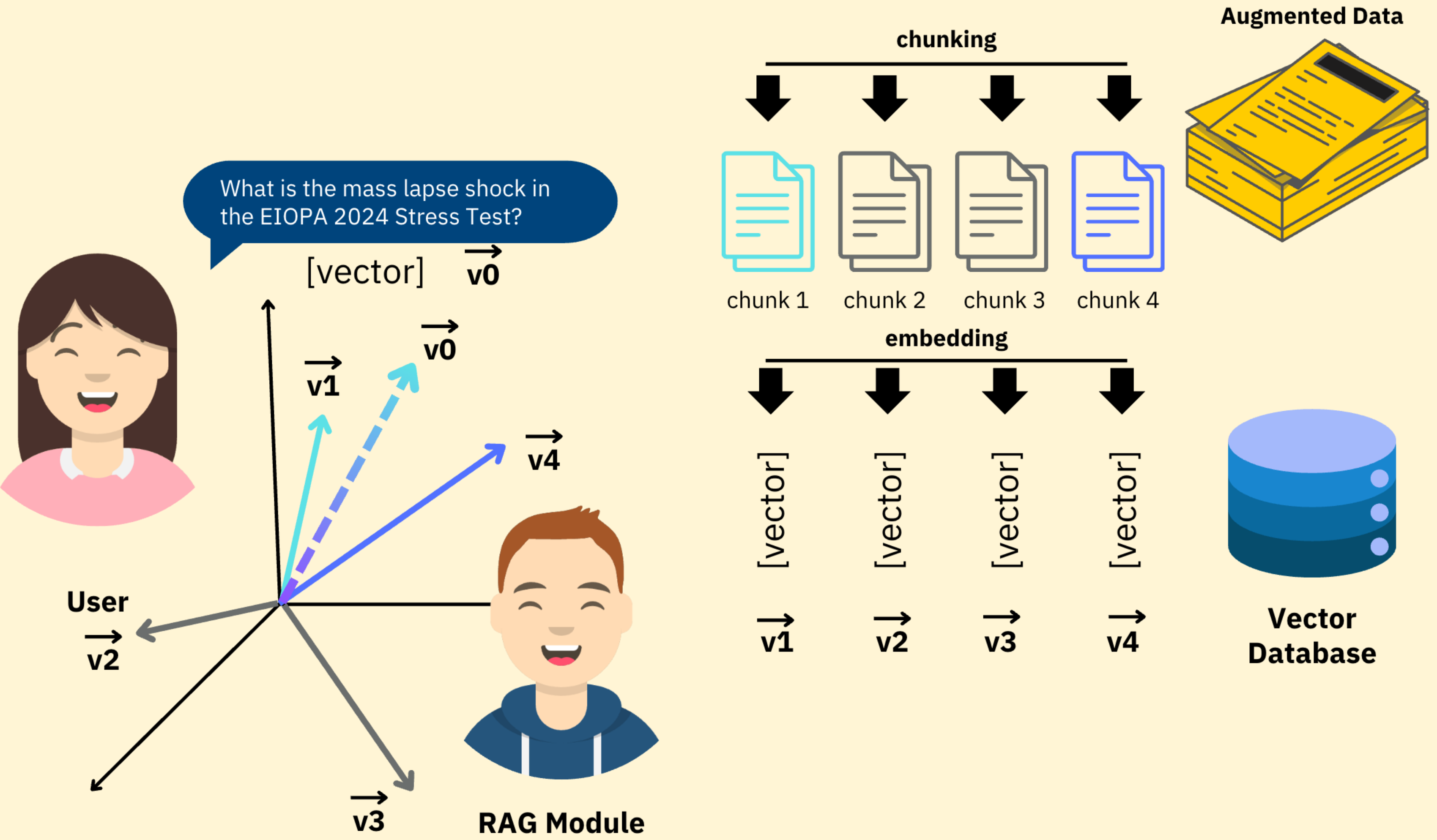
A vector database stores these embeddings in a structured format, allowing for efficient similarity-based searches.



Vector Search

When the user submits a query, it is embedded into a vector using the same embedding model.

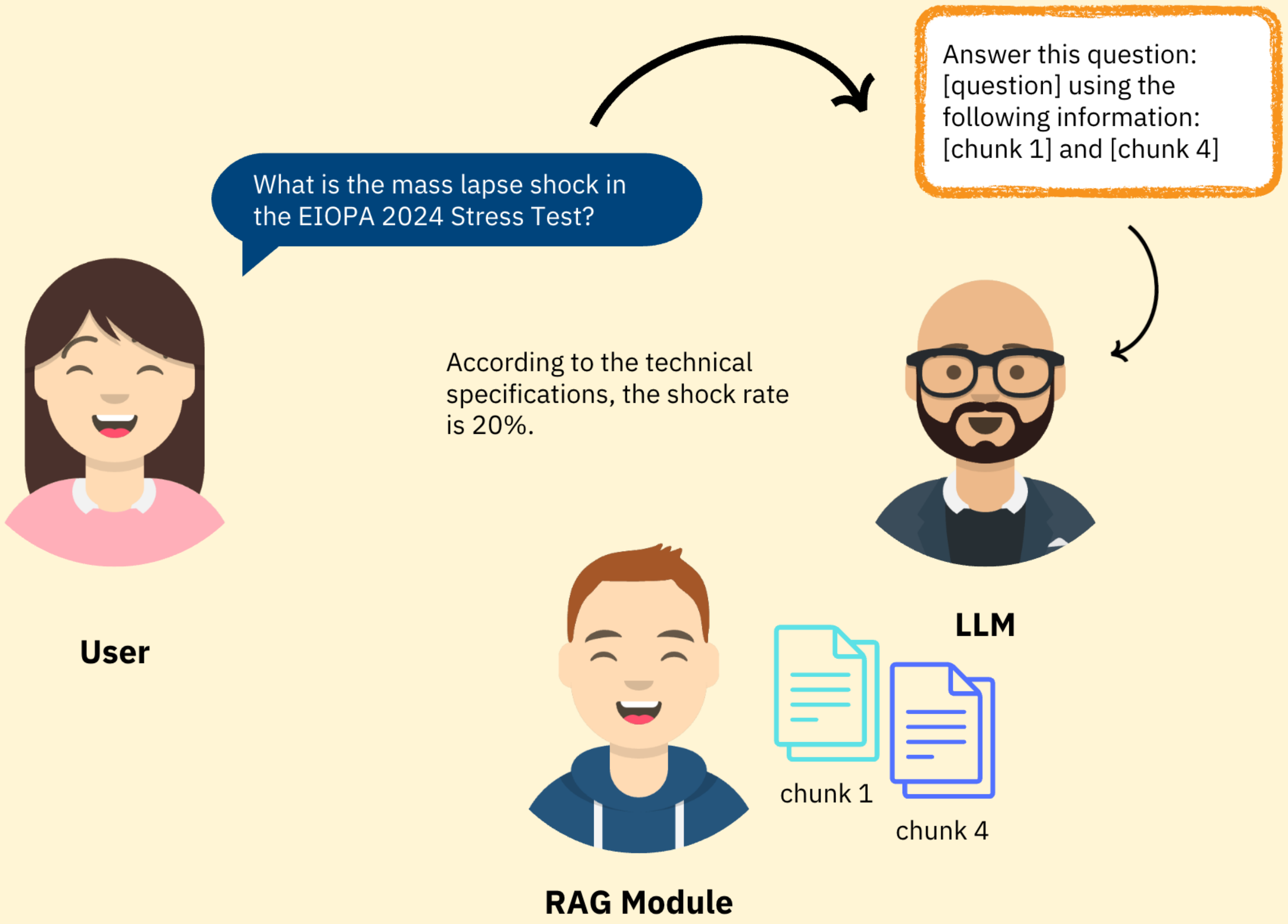
The system performs a **vector search** to find the most similar vectors in the database using a distance metric like **cosine similarity** or **Euclidean distance**.



Retrieval-Augmented Generation (RAG)

The closest matches (most semantically similar) are retrieved and passed to the LLM as part of the prompt context.

More advanced techniques are under development and testing to enhance RAG's performance and accuracy, such as **Graph RAG** and other hybrid retrieval approaches.



Chain-of-Thought (CoT)

Example: **Prompt-based few-shot CoT (Inference-time technique)** → guide the model to think step-by-step by showing, in the system prompt, examples of step-by-step reasoning before the question.

By showing **structured examples**, we:

- Teach it **how to break down** the actual logic (mortality probabilities, discounting, summation).
- Implicitly define **the reasoning style**: short, precise, and actuarial.
- Encourage **step-by-step numeric reasoning**, reducing arithmetic or conceptual errors.

User

A 3-year term insurance of 100,000 is issued to a life aged 40.
q_{40} = 0.002, q_{41} = 0.0025, q_{42} = 0.003, and i = 2%.
What is the EPV of benefits?

A: Let's think step by step.
1. Benefit amount = 100,000
2. Death can occur in year 1, 2, or 3.
3. Probabilities:
 - Year 1: q_{40} = 0.002
 - Year 2: $(1 - q_{40}) \times q_{41} = 0.998 \times 0.0025 = 0.002495$
 - Year 3: $(1 - q_{40})(1 - q_{41}) \times q_{42} = 0.998 \times 0.9975 \times 0.003 = 0.002985$
4. Discount factors:
 - $v^1 = 0.98039$, $v^2 = 0.96117$, $v^3 = 0.94232$
5. EPV = $100{,}000 \times [0.002\times0.98039 + 0.002495\times0.96117 + 0.002985\times0.94232]$
6. EPV = $100{,}000 \times 0.006346 = 634.6$
Answer: The EPV is approximately 634.6.

LLM

[question]

You are an actuary. Compute the expected present value (EPV) of benefits for each case.
Show your reasoning step by step before giving the final answer.

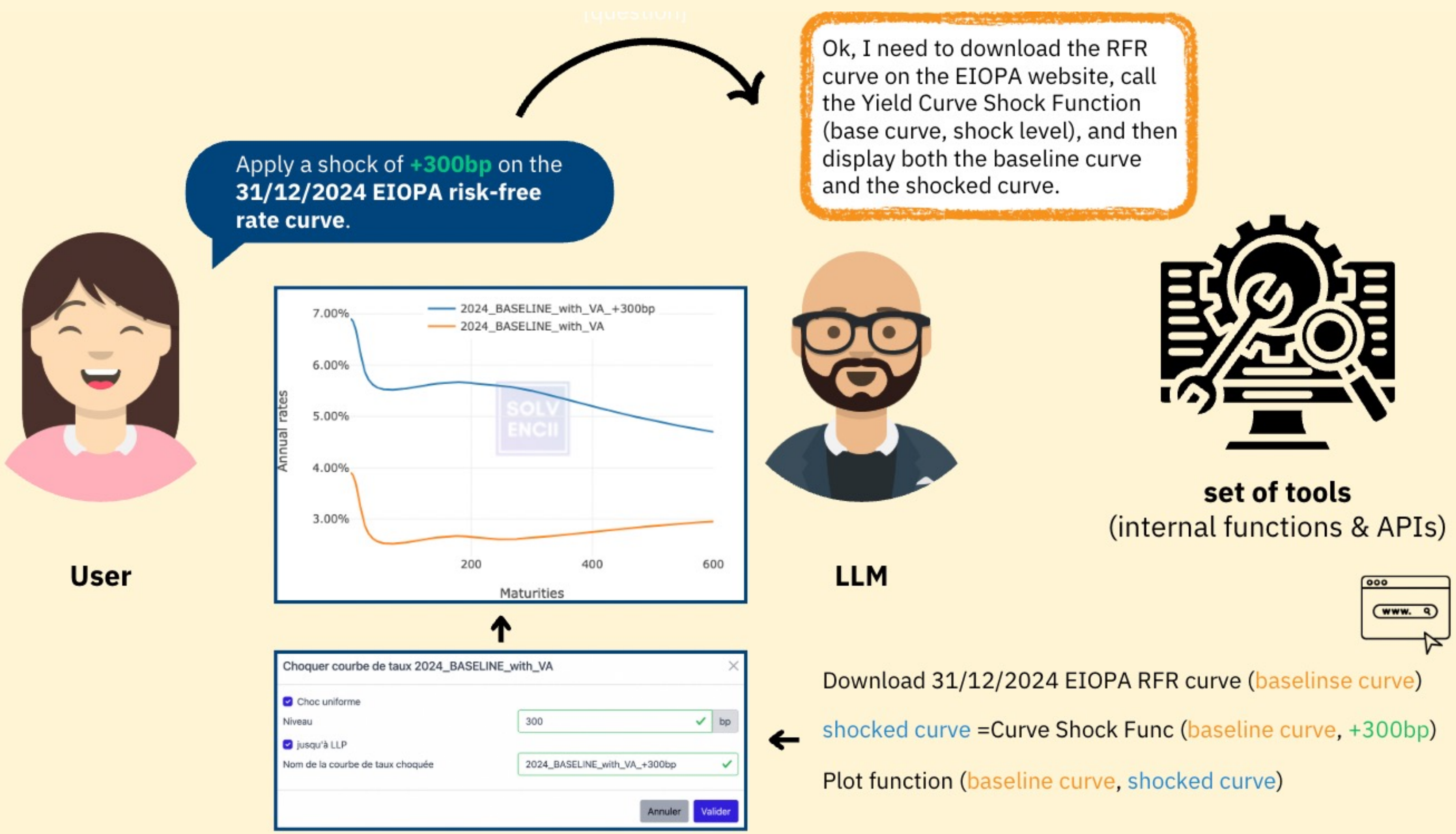
Example 1:
Q: A 1-year term insurance of 100,000 is issued to a life aged 40.
The annual mortality rate q_{40} = 0.002 and interest rate i = 2%.
What is the EPV of benefits?

A: Let's think step by step.
1. Benefit amount = 100,000
2. Probability of death within the year = q_{40} = 0.002
3. Present value factor = 1 / (1 + i) = 1 / 1.02 = 0.98039
4. EPV = $100{,}000 \times 0.002 \times 0.98039 = 196.08$
Answer: The EPV is approximately 196.08.

Example 2: ...

Now answer this question in the same reasoning style:
Q: [question]

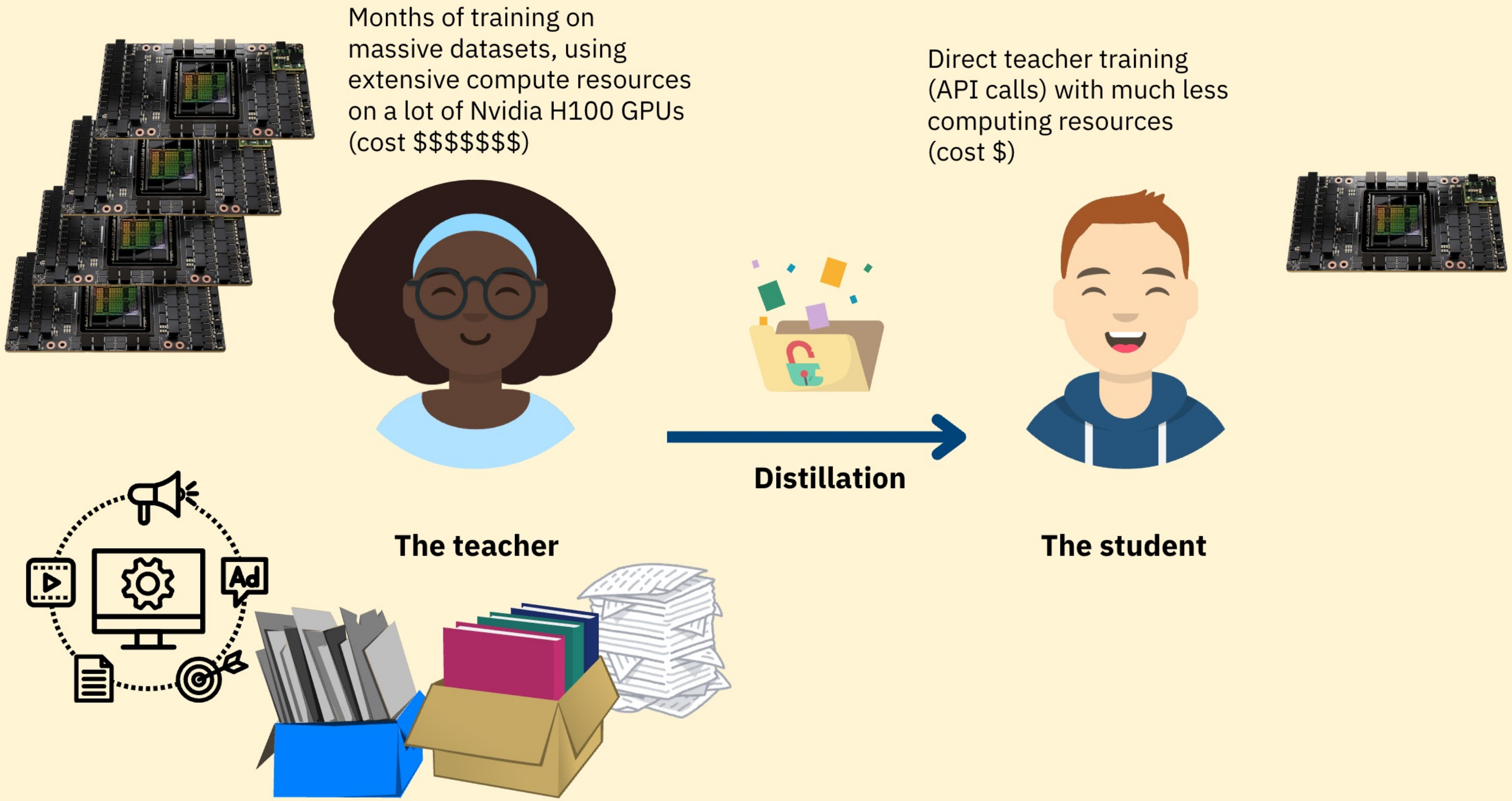
Tool Use


Apply a shock of +300bp on the 31/12/2024 EIOPA risk-free rate curve.
Ok, I need to download the RFR curve on the EIOPA website, call the Yield Curve Shock Function (base curve, shock level), and then display both the baseline curve and the shocked curve.
User
LLM
set of tools
(internal functions & APIs)
2024_BASELINE_with_VA_+300bp
2024_BASELINE_with_VA
Annual rates
7.00%
6.00%
5.00%
4.00%
3.00%
200
400
600
Maturities
Choquer courbe de taux 2024_BASELINE_with_VA
Choc uniforme
Niveau
300
bp
jusqu'à LLP
Nom de la courbe de taux choquée
2024_BASELINE_with_VA_+300bp
Annuler
Valider
www.
Download 31/12/2024 EIOPA RFR curve (baselinse curve)
shocked curve =Curve Shock Func (baseline curve, +300bp)
Plot function (baseline curve, shocked curve)

Distillation

A model compression technique where a smaller, more efficient model (the *student*) is trained to replicate the performance of a larger, more complex model (the *teacher*).

Train the smaller model to match the teacher's output probabilities, hidden state representations, or attention patterns.
→ reduce the computational and memory requirements of LLMs while preserving most of the teacher model's accuracy and reasoning capabilities.



AI Agents

System that can perceive its environment, process information, and take actions to achieve specific goals, often with a degree of autonomy and adaptability.

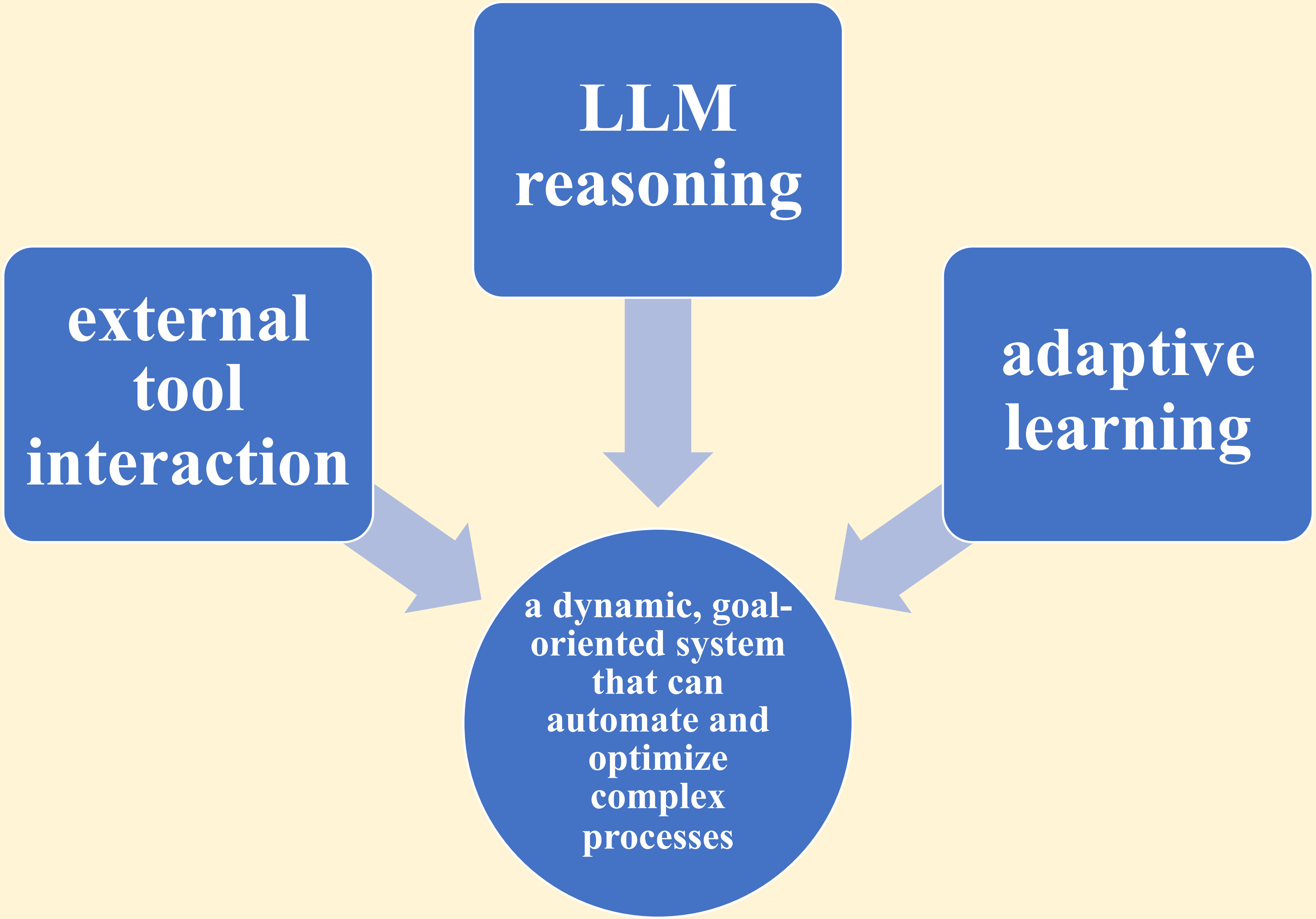
An AI agent typically consists of the following core components:

- **Perception** – The agent receives input from its environment, which could be user prompts, external data feeds, or system-generated signals.
- **Memory/State** – The agent maintains a state that tracks the current context and past interactions to inform future decisions.
- **Reasoning/Decision-Making** – The agent processes the input using LLMs or other models to evaluate possible actions and determine the best course of action.
- **Action/Execution** – The agent executes the chosen action, which could involve generating text, retrieving information, running calculations, or triggering an external process.
- **Feedback Loop** – The agent evaluates the outcome of its action and adjusts its future decisions based on the results, creating a learning loop.

Building AI Agents

- **Define the Objective and Scope**
  - Identify the specific problem the agent should solve (e.g., interpreting ALM results, suggesting asset allocation strategies, etc.).
  - Set boundaries for the agent's behavior and capabilities (e.g., which data sources it can access, what decisions it can make).
- **Select the Core Model**
  - Choose an appropriate LLM (e.g., GPT, LLaMA) or fine-tune an existing model to suit the task.
  - Pre-train or fine-tune the model on domain-specific data (e.g., financial models, market data).
- **Build the Reasoning Engine**
  - Implement **chain-of-thought** reasoning to allow the agent to break down complex problems.
  - Use **tool-calling** and **function-calling** to allow the agent to interact with external systems (e.g., financial models, APIs).
  - Employ **planning** algorithms (e.g., tree search, reinforcement learning) to enable multi-step decision-making.
- **Integrate a Retrieval System**
  - Implement **RAG (Retrieval-Augmented Generation)** to allow the agent to access real-time market data, financial reports, and other external sources.
  - Fine-tune the retriever to optimize relevance and minimize noise.
- **Create a Feedback and Learning Loop**
  - Apply **Reinforcement Learning from Human Feedback (RLHF)** to refine outputs based on user evaluations.
  - Monitor the agent's performance and retrain periodically to adapt to changing market conditions or user needs.
- **Build an Interface**
  - Develop a user-friendly interface (e.g., chatbot or web app) to allow users to interact with the agent.
  - Enable multi-turn conversation handling to maintain context across user interactions.

An effective AI Agent

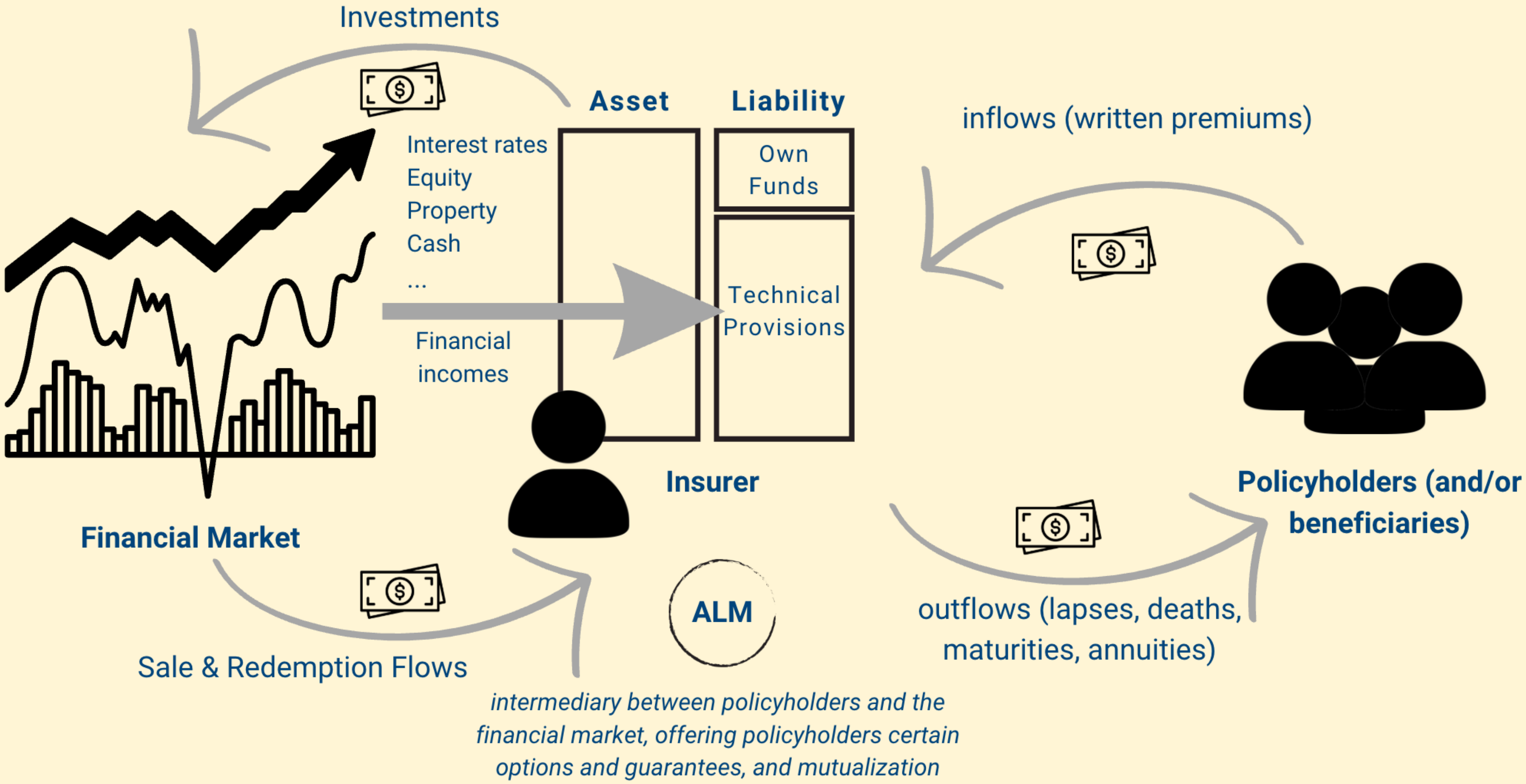


Asset and Liability Management

ALM modeling & ALM software

Asset and Liability Management (ALM)

A core discipline within actuarial science that focuses on managing the interaction between an institution's assets and liabilities to optimize financial performance, minimize risk, and ensure long-term solvency.

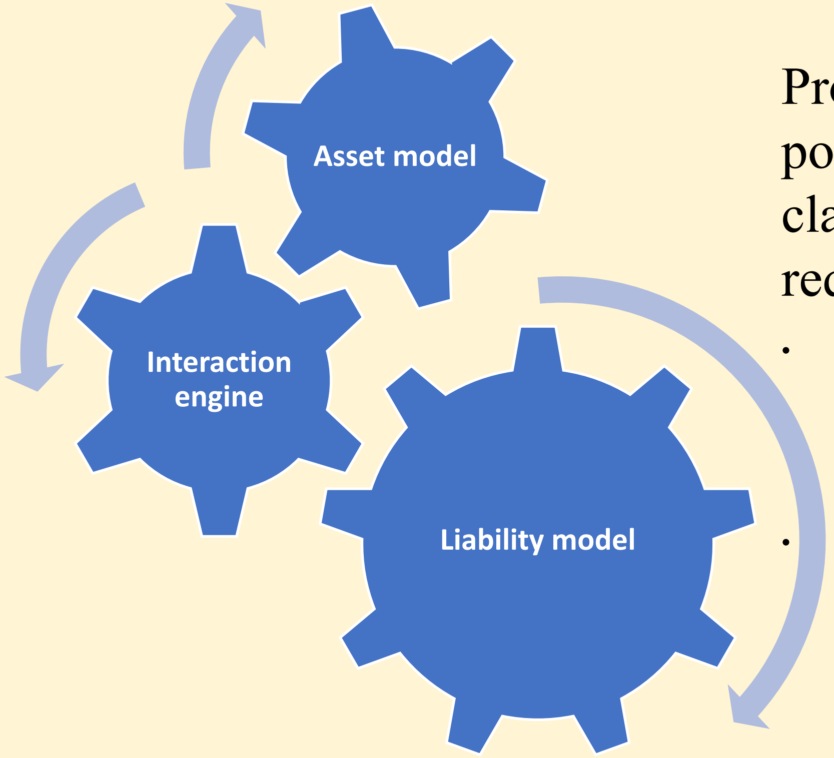


ALM Model Structure

Connects asset and liability models to simulate how assets and liabilities evolve together over time under different market scenarios.

- Applies dynamic asset allocation rules.
- Models reinvestment and disinvestment strategies.
- Simulates the impact
- of market shocks and
- economic cycles
- Models crediting and profit-sharing strategies

Models the behavior of financial markets, asset returns, interest rates, and credit spreads.
Uses stochastic models to simulate market dynamics.
Incorporates historical correlations between asset classes



Projects future cash outflows based on policyholder behavior, mortality rates, claims experience, and regulatory requirements.

- Uses actuarial models (e.g., survival models, annuity factors) to simulate liability cash flows.
- Accounts for optionality in insurance contracts (e.g., policyholder lapses, surrender options).

Challenges in traditional ALM software

- **Data Preparation and Input Complexity:** ALM models require the ingestion of diverse datasets—from market indices and interest rates to demographic statistics and policy details. Standardizing these data sources into a model-ready format is nontrivial and often requires manual intervention, increasing the potential for errors.

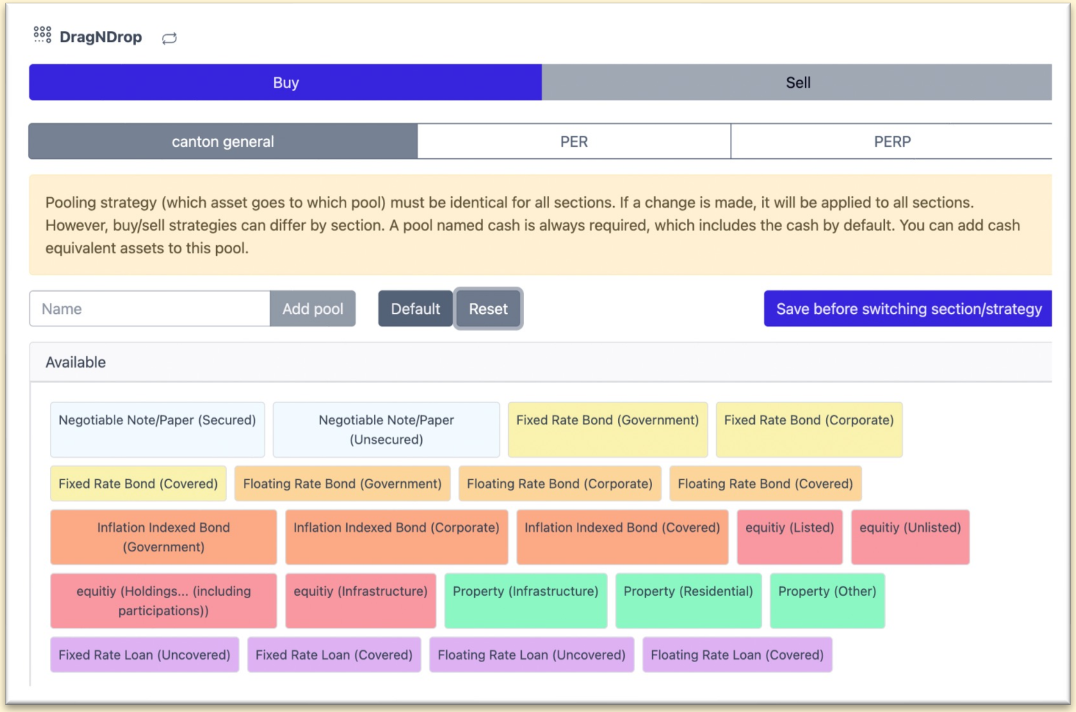
- **Algorithmic Sophistication in ALM:** ALM algorithms involve complex processes such as asset allocation, investment strategies, crediting methodologies, and profit-sharing mechanisms. Each component demands precise calibration and robust mathematical modeling, which can overwhelm traditional computational systems, especially during extensive scenario simulations.

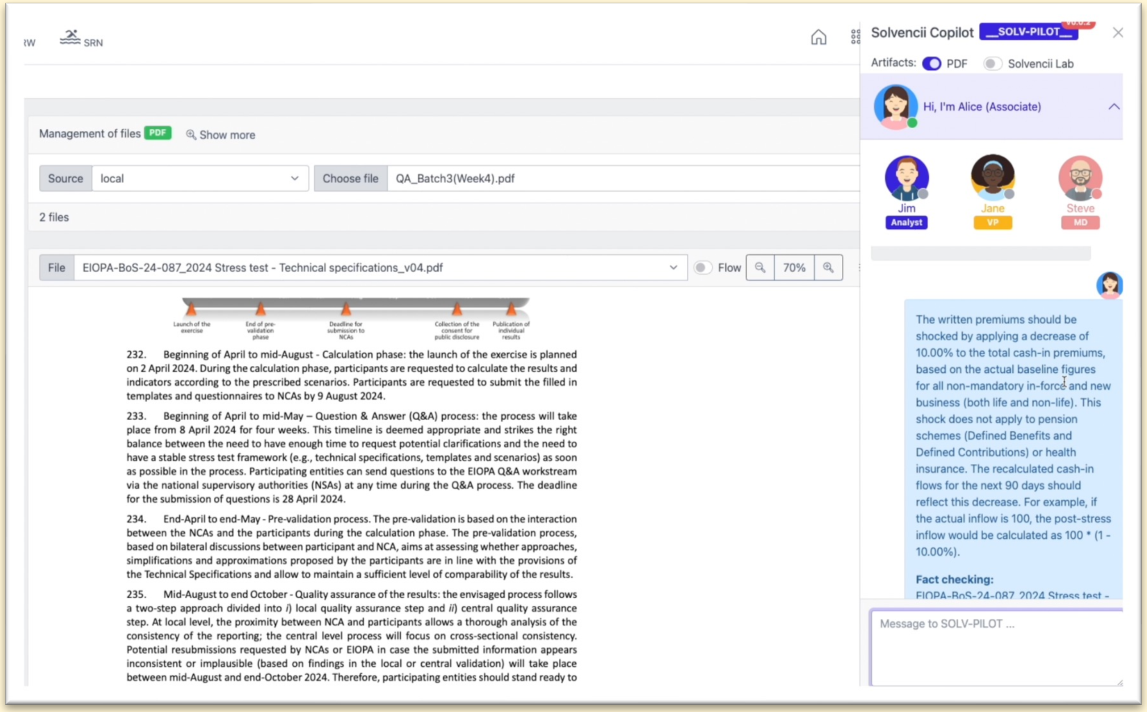
- **Output Interpretation and Decision-Making:** The output of traditional ALM models is often a vast array of scenarios and risk assessments. Interpreting these outputs requires deep domain expertise and can be time-consuming, reducing the effectiveness of decision-making under tight time constraints.

Integrating AI Agents into ALM systems

User Experience:

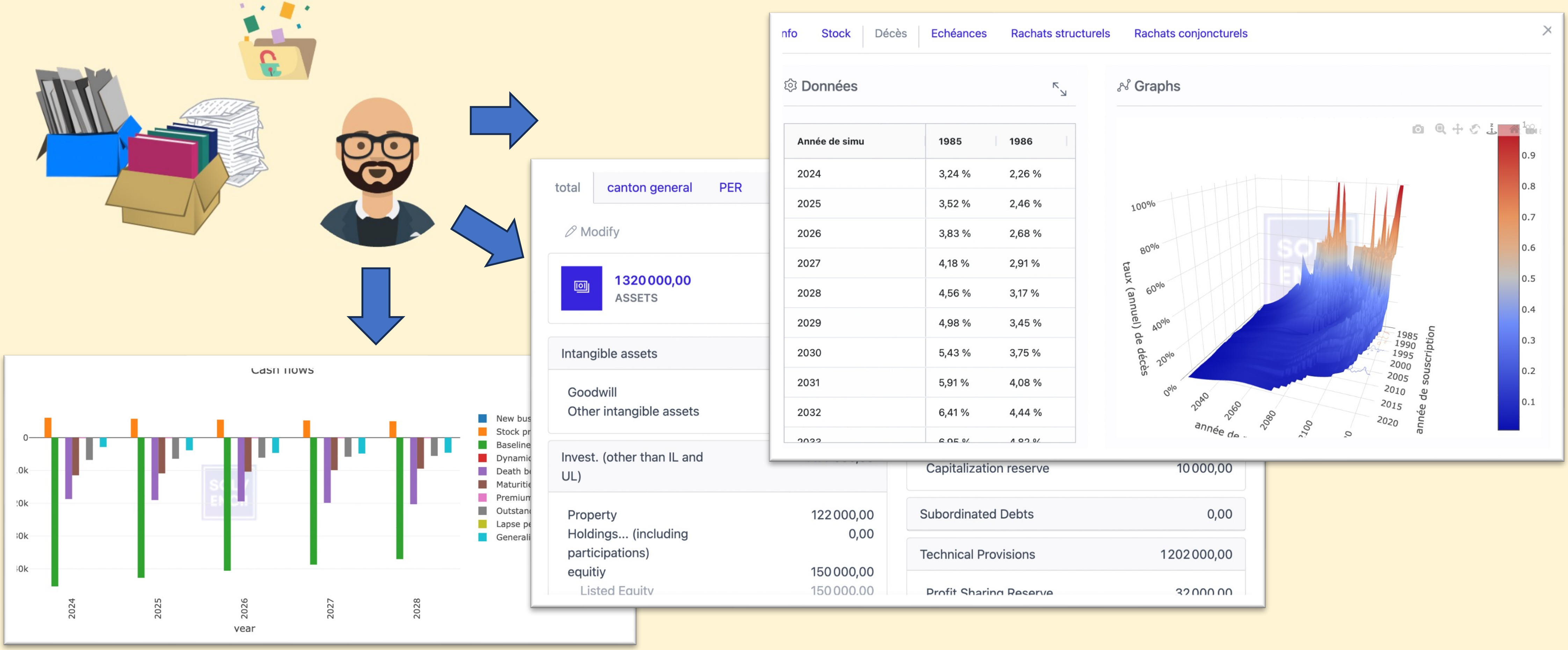
- act as intelligent assistants, helping users navigate the different components of ALM software and interpret documentation.
- answer user queries, explain model outputs, and suggest improvements by leveraging domain knowledge from both the internal ALM model and external financial and regulatory sources available on the internet.





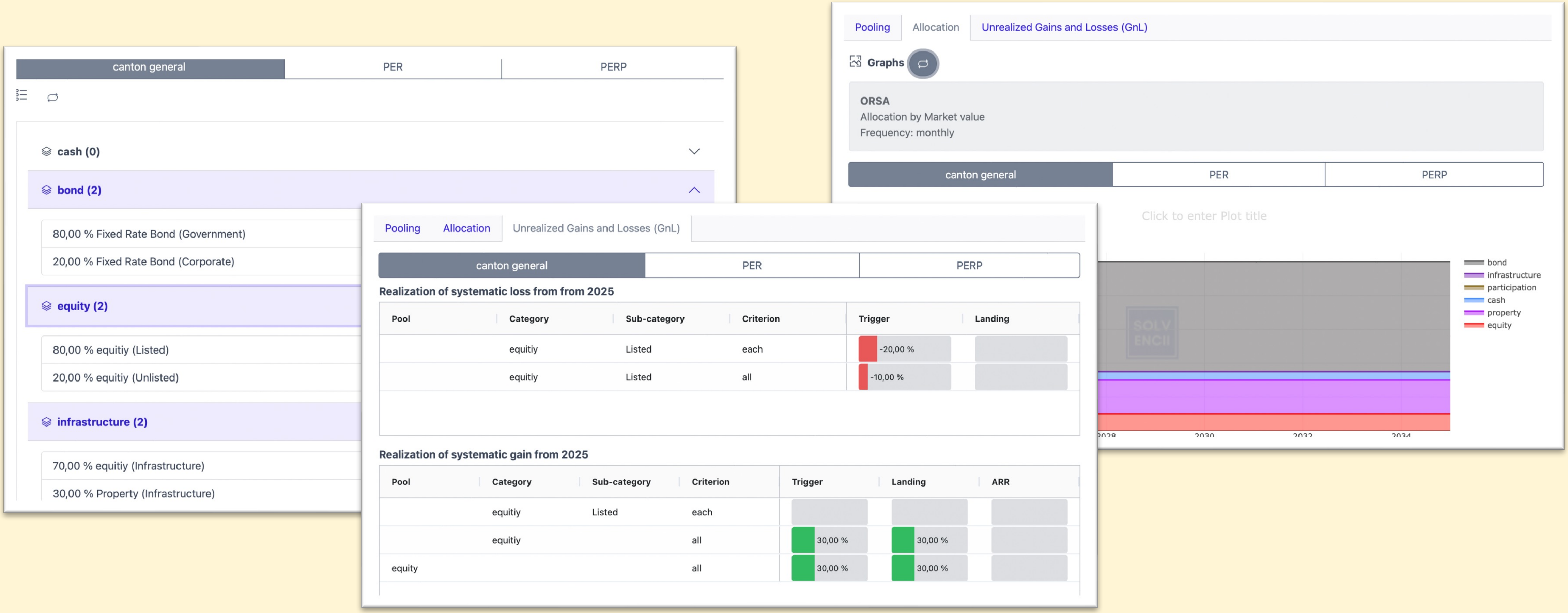
Integrating AI Agents into ALM systems

Automating Data Ingestion: automate the process of collecting, cleaning, and standardizing data from diverse sources, including financial market data, actuarial reports, economic forecasts, and internal balance sheets, and updating model inputs in real time. → Reduce manual errors and ensures that the data is structured in a format compatible with ALM models ; Enable real-time adjustments to ALM strategies, improving the responsiveness of the model to changing market conditions.



Integrating AI Agents into ALM systems

Algorithmic Enhancement: assist in fine-tuning ALM model parameters by employing optimization algorithms and machine learning techniques. (eg. reinforcement learning can be used to identify optimal asset allocation, investment strategies, systematic gain & loss realization, …)

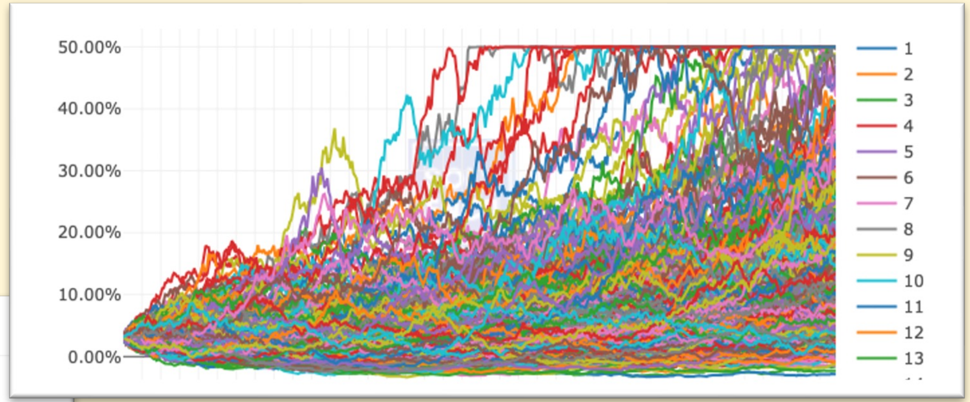


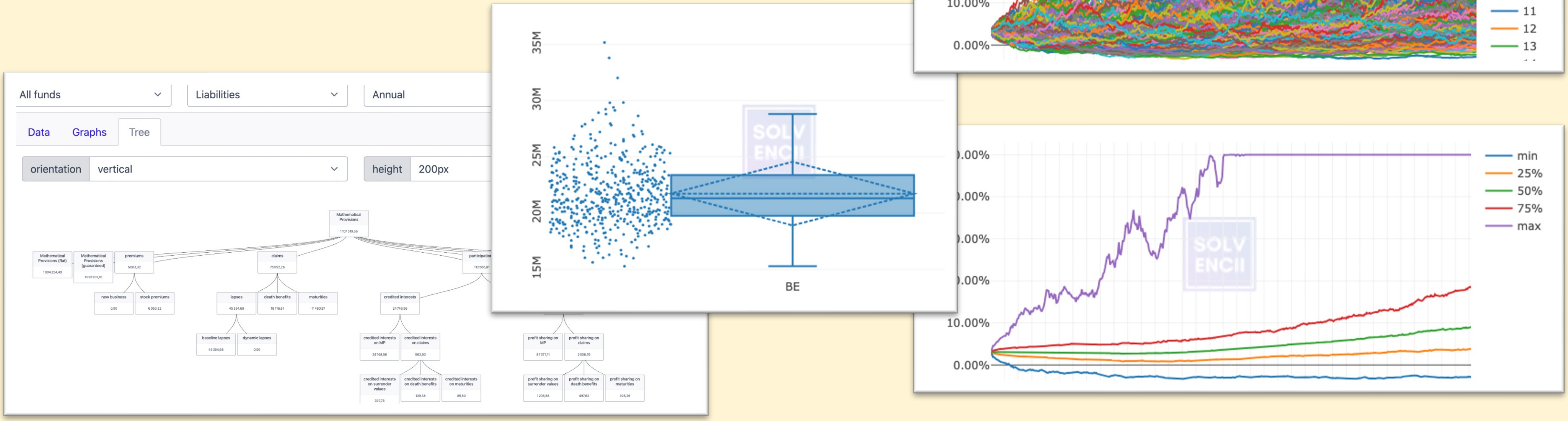
Integrating AI Agents into ALM systems

Output Synthesis: distill insights from complex simulation outputs.

- Identify patterns in large datasets,
- Generate natural language summaries of scenario outcomes, and provide actionable recommendations.

Summary of results, analysis and recommendations





Case Study

Solvencii Copilot in Solvencii Lab

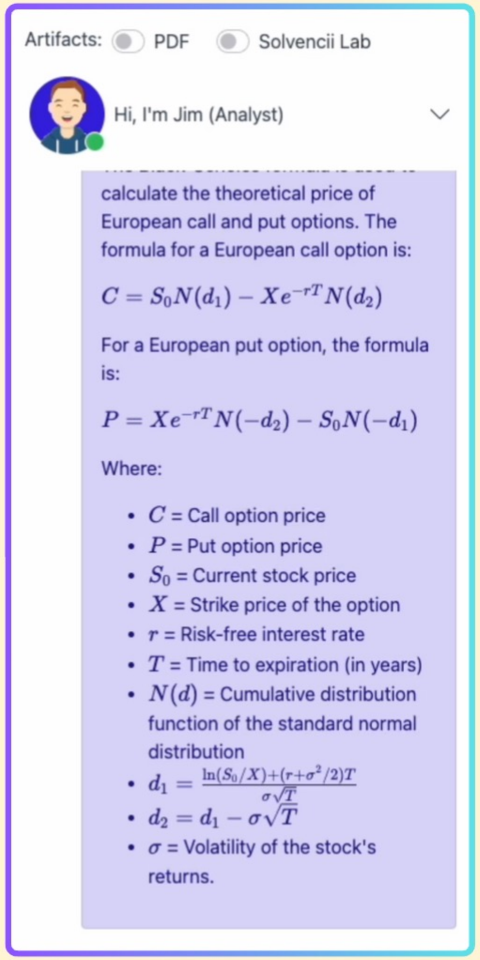
Solvencii Lab & Solvencii Copilot

Solvencii Lab is a SaaS platform for ALM modeling that integrates multiple modules for **data processing**, **simulation**, and **analysis** within a unified environment. Built with modern web technologies, Solvencii Lab offers a flexible and scalable architecture, making it easier to leverage state-of-the-art AI models and cutting-edge software techniques compared to traditional ALM software.

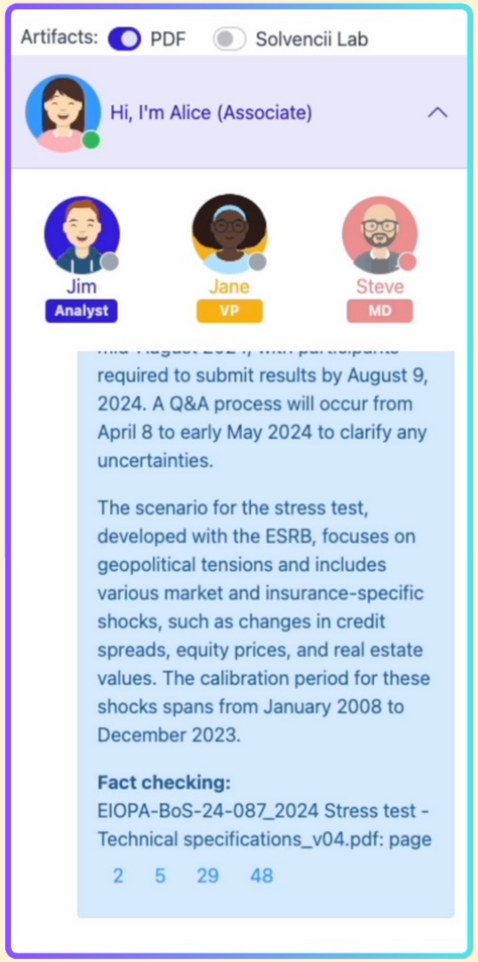
Solvencii Copilot is an advanced AI assistant embedded within Solvencii Lab, designed to extend the capabilities of traditional ALM software by integrating deep financial and actuarial expertise with state-of-the-art language models and tool orchestration. More than just a chatbot, Solvencii Copilot functions as a strategic co-pilot for actuaries, providing insights, automation, and enhanced decision-making support.

Both **Solvencii Lab** and **Solvencii Copilot** are openly and freely accessible at solvencii.fr. Users can explore the platform's full range of ALM modeling functionalities and experience the capabilities of Solvencii Copilot directly.

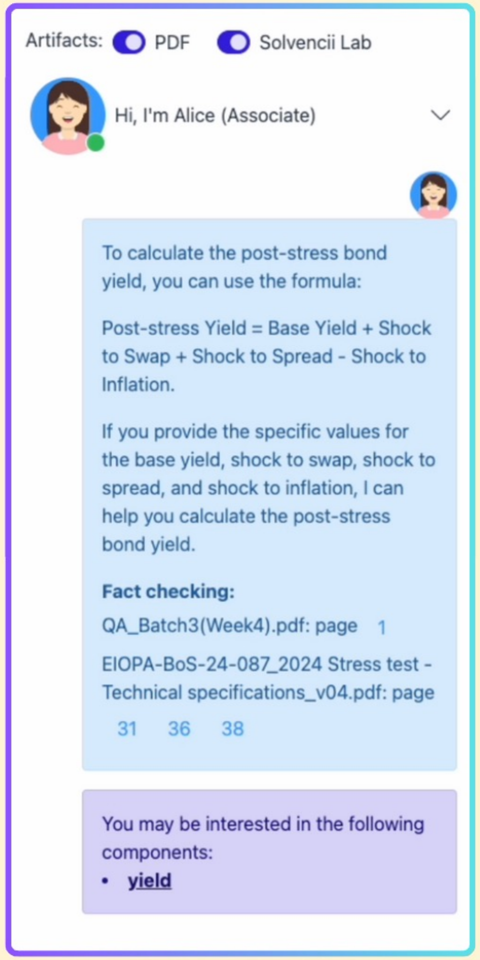
Key capabilities of Solvencii Copilot



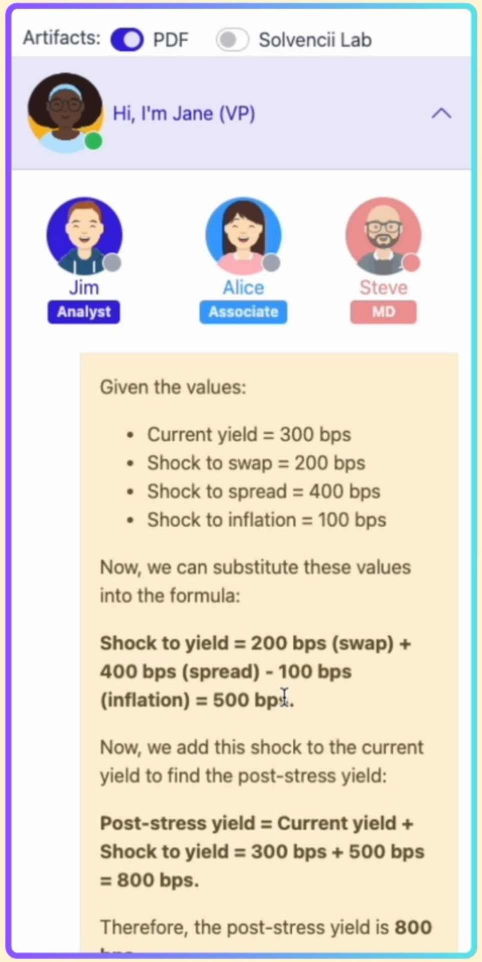
Broad expertise in quantitative finance, actuarial science, and risk management



Retrieval-Augmented Generation (RAG) using external PDF files, with a fact-checking feature



Recommending Solvencii Lab's components and functionalities tailored to user queries



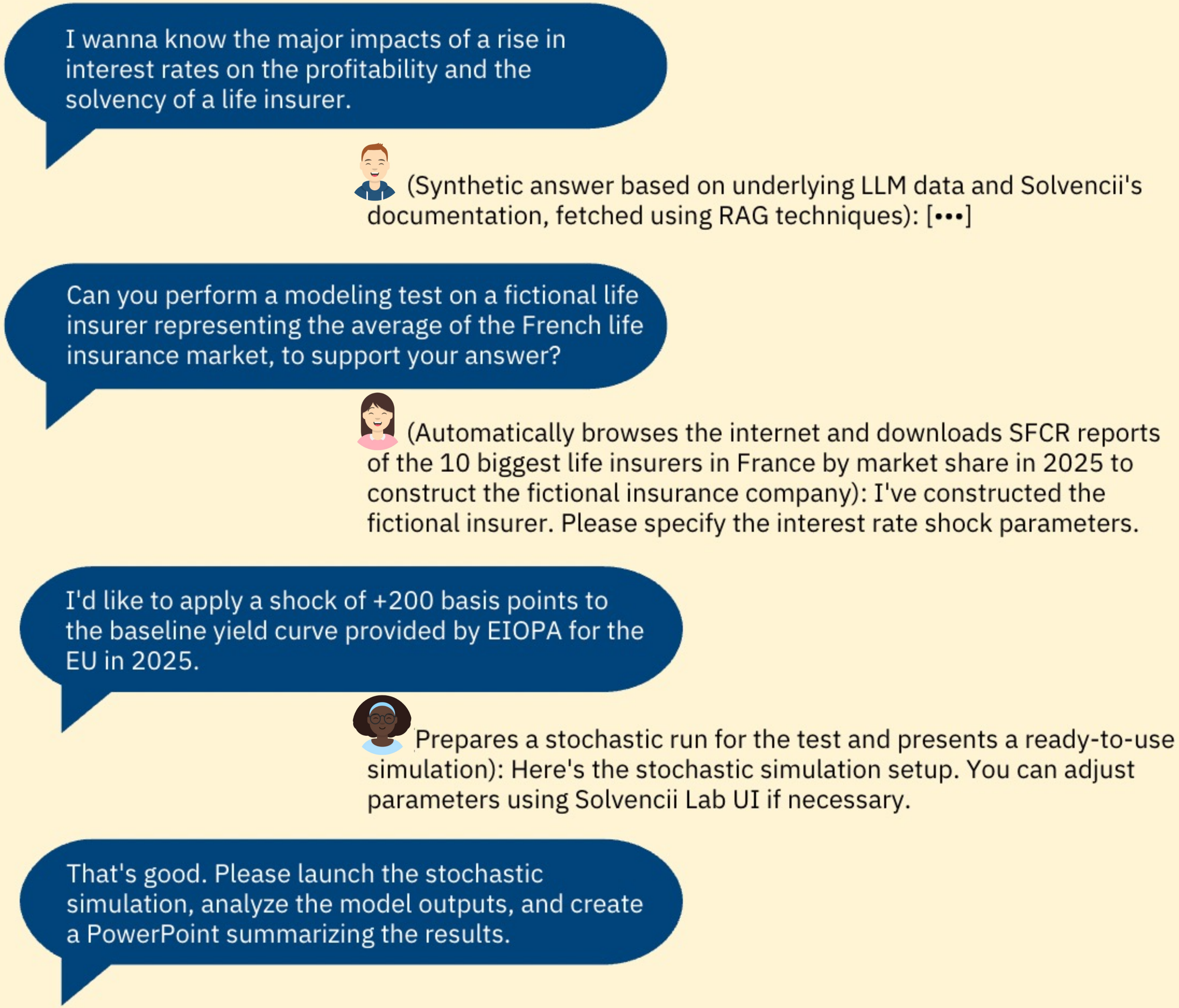
Conversational memory, self-reflection, and self-reasoning for enhanced problem-solving

Illustration

Solvencii Copilot is still evolving alongside the rapid advancements in LLMs and AI agent frameworks.

Its capabilities are expected to grow as AI models become more powerful, context windows expand, and multimodal capabilities improve.

Example scenario illustrating how Solvencii Copilot could assist a user within Solvencii Lab:



Multi-agent architecture

- **Model diversification:** By leveraging different LLM models available on the market, Solvencii Copilot can match each task with the most suitable model. Even if we stick with a single model, we can assign different roles to various AI assistants using prompt templates.
- **Dynamic Orchestration:** Solvencii Copilot can dynamically decide which model to activate for a given task, optimizing for both **performance** and **cost**. For complex calculations or scenario analysis, a high-performing model may be required. For simpler tasks like data formatting or interpretation, a smaller, more efficient model may be sufficient.
- **Collaborative Problem Solving:** In some cases, instead of relying on a single large model, Solvencii Copilot can enable **collaboration between mid-sized models**. This mirrors the real-life scenario where a team of actuaries might work together—each contributing specialized knowledge to solve a complex ALM task.
- **Redundancy and Robustness:** A multi-agent setup also enhances system reliability. If one model underperforms or fails to deliver a satisfactory result, another model can take over or refine the output, ensuring consistent and accurate responses

Behind the scenes

The making of Solvencii Copilot

Model & Framework

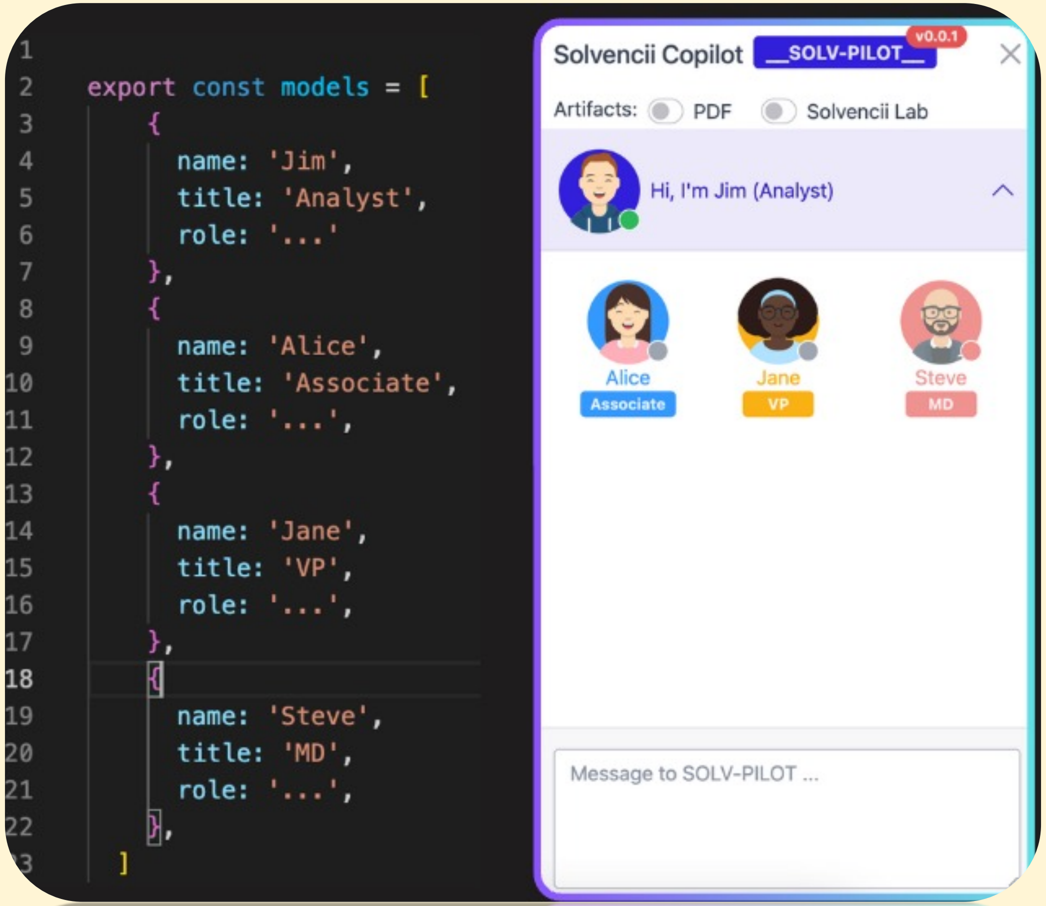
Model & Framework

```javascript
import { ChatOpenAI } from '@langchain/openai'
const llm = new ChatOpenAI({
    apiKey: process.env.REACT_APP_OPENAI_API_KEY,
    model: 'gpt-4o-mini',
    temperature: 0,
  })
```

OpenAI

LangChain

Multi-agent architecture



Prompt Template, History & Memory

prompt

```
export const getAnswer = async (model, question, history) => {
  let answer = ''
  const systemTemplate =
    `You are ` +
    model.name +
    `, ` +
    model.title +
    ` at Solvencii Copilot team (SOLV-PILOT), an AI assistant for
    question-answering tasks on financial, actuarial and risk management
    topics. You can interact with Solvencii Lab, a software for Asset and
    Liability Mangement. Use your knowledge and this conversation history: `
    history +
    ` ` +
    model.role

  const prompt = ChatPromptTemplate.fromMessages([
    ['system', systemTemplate],
    ['human', '{input}'],
  ])
  const chain = prompt.pipe(llm).pipe(parser)
  try {
    answer = await chain.invoke({ input: question })
  } catch (e) {
    console.log(e)
  }
  return answer
}
```

```
questions = [
    'question n°1',
    'question n°2',
    'question n°3'
]

answers = [
    {name: 'Jim', message: 'réponse n°1'},
    {name: 'Alice', message: 'réponse n°2'},
    {name: 'Jane', message: 'réponse n°3'},
]

const history = answers.map(
    (a, i) =>
      'user: ' +
      questions[i] +
      '. ' +
      a.name +
      ': ' +
      a.message
)
```

```
output = await getAnswer(model, prompt, history.slice(-10).join(' '))
```

```
setAnswers((a) => [...a, { name: copilot.name, message: output }])
```

RAG

RAG

```
const docRetrieve = async () => {
  const splits = await textSplitter.splitDocuments(files.map((f) => f.docs).flat())
  const vectorstore = await MemoryVectorStore.fromDocuments(
    splits,
    new OpenAIEmbeddings({ apiKey: process.env.REACT_APP_OPENAI_API_KEY }),
  )
  const retriever = vectorstore.asRetriever()
  dispatch(setRetriever(retriever))
}
docRetrieve()
```

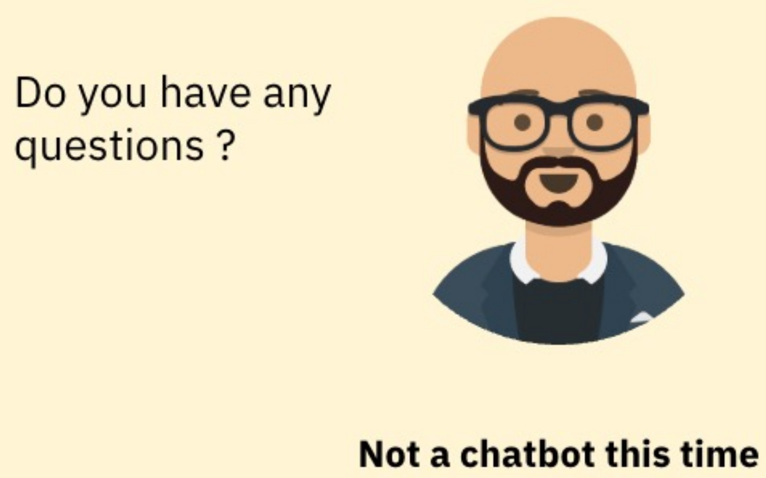
```
export const getAnswerWithRAG = async (model, question, retriever, history) => {
  let answer = ''
  const systemTemplate =
    `You are ` +
    model.name +
    `, ` +
 (SOLV-PILOT), an AI assistant for
 financial, actuarial and risk management
th Solvencii Lab, a software for Asset and
this conversation history: ` +
ugh, use the provided context: {context}`
late.fromMessages([

  const questionAnswerChain = await createStuffDocumentsChain({ llm, prompt })
  const ragchain = await createRetrievalChain({
    retriever,
    combineDocsChain: questionAnswerChain,
  })

  try {
    answer = await ragchain.invoke({ input: question })
    console.log(answer)
  } catch (e) {
    console.log(e)
  }

  return answer
}
```

Thank you!



duc-hien.vu@solvencii.com
contact@solvencii.com

Solvencii.com | Solvencii.fr

https://www.linkedin.com/company/solvencii/