# AI-Driven Underwriting Workflow

A Proof of Concept using GenAI and ML

CAS ETH in Machine Learning in Finance and Insurance
Author: Akos Arendas

November 03, 2025

**Abstract**

This report presents a Proof of Concept (PoC) for enhancing underwriting processes through two main components: (1) a custom Retrieval-Augmented Generation (RAG) system for guideline-focused knowledge retrieval, and (2) a machine learning-based similarity assessment framework for identifying analogous insurance cases. The RAG module integrates Azure OpenAI services with FAISS vector storage to offer fast and context-aware retrieval of internal underwriting manuals, thereby reducing manual document lookups. Complementing this, the similarity analysis leverages dimensionality reduction techniques (UMAP and autoencoders) and density-based clustering (DBSCAN, HDBSCAN) along with simple distance based similarities, to identify historical cases most comparable to incoming quotes. The identified process is a clear example of how modern generative AI techniques, advanced machine learning models, and traditional automation can work together in a modular, scalable architecture. This approach enables immediate impact – reducing guideline lookups, standardizing decisions, and freeing underwriters to pursue strategic new-business opportunities. While results are promising, further validation (in particular the similarity engine), broader textual data ingestion, and production-grade integration with pricing tools are planned to ensure robustness, governance, and readiness for production deployment.

# 1 Introduction

## 1.1 Business background

The project is situated within Zurich Global Employee Benefits Solutions (ZGEBS) reinsurance business. In this model, the reinsurer (ZGEBS) collaborates with a network of local insurance partners to provide group protection products (such as life, disability, accident and medical covers) to multinational companies. While the policies are issued locally, the risks are pooled and managed globally. The business operates in a business-to-business (B2B) framework, with contracts typically renewed on an annual basis. This means that the portfolio consists of two main types of quotations:

- **New business**: representing new client acquisitions or new subsidiaries joining the global program.
- **Renewal**: representing existing contracts that are re-priced or re-evaluated each year.

Each renewal was once a new business case, meaning that the historical information about every case accumulates over time. This historical dimension plays a critical role in the underwriting workflow.

## 1.2 Motivation

Within ZGEBS, the underwriting process distinguishes between first-stage and second-stage underwriting. The latter is not in the scope of the current report. The first-stage underwriting focuses on reviewing the work performed by the local insurer and assessing its alignment with the reinsurer's internal standards and technical guidelines.

At a high level[1], when an underwriter receives a case, the following decision flow applies:

[1]This is a simplified approach, primarily for the sake of clarity. The actual business processes are more complex, but fall beyond the scope of this report.

1. Classification: determine whether the case represents new business or a renewal.
2. For new business:
    - Retrieve any existing information about the multinational client or related subsidiaries.
    - Identify similar cases in the existing portfolio to guide technical assumptions.
    - Review the internal underwriting guidelines for applicable parameter ranges, exclusions, and manual adjustments.
3. For renewals:
    - Review last year's quotation, pricing assumptions, and textual comments stored in the pricing tool or internal documentation.
    - Assess whether new information is available from the local insurer, the network partner, or internal systems (emails, pricing notes, claims data).
    - Quantify and interpret any changes in experience, exposure, or plan design, and determine whether these changes are material.
    - Validate that the renewed case still complies with internal policies and pricing frameworks.

After this analysis, the underwriter manually updates the pricing tool to generate the quotation. This includes re-selecting parameter tables (e.g., mortality or morbidity tables) and applying the relevant industry-specific or discretionary adjustment factors.

## 1.3 Problem statement

Although the described workflow ensures high technical quality, it has become increasingly resource-intensive and time-consuming, particularly for renewal cases. Empirical observations and internal feedback indicate that underwriters spend a disproportionate amount of time re-analyzing renewals, cases that often exhibit only marginal year-to-year changes.

This manual validation effort limits the team's capacity to focus on new business opportunities, which are strategically more important for growth and portfolio diversification. In particular:

- A large portion of the underwriting time is consumed by data retrieval and historical validation rather than analytical assessment.
- Significant expertise is required to interpret free-text information from emails, pricing notes, and comments, which are not systematically structured.
- Manual system interactions – such as initializing pricing tools and selecting parameter tables – further contribute to operational inefficiencies.

As a result, the current setup constrains scalability and limits the reinsurer's ability to expand its new business pipeline.

For new business, the risk assessment is thorough, hence it is time consuming by nature. Checking internal guidelines are must, and one of the early steps involves portfolio assessment, in which similar cases are explored. This many times based on expertise and heuristics. Inconsistent decision-making across underwriters, challenges in retaining and transferring institutional knowledge, and difficulties in scaling processes to manage increasing data volumes are potential fields for machine learning automation techniques.

## 1.4 Objective of the Project

The objective of this project is to design an AI-enhanced underwriting workflow that accelerates the assessment and quotation process, while maintaining technical robustness and compliance with underwriting standards.

The proposed solution aims to combine several technological elements:

- Generative AI components for information retrieval, summarization, and text interpretation (e.g., retrieve case-specific information from internal guidelines and extracting insights from historical comments or emails).
- Machine learning elements for similarity assessment (or change detection by flagging material deviations).
- Process automation and API integration to enable smoother data exchange between systems and reduce manual re-entry of information.

Together, these components are expected to form a semi-automated, intelligent underwriting assistant that supports human decision-making and reallocates underwriters' time towards high-value new business analysis. It is important to note that the ultimate goal is to provide a comprehensive recommendation system/application with a streamlined user-interface for underwriters in ZGEBS, both for renewals and for new business. The semi-automated approach is deliberately designed to keep human expertise at the center of underwriting decisions, ensuring that AI-driven efficiencies are balanced with robust risk control, regulatory compliance, and governance standards – so that every recommendation supports responsible, reliable, and auditable business practices.

**Expected benefits and measurement**

- Efficiency gains: reduction in time spent per renewal case; lower manual re-entry rates.
- Throughput expansion: more renewals processed per underwriter, freeing capacity for new business work.
- Quality and consistency: standardized extraction of key parameters and rationale, improved traceability of pricing decisions.
- Strategic impact: faster response to market opportunities and improved ability to scale global programs.

**Alignment with Zurich's values and strategy**

- Brighter: Use of AI to enhance underwriting capabilities in a thoughtful, reliable, and user-friendly way.
- Future: Building scalable, data-driven processes that support sustainable growth and portfolio diversification.
- Together: Collaborative augmentation of underwriters' expertise, maintaining human-in-the-loop governance.

## 1.5 Scope of the Report

The current report outlines certain aspects of the broader project. In different sections I describe two independent parts. Section 3 (RAG system) focusing on a custom made knowledge retriever system and in section 4 (Similarity assessment) a similarity assessment model with machine learning techniques is described.

Given the complexity of the business (reinsurance structure, various covers, different international programs (captive, multinational pooling, global underwriting) etc.), the current report focuses only on selected cases. For the knowledge retrieval, known cases are selected, where the result is known and the historical documentation is adequate. The similarity assessment focusing on one coverage (life), however the ML pipeline can be applied for other benefit types also.

There are many additional aspects of the whole underwriting process (e.g., the experience rating and credibility pricing) which are not covered, however must be part of the detailed project scope in later stages. Potential next steps are listed in section 5 (Limitations and Future Work).

# 2 Technical details of the UW workflow

## 2.1 Local Client Input Sheet (LCIS)

Underwriters receive the data in excel files via emails. These templates are called LCIS (Local Client Input Sheet) and contains various information about the local company (subsidiary of the multinational). There are local company level information (e.g., name, industry category, size, etc.), benefit design related data (census data for the insured population for life or disability coverages, waiting periods (in case of disability), or for example termination age), and also claims experience (for credibility pricing). With proper ETL (extract-transform-load) process, it is important to parse the important data for knowledge retrieval (e.g., what is the country, how big the company is, characteristics of the insured population) and for similarity assessment as well.

## 2.2 Textual data

There are multiple source of textual data that can be utilized in the workflow. Even though, the current work focuses on the Underwriting Guideline, the list of the textual sources is:

- Guidelines, policies, manuals (such as the aforementioned UW guideline)
- Correspondence between the local insurance company (network partner – NWP), mainly emails
- Internal converastion between doer and reviewer
- Comment fields in the LCIS templates
- Saved commentary from the pricing database (several fields)

## 2.3 Pricing tool

By uploading the input data (LCIS) to the first stage pricing tool, an entity is created for the given customer, in a given country for a given year for the selected benefits. The selection of the different parameters and assumptions has to be done afterwards. The pricing outputs are stored in SQL databases, which serve historical pricing assumption retrievals and also for similarity assessment.

# 3 RAG system

The literature for different technologies that are capable for complete system is extensive, offering a wide range of options to choose from according to specific needs and preferences. In Appendix 6.1 there are references related to RAG, FAISS, LangChain and OpenAI literature.

Our RAG (Retrieval-Augmented Generation) system is designed for underwriting guideline analysis and recommendations. Currently it is focusing on internal manuals. The system leverages Azure OpenAI models with FAISS vector storage to provide intelligent document retrieval and AI-powered responses for a given quote that the underwriters receive from our NWP via the LCIS templates. The system works for both renewal and new business.

In our PoC solution we built a system that contains the following elements:

- **Custom Azure OpenAI Integration**: Direct REST API implementation with comprehensive error handling and dual-endpoint architecture
- **FAISS-Based Architecture**: In-memory vector storage solution (for the PoC, cloud-based storage and API connection is planned for PROD)
- **LangChain Agent**: Purpose-built agent tools for insurance underwriting analysis and risk assessment (parameters and assumption factors)
- **Modular Data Processing Pipeline**: Comprehensive system for LCIS (Local Contract Information Sheet) extraction and historical data enrichment

## 3.1 System Architecture

The UW AI POC system follows a layered architecture that transforms raw underwriting data through multiple processing stages into intelligent AI-powered responses.

Figure 1 illustrates the complete system architecture, showing the modular design and data flow.

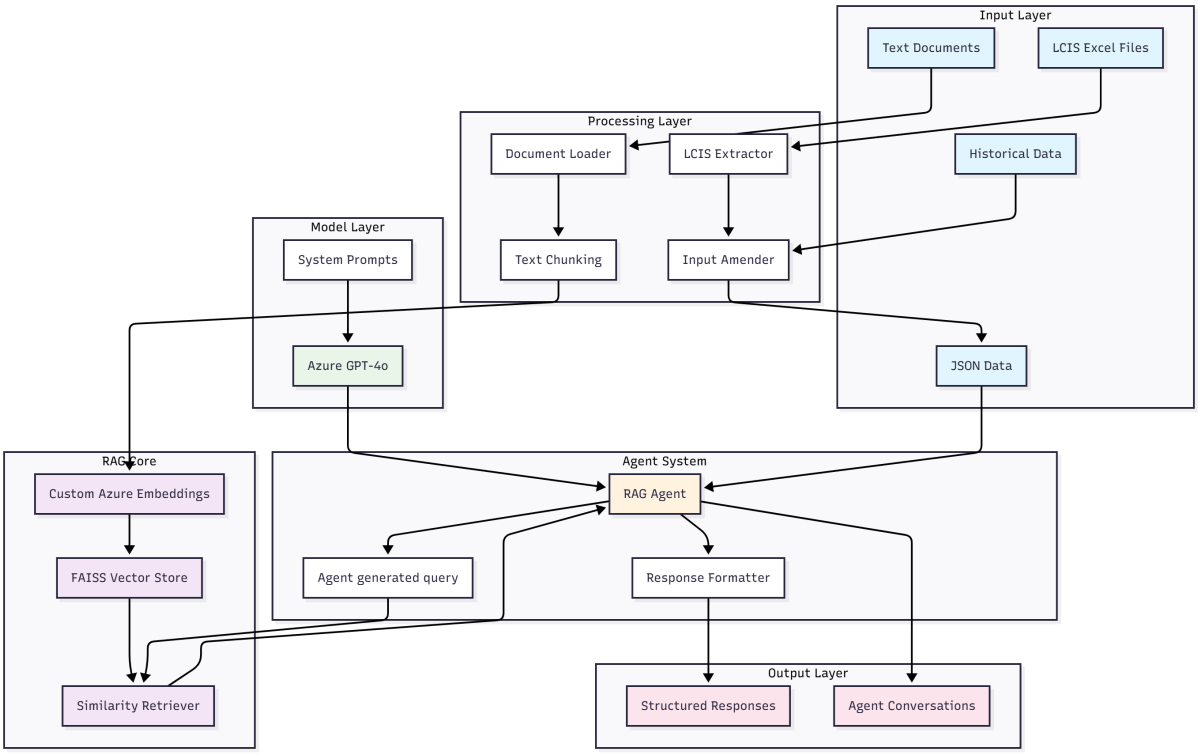



Figure 1: UW AI POC System Architecture Overview

The data flows through six distinct layers, each with specialized responsibilities:

1. **Input Layer**: Data ingestion layer, the inputs are detailed in section 2. The JSON data (which is the output of the extracted and enriched raw data) is a temporary file, which serves as an input for the AI Agent.

2. **Processing Layer**: Raw inputs undergo specialized processing to prepare them for the RAG system. Data transformation step produces enriched JSON data that becomes available to the agent system, while the document processing path feeds into the RAG core for knowledge base construction.

3. **RAG Layer**: This knowledge retriever layer implements the retrieval-augmented generation infrastructure. First, custom Azure embeddings convert text chunks into high-dimensional vectors using Azure OpenAI's `text-embedding-ada-002` model, creating rich semantic representations of document content. These embedding vectors are then stored in a FAISS vector store, which provides an efficient in-memory index for rapid similarity searches across thousands of documents while maintaining metadata associations for each chunk. When a query is received, the similarity retriever conducts k-nearest neighbor searches, ranks documents by their semantic similarity to the query, applies score thresholds to ensure result quality, and returns the most relevant document chunks for further processing.

4. **Model Layer**: The Model Layer serves as the foundation for natural language understanding and generation. At its core is Azure GPT-4o (currently `gpt-4o-20240513-globaltext`), the primary large language model that generates responses, interprets context, and reasons about underwriting scenarios. Supporting this capability, there are system prompts, which consist of configurable instruction templates that guide the model's behavior, define output formats, incorporate JSON interpolation for dynamic data injection, and ensure consistency in every response produced.

5. **Agent Layer**: The Agent System uses LangChain architecture and coordinates retrieval and AI-generated responses by combining inputs from multiple sources, optimizing search queries, and formatting outputs with metadata and citations. While this describes the intended final design, development for the PoC is still ongoing. Underwriters will be able to rerun analyses or request additional tasks as needed once the system is complete. Currently the structured responses are generated.

6. **Output Layer**: The Output Layer delivers results in structured formats, such as JSON with defined schemas, extracted data, recommendations, and supporting evidence. It also provides agent conversations in natural language, including multi-turn histories and explanations for underwriting decisions. This describes the planned end-state; for the PoC, development is ongoing, and underwriters will be able to rerun analyses or request additional tasks once the system is fully implemented. As mentioned above, the structured output can be already produced.

## 3.2 Technical Implementation Details

### 3.2.1 Dual-Endpoint Azure OpenAI Architecture

A critical architectural development in Azure OpenAI integration is the dual-endpoint approach for LLM endpoint configuration. This design allows the system to maintain separate endpoints for different AI services, ensuring that each service operates independently and efficiently within the overall infrastructure.

```
# LLM Endpoint Configuration
LLM_ENDPOINT = "https://dbi-llm-loungecs-we-dev-app-001.azurewebsites.net/api"
LLM_DEPLOYMENT = "gpt-4o-20240513-global"
LLM_API_VERSION = "2023-05-15"

# Embedding Endpoint Configuration
EMBEDDING_ENDPOINT = "https://dbi-llm-loungeeg-we-dev-app-001.azurewebsites.net
    /api/openai/deployments"
EMBEDDING_DEPLOYMENT = "text-embedding-ada-002"
EMBEDDING_API_VERSION = "2023-12-01-preview"
```

This separation provides several benefits. Service isolation enables the independent scaling and monitoring of large language model (LLM) and embedding services, allowing each to operate efficiently without impacting the other. Performance optimization is achieved through specialized endpoints tailored for different workload patterns, ensuring that each service delivers optimal results. Cost management is streamlined by separating billing and quota management for different types of AI services, providing clearer oversight and control. Additionally, reliability is enhanced through fault isolation, which prevents cascading failures between services and supports a more robust infrastructure. This setup provides a production ready architecture.

### 3.2.2 FAISS Vector Storage Implementation

The system implements FAISS-based vector storage with several key design decisions. FAISS is configured for in-memory operation for the PoC.

```
def create_retriever_from_texts(texts, k=2, embeddings=None):
    """Create FAISS retriever from text documents"""
    if embeddings is None:
        embeddings = create_custom_embeddings()

    # Create FAISS vector store in memory
    vectorstore = FAISS.from_texts(
        texts=texts,
        embedding=embeddings
    )
```

```
    # Configure similarity search retriever
    retriever = vectorstore.as_retriever(
        search_type="similarity",
        search_kwargs={"k": k}
    )

    return retriever
```

Key features include in-memory storage. The system offers configurable similarity search with an adjustable k-value, supporting both similarity and `similarity_score_threshold` search types. Additionally, it utilizes efficient CPU-based indexing, making it well-suited for PoC purposes.

### 3.2.3 Custom Azure OpenAI Embeddings

The system implements a custom embedding class for direct Azure OpenAI integration:

```
class CustomAzureEmbeddings:
    def __init__(self, api_url, api_key, model_deployment_name):
        self.api_url = api_url
        self.api_key = api_key
        self.model_deployment_name = model_deployment_name

    def embed_query(self, text: str) -> List[float]:
        """Generate embedding for single query"""
        return self._generate_embeddings([text])[0]

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        """Generate embeddings for document batch"""
        return self._generate_embeddings(texts)

    def _generate_embeddings(self, texts: List[str]) -> List[List[float]]:
        """Direct REST API call to Azure OpenAI"""
        # Implementation with error handling and retry logic
```

This custom implementation provides direct API control, allowing full management over request formatting and error handling. It supports batch processing for efficient handling of multiple documents in single requests, and is designed for seamless enterprise integration, aligning with Azure authentication and security standards. Additionally, it offers robust error resilience through comprehensive error handling, including retry logic and fallback mechanisms. Lastly, this fits into the internal ecosystem, since many default features are not available internally

### 3.2.4 LangChain Agent Integration

The system implements specialized LangChain agents:

```
def working_rag_agent(retriever, system_prompt=None):
    """Create RAG agent for underwriting analysis"""

    @tool
    def interrogate_underwriting_guideline(query: str) -> str:
        """Retrieve info from Underwriting Guidelines via semantic similarity
            """
        docs = retriever.invoke(query)
        return "\n".join([
            f"Source: {doc.metadata.get('source', 'Unknown')}\n"
            f"Content: {doc.page_content}"
            for doc in docs
        ])

    def agent_invoke(input_data):
```

```
        """Agent workflow with autonomous tool selection"""
        # Extract user input
        user_input = extract_input(input_data)

        # Retrieve relevant documents
        retrieved_info = interrogate_underwriting_guideline(user_input)

        # Generate LLM response with context
        llm = new_simple_model()
        prompt = construct_prompt(system_prompt, retrieved_info, user_input)
        response = llm.invoke(prompt)

        return format_response(response)

    return SimpleRAGAgent(agent_invoke)
```

## 3.3 RAG Pipeline Data Flow

Figure 2 illustrates the dataflow in the RAG pipeline. In the process diagram, many components are provided by the different technologies (e.g., by LangChain orchestration), however it illustrates how the knowledge retrieval works:

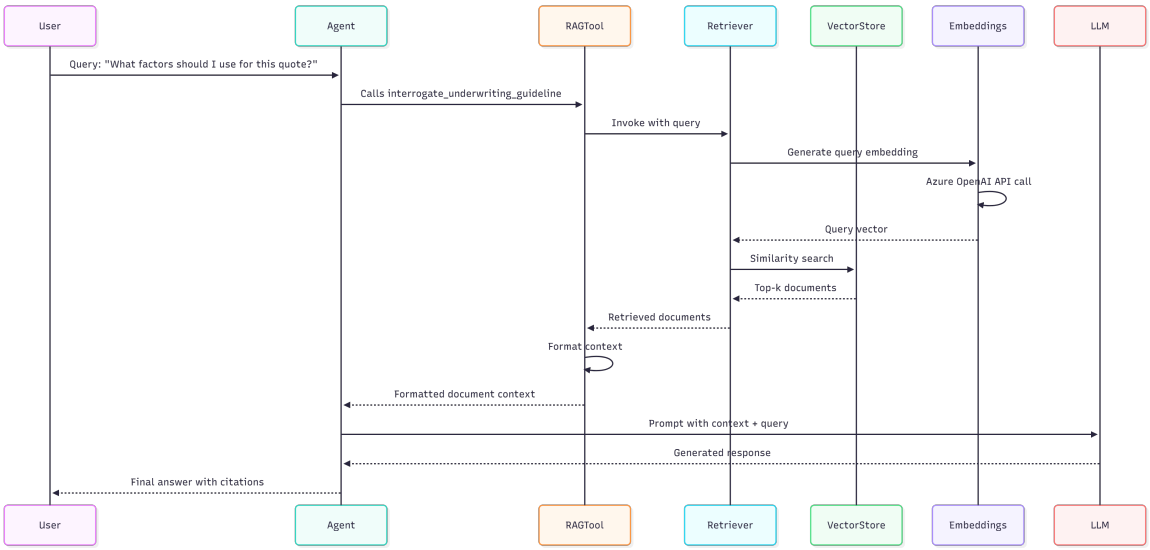



Figure 2: Step-by-step process flowchart of the RAG pipeline. The process starts at the upper left corner and goes to the right and from top to down, following the arrows.

## 3.4 Example

Given proprietary information, only the structure can be presented of the output in this report, and only partially. For instance, the Activity and Occupational loadings are recommended by an exact factor, however now a general sentence is provided. So as in Source, the name of the section names are masked. However, this represents the structured output (JSON). The agent provides a general summary which is followed by exact recommendations with references.

```
{
  "General_Summary": "For the underwriting of the group insurance quote for RAG
      Inc. in <a country>, several guidelines need to be followed. We need to
     use the UK life table and apply a 85% adjustment factor. Activity loading
     and occupational loading should consider specific occupation and location
     adjustments or SI codes. The guarantee period is typically 1-2 years, but
     extensions may increase the rates by approximately 5% per year. ",
```

```
"Table": [
  {
    "Item": "Life table",
    "Guideline Summary": "Use the UK life table with a 85% adjustment.",
    "Source": "4.3.3 <section name>"
  },
  {
    "Item": "Adjustment Factor",
    "Guideline Summary": "Apply a 85% adjustment factor.",
    "Source": "4.3.3 <section name>"
  },
```


### 3.5 Business impact and assessment

The system potentially cuts guideline lookup time from hours to minutes (or even seconds), applies standards consistently across teams and regions, elevates decision quality with expert-level guidance for complex cases, and enforces regulatory requirements systematically.

This project showcases enterprise-grade AI architecture with robust monitoring and error handling, deep alignment with specialized business needs, performance optimization for low-latency use, and secure, compliant, and scalable integration into enterprise environments.

The UW AI PoC demonstrates a practical, high-performance RAG architecture with clear business value, strong error handling and observability, and a design ready to scale. While the PoC is not the final state, it establishes a solid foundation for enterprise deployment and future enhancements. Overall, it provides a meaningful contribution to applied AI engineering in regulated settings and a strong basis for ongoing development.

## 4 Similarity assessment

We discussed knowledge retrieval in the previous section, which can support the underwriting process by checking internal guidelines see whether there is an exact guidance on the approach that should be applied for a given quote. From another aspect, we can retrieve other information from the history of our own portfolio.

Understanding of similar insurance policies support the underwriting process from different aspects. Primarily it helps the decision for a new business what kind of parameter tables (e.g., mortality tables) and adjustments (country related, industry related, etc.) should be applied. The main question is, "What was out approach for those cases?".

On the other hand, understanding similarities might help improve the quality of the first stage underwriting by detecting potential errors or overlooked aspects in case of renewals.

By leveraging machine learning-based similarity search methodologies, the system aims to enhance decision-making processes, and targeting to reduce processing time by 70-80%, and improve consistency across underwriters.

The following sections contain the ML pipeline and the assessment of the current similarity model. Even though, the current report doesn't aim to prove theoretical mathematical results, on one hand, certain techniques and their original source is referenced, and on the other hand, some mathematical formulas are presented.

Aspects of technical implementation in Python can be found in Appendix 6.2.

### 4.1 Available Data

The analysis utilizes a comprehensive dataset comprising from the last 3 years:

- **Volume**: 3 746 Life insurance records after coverage filtering

- **Features**: 26 business attributes expanding to 261 engineered features
- **Coverage Types**: Focused on Life insurance with extensibility to Disability, Accident, and Medical (subsection 1.5 (Scope of the Report))
- **Geographic Scope**: 93 countries represented in the dataset

Key feature categories include program structure (ZIPE, Global Profit Share, Captive), geographic information, demographic characteristics, financial metrics, and technical policy design elements. Comprehensive data quality measures ensure currency standardization, missing value imputation, and categorical range conversion for improved model stability.

## 4.2 Machine Learning Pipeline

The machine learning pipeline integrates dimensionality reduction techniques with density-based clustering to transform high-dimensional insurance data into actionable similarity assessments. Our approach combines manifold learning methods (McInnes et al., 2018) with deep neural network autoencoders (Hinton and Salakhutdinov, 2006) to learn effective low-dimensional representations that preserve meaningful relationships in the original data space (Bengio et al., 2013).

### 4.2.1 Import of Resources

The implementation leverages a comprehensive Python ecosystem for machine learning and data analysis, including `pandas` and `numpy` for data manipulation, `scikit-learn` for preprocessing and clustering algorithms, `umap-learn` (McInnes et al., 2018) for manifold learning and dimensionality reduction, `tensorflow`/`keras` for neural network implementation (autoencoder); and `dbscan` and `hdbscan` (McInnes et al., 2017) for non-hierarchical and hierarchical density-based clustering.

To ensure reproducibility, all experiments use fixed random seeds [2]:

```
import numpy as np
import tensorflow as tf
np.random.seed(42)
tf.random.set_seed(42)
tf.keras.utils.set_random_seed(42)
```

### 4.2.2 Data Generation and Preprocessing

The preprocessing pipeline implements a systematic approach to handle the complexity of insurance data:

**Data Loading and Quality Control** Multi-source integration processes main insurance data, FX rates, and feature mapping files. Coverage filtering focuses on Life insurance records (3 746 from 12 383 total). Quality validation includes schema checking and completeness verification.

Data cleaning operations address infinity values (converted to 0) and categorical standardization (invalid values are converted to NaN).

**Currency Conversion and Categorical Processing** All monetary values are converted to CHF using latest exchange rates for Gross_Premium_CHF, Max_Sum_Insured_CHF, Average_Sum_Insured_CHF, and Total_Sum_Insured_CHF.

Numerical ranges are converted to categorical bins for improved stability:

- Number_Lives: 6 categories (0-499, 500-999, 1000-2999, 3000-4999, 5000+)
- Termination_Age: 4 categories (60-65, 66-70, 70+)
- Ratio_Men: 4 gender ratio categories

[2] Why 42? Because it is the "*Answer to the Ultimate Question of Life, the Universe, and Everything*" (Adams, 1979).

### 4.2.3 Data Split

The pipeline doesn't contain supervised learning that requires train/test splits. The autoencoder training uses a 20% validation split (80% training, 20% validation) during the neural network optimization process only. No separate test set is maintained as this is an unsupervised similarity analysis.

### 4.2.4 Exploratory Data Analysis

Statistical analysis reveals a dataset shape of (3 746, 26) after preprocessing, with 4 numerical and 22 categorical features, and 0 missing values post-imputation.

The numerical features are highly correlated, however in the current analysis all of them are kept.

Factor Analysis of Mixed Data (FAMD) with 20 components reveals total inertia of 2 578, with primary components showing 359.3%, 135.4%, and 112.9% explained variance. The high explained variance in the first component suggests strong underlying structure in the insurance data, validating the potential for effective dimensionality reduction.

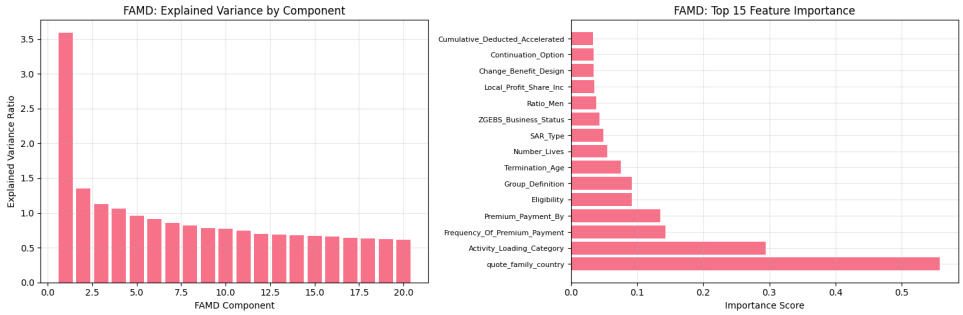



Figure 3: Factor Analysis of Mixed Data (FAMD) - Feature importance and explained variance visualization showing the contribution of different variables to the principal components.

Figure 3 displays the FAMD analysis results, illustrating the relative importance of features in explaining the variance within the insurance dataset. The visualization reveals which features contribute most significantly to the underlying data structure, supporting the dimensionality reduction approach. Certain approaches suggest to reduce already the feature variables after FAMD, but in the current analysis, all the variables are kept.

### 4.2.5 Data Normalization

`MinMaxScaler` (variables are not normally distributed, hence `StandardScaler` is not a natural choice) is applied to numerical features (Gross_Premium_CHF, Max_Sum_Insured_CHF, Average_Sum_Insured_CHF, Total_Sum_Insured_CHF), scaling to [0,1] range for consistent neural network input. This normalization is critical for neural network training, ensuring that features with different scales contribute appropriately to the learning process.

One-hot encoding transforms 22 categorical variables into 257 binary indicator variables, preserving categorical relationships while enabling numerical computation. The final feature space comprises 261 dimensions (4 numerical + 257 binary). While this creates a high-dimensional representation, dimensionality reduction techniques like UMAP and autoencoders aim to discover compact, meaningful representations from such sparse, high-dimensional inputs.

## 4.3 Dimensionality Reduction

### 4.3.1 UMAP Implementation

UMAP (Uniform Manifold Approximation and Projection) (McInnes et al., 2018) provides topologically-aware dimensionality reduction preserving both local and global data structure. Unlike traditional dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-SNE (van der

Maaten and Hinton, 2008), UMAP is grounded in manifold theory and Riemannian geometry, allowing it to preserve both local neighborhood structure and global topological properties of the data (McInnes et al., 2018).

UMAP constructs a k-nearest neighbor graph in high-dimensional space, then optimizes a low-dimensional embedding to preserve the topological structure by minimizing [3]:

$$\mathcal{L} = \sum_{i,j} w_{ij} \log\left(\frac{w_{ij}}{v_{ij}}\right) + (1 - w_{ij}) \log\left(\frac{1 - w_{ij}}{1 - v_{ij}}\right) \tag{1}$$

where $w_{ij}$ represents high-dimensional affinities and $v_{ij}$ represents low-dimensional affinities.

Implementation parameters include `n_components=2` for 2D visualization space, `n_neighbors=15` to balance local and global structure preservation, `min_dist=0.1` for minimum separation in low-dimensional space, and `metric='euclidean'` for distance metric in high-dimensional space. The choice of these hyperparameters follows established best practices for UMAP applications, where `n_neighbors` controls the balance between local and global structure (lower values emphasize local structure, higher values emphasize global structure), and `min_dist` determines how tightly points can be packed in the low-dimensional representation.

#### 4.3.2 Autoencoder Implementation

Autoencoders represent a powerful class of neural network architectures for unsupervised learning and dimensionality reduction (Hinton and Salakhutdinov, 2006; Goodfellow et al., 2016). The seminal work by Hinton and Salakhutdinov (2006) demonstrated that deep autoencoders can learn low-dimensional codes that significantly outperform traditional linear methods like PCA. By training a multilayer neural network to reconstruct its input through a narrow bottleneck layer, autoencoders discover compact, nonlinear representations that capture the essential structure of high-dimensional data (Bengio et al., 2013).

The autoencoder learns optimal nonlinear compression through symmetric encoder-decoder architecture. The encoder learns mapping $f : \mathbb{R}^{261} \to \mathbb{R}^2$:

$$z = f(x) = \sigma_2(W_2\sigma_1(W_1x + b_1) + b_2) \tag{2}$$

The decoder learns reconstruction $g : \mathbb{R}^2 \to \mathbb{R}^{261}$:

$$\hat{x} = g(z) = \sigma_4(W_4\sigma_3(W_3z + b_3) + b_4) \tag{3}$$

The architecture follows Input: 261 $\to$ Dense(128) $\to$ Dense(64) $\to$ Dense(2) $\to$ Dense(64) $\to$ Dense(128) $\to$ Output: 261, with Mean Squared Error loss function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} ||x_i - g(f(x_i))||^2 \tag{4}$$

Training configuration includes 50 epochs with early stopping (patience=5) (Prechelt, 1998), batch size 32, 20% validation split, and Adam optimizer (Kingma and Ba, 2014) with default learning rate. The Adam optimizer is chosen for its adaptive learning rate properties and robust performance across different problem domains. Early stopping prevents overfitting by monitoring validation loss and halting training when improvement ceases, a crucial regularization technique for neural networks. This architecture follows a good practice, however further experimental structures should be tested.

Figure 4 shows the autoencoder training progression, with both training and validation loss decreasing steadily over epochs. The close alignment between training and validation curves indicates proper model convergence without significant overfitting, validating the chosen architecture and hyperparameters.

[3] This is the UMAP cost function, which is actually the cross entropy. This representaion can be found in the citation, equation (13).

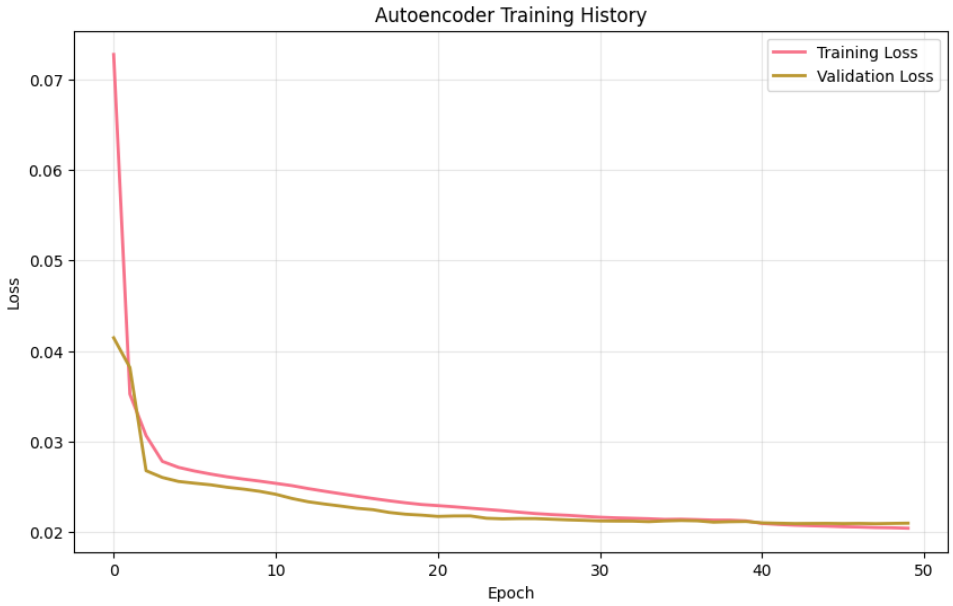



Figure 4: Autoencoder Training History - Training and validation loss curves over epochs, demonstrating convergence and model learning progression.

## 4.4 Similarity Methods Analysis

### 4.4.1 Distance Calculations

For reduced 2D representations, Euclidean distance provides the primary similarity metric:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{2}(x_{i,k} - x_{j,k})^2} \tag{5}$$

Distances are converted to similarity scores using:

$$S(x_i, x_j) = 1 - \frac{d(x_i, x_j)}{d_{max}} \tag{6}$$

where $d_{max}$ represents the maximum pairwise distance in the dataset. This similarity score $S \in [0, 1]$ provides intuitive interpretation: $S > 0.95$ indicates extremely similar cases, $0.90 < S \leq 0.95$ indicates very similar cases suitable for pricing reference, $0.85 < S \leq 0.90$ indicates moderately similar cases useful for risk assessment, and $S \leq 0.85$ indicates limited similarity requiring careful interpretation.

### 4.4.2 DBSCAN Implementation

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) represents a paradigm shift in clustering methodology by identifying clusters based on density connectivity without requiring pre-specified cluster counts. Introduced by Ester et al. (1996), DBSCAN addresses fundamental limitations of traditional clustering algorithms like k-means by discovering clusters of arbitrary shape and explicitly handling noise points. The algorithm's density-based approach makes it particularly suitable for real-world datasets where clusters may have varying shapes and densities, and where outliers are present.

Key mathematical definitions include:

- $\varepsilon$-neighborhood: $N_\varepsilon(p) = \{q \in D | dist(p, q) \leq \varepsilon\}$
- Core Point: $|N_\varepsilon(p)| \geq minPts$

- Density-connected: Points reachable through chain of core points

Optimal $\varepsilon$ is determined using k-distance analysis by computing k-distance for each point (k = minPts = 5), sorting distances in ascending order, and selecting $\varepsilon$ at maximum curvature point in the k-distance plot. This systematic approach to parameter selection, ensuring robust cluster identification. The algorithm's ability to identify noise points (data points that do not belong to any cluster) is particularly valuable in insurance applications where outliers may represent unusual risk profiles requiring special attention.

#### 4.4.3 HDBSCAN Implementation

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) extends DBSCAN by building a hierarchy of clusters across different density levels, addressing one of DBSCAN's primary limitations: its inability to handle clusters of varying densities. The theoretical foundation, established by Campello et al. (2013), introduces the concept of hierarchical density estimates that enable robust cluster extraction across multiple density scales. The widely-used Python implementation (McInnes et al., 2017) provides efficient algorithms for large-scale applications while maintaining the mathematical rigor of the original formulation.

The mathematical framework includes:

1. Mutual Reachability Distance:

$$d_{mreach-k}(a,b) = \max\{core_k(a), core_k(b), d(a,b)\} \tag{7}$$

2. Minimum Spanning Tree construction using mutual reachability distances
3. Cluster Hierarchy building dendrogram by removing edges in order of decreasing weight
4. Stability Selection choosing clusters maximizing stability measure:

$$S(C) = \sum_{p \in C} (\lambda_p - \lambda_{birth}(C)) \tag{8}$$

HDBSCAN provides key advantages including handling varying cluster densities, providing cluster stability scores, identifying hierarchical cluster relationships, and demonstrating robustness to parameter selection. Unlike DBSCAN, which requires careful $\epsilon$ tuning and struggles with varying density clusters, HDBSCAN automatically determines appropriate cluster boundaries at multiple density levels. The stability-based cluster extraction ensures that only robust, persistent clusters are identified, reducing sensitivity to noise and parameter choices. This makes HDBSCAN particularly valuable for insurance portfolio segmentation, where different market segments may exhibit substantially different densities in the feature space.

### 4.5 Scenario Comparison and Results

#### 4.5.1 Baseline Scenario (S1): Standard One-Hot Encoding

The baseline approach applies standard one-hot encoding to all categorical variables without domain-specific weighting. Table 1 summarizes the results of the dimensionality reduction for both UMAP and autoencoder.

| Metric | UMAP | Autoencoder |
|---|---|---|
| Original Dimensions | (3746, 261) | |
| Reduced Dimensions | (3746, 2) | |
| Training Loss | - | 0.0204 (final) |
| Validation Loss | - | 0.0210 (final) |
| Mean (Component 1) | 2.530 | 17.873 |
| Mean (Component 2) | -4.930 | 12.569 |
| Std Dev (Component 1) | 4.687 | 12.400 |
| Std Dev (Component 2) | 10.723 | 9.587 |
| Range (Component 1) | [-6.319, 27.059] | [0.000, 56.883] |
| Range (Component 2) | [-16.149, 20.028] | [0.000, 59.208] |

Table 1: Dimensionality Reduction and Clustering Analysis Results (Scenario 1)

Table 2 summarizes the similarity results based on the Euclidean distance (described in section 4.4.1) for both dinmensionality reduction methods.

| Method | Mean | Std Dev | Max |
|---|---|---|---|
| UMAP Distances | 12.050 | 11.345 | 38.364 |
| Autoencoder Distances | 19.064 | 11.307 | 72.875 |

Table 2: Distance Analysis Results (Scenario 1)

Tables 3 and 4 summarize the similarity results based on the clustering results for both dimensionality reductions and for both clustering methods.

| Method | Clusters | Noise Points | Noise % | Mean Cluster Size |
|---|---|---|---|---|
| DBSCAN | 122 | 202 | 5.4% | 29.0 |
| HDBSCAN | 215 | 758 | 20.2% | 13.9 |

Table 3: Clustering Results based on UMAP (Scenario 1)

| Method | Clusters | Noise Points | Noise % | Mean Cluster Size |
|---|---|---|---|---|
| DBSCAN | 42 | 232 | 6.2% | 9.0 |
| HDBSCAN | 210 | 1317 | 35.2% | 11.6 |

Table 4: Clustering Results based on Autoencoder (Scenario 1)

#### 4.5.2 Expert-Weighted Scenario (S2)

The expert-weighted approach applies domain knowledge to weight categorical features:

```
feature_weights = {
    'quote_family_country': 3.0,      # Geographic location highly important
    'Activity_Loading_Category': 2.0  # Industry risk classification important
}
```

The results of the Expert-Weighted Scenario are summarized in Table 5

| Metric | UMAP | Autoencoder |
|---|---|---|
| Original Dimensions | (3746, 261) | |
| Reduced Dimensions | (3746, 2) | |
| Training Loss | - | 0.0508 (final) |
| Validation Loss | - | 0.0524 (final) |
| Mean (Component 1) | 10.489 | 22.535 |
| Mean (Component 2) | 3.851 | 21.665 |
| Std Dev (Component 1) | 7.075 | 15.571 |
| Std Dev (Component 2) | 7.810 | 14.156 |
| Range (Component 1) | [-10.155, 29.621] | [0.000, 76.580] |
| Range (Component 2) | [-13.247, 23.248] | [0.000, 68.198] |

Table 5: Dimensionality Reduction and Clustering Analysis Results (Scenario 1)

The similarity results can be found in Table 6.

| Method | Mean | Std Dev | Max |
|---|---|---|---|
| UMAP Distances | 13.006 | 7.276 | 41.679 |
| Autoencoder Distances | 26.084 | 14.334 | 91.595 |

Table 6: Distance Analysis Results (Scenario 1)

Tables 7 and 8 summarize the similarity results based on the clustering results for both dimensionality reductions and for both clustering methods.

| Method | Clusters | Noise Points | Noise % | Mean Cluster Size |
|---|---|---|---|---|
| DBSCAN | 147 | 187 | 5.0% | 24.2 |
| HDBSCAN | 182 | 337 | 9.0% | 18.7 |

Table 7: Clustering Results based on UMAP (Scenario 2)

| Method | Clusters | Noise Points | Noise % | Mean Cluster Size |
|---|---|---|---|---|
| DBSCAN | 64 | 221 | 5.9% | 55.1 |
| HDBSCAN | 225 | 1154 | 30.8% | 11.5 |

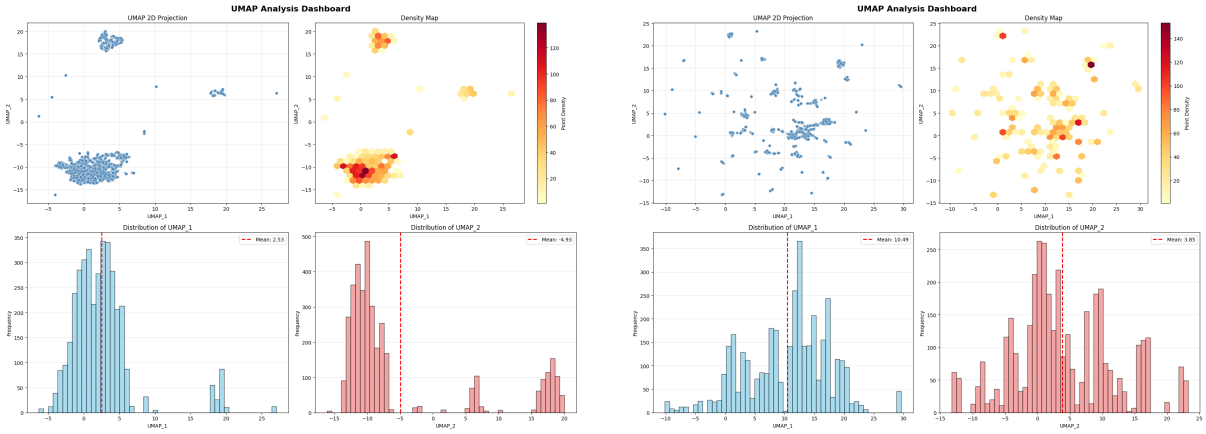
Table 8: Clustering Results based on autoencoder (Scenario 2)

#### 4.5.3 Visualizations

Figure 5 presents a side-by-side comparison of UMAP projections for both scenarios. The expert-weighted approach (Scenario 2) exhibits more distinct clustering patterns and improved separation between different insurance market segments, particularly evident in the geographic and industry-based groupings. The enhanced structure validates the benefit of incorporating domain expertise into the feature engineering process.

Figure 6 illustrates the distribution of pairwise distances for both dimensionality reduction methods across scenarios. The expert-weighted approach (Scenario 2) shows more concentrated distance distributions, indicating improved consistency in similarity measurements and better discrimination between similar and dissimilar cases.

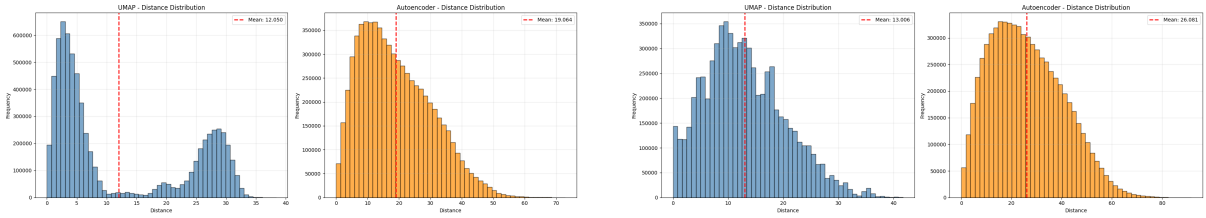
Figure 7 demonstrates the clustering analysis results for the expert-weighted scenario. DBSCAN produces larger, more cohesive clusters suitable for high-level market segmentation, while HDBSCAN identifies more granular clusters that capture subtle risk distinctions within the insurance portfolio. The color-coding reveals distinct geographic and risk-based patterns that align with business intuition.




(a) Scenario 1: Standard encoding

(b) Scenario 2: Expert-weighted encoding

Figure 5: UMAP 2D Projections Comparison - Scenario 1 (left) shows the baseline one-hot encoding results, while Scenario 2 (right) demonstrates the expert-weighted approach with improved cluster separation and structure.




(a) Scenario 1: Distance distributions

(b) Scenario 2: Distance distributions

Figure 6: Distance Distribution Analysis - Comparison of UMAP and autoencoder distance distributions between scenarios, showing the statistical properties of similarity measurements.

### 4.5.4 Performance Metrics Evaluation

Table 9 presents a comprehensive comparison of key metrics across both scenarios and dimensionality reduction methods.

| Metric | S1 UMAP | S2 UMAP | S1 AE | S2 AE |
|---|---|---|---|---|
| Mean Distance | 12.050 | 13.006 | 19.064 | 26.084 |
| Distance Std | 11.345 | 7.276 | 11.307 | 14.334 |
| Cluster Count (DBSCAN) | 122 | 147 | 42 | 64 |
| Noise Points (DBSCAN) | 5.4% | 5.0% | 6.2% | 5.9% |
| Mean Cluster Size | 29.0 | 24.2 | 83.7 | 55.1 |

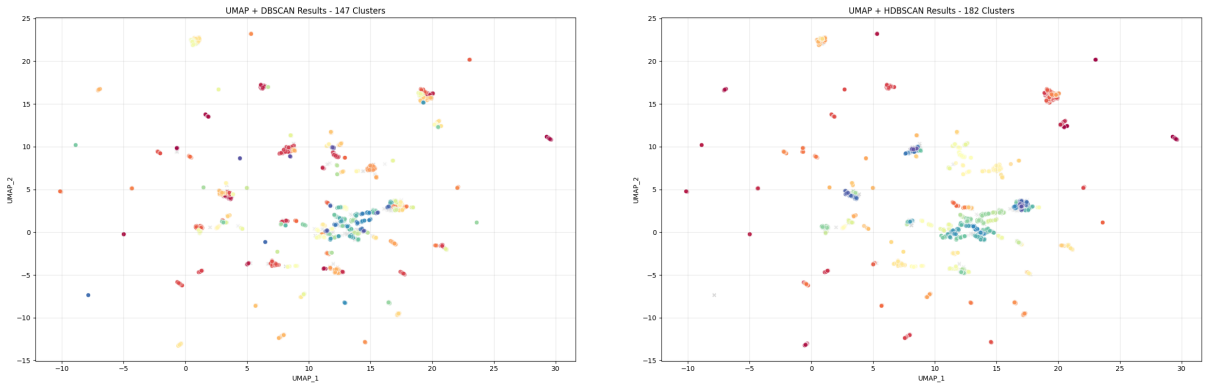
Table 9: Scenario Performance Comparison

Key observations:

1. **Improved Separation**: Scenario 2 shows reduced distance standard deviation in UMAP (7.276 vs 11.345), indicating more consistent similarity scoring. This improvement aligns with UMAP's theoretical properties of preserving both local and global structure.

2. **Enhanced Clustering Structure**: More clusters identified in Scenario 2 (UMAP: 147 vs 122), suggesting better discrimination of market segments. The density-based approach of DBSCAN naturally identifies these segments without requiring pre-specification of cluster counts.

3. **Reduced Noise**: Scenario 2 UMAP shows lower noise percentage (5.0% vs 5.4%), indicating improved cluster cohesion and more robust cluster assignments.




(a) DBSCAN clustering results

(b) HDBSCAN clustering results

Figure 7: Clustering Analysis Results for Scenario 2 - DBSCAN (left) identifies 147 density-based clusters with 5.0% noise points, while HDBSCAN (right) reveals 182 hierarchical clusters with 9.0% noise points, demonstrating different granularities of market segmentation.

4. **Feature Weighting Impact**: Expert weighting produces more granular clustering, particularly beneficial for geographic and industry-based risk segmentation. This demonstrates the value of incorporating domain knowledge into similarity metrics.

Similar analysis for autoencoders and HDBSCAN can be performed also. The results are ready, but the deep analysis is left for future work. However, in the Appendix 6.3, visualization of the results can be found.

## 4.6 Business Application of Machine Learning Models

### 4.6.1 Underwriting Process Integration

The developed system integrates into the underwriting workflow through automated quote ingestion and processing, real-time similarity computation against historical database, presentation of top $n$ (currently set to 10) similar cases with business context, and comprehensive decision support including loading factors, claim experience, and pricing history.

Traditional underwriting process time (2-3 hours) potentially can be to significantly through automated case identification, standardized risk comparison framework, consistent similarity metrics, and historical pattern recognition beyond human capability.

### 4.6.2 Example

Figure 8 illustrates a practical application of the similarity search system, showing how the algorithm identifies and ranks similar historical cases for a new insurance quote. The visualization displays similarity scores, distances, and key business attributes that enable underwriters to quickly assess comparable cases and make informed pricing and risk assessment decisions.

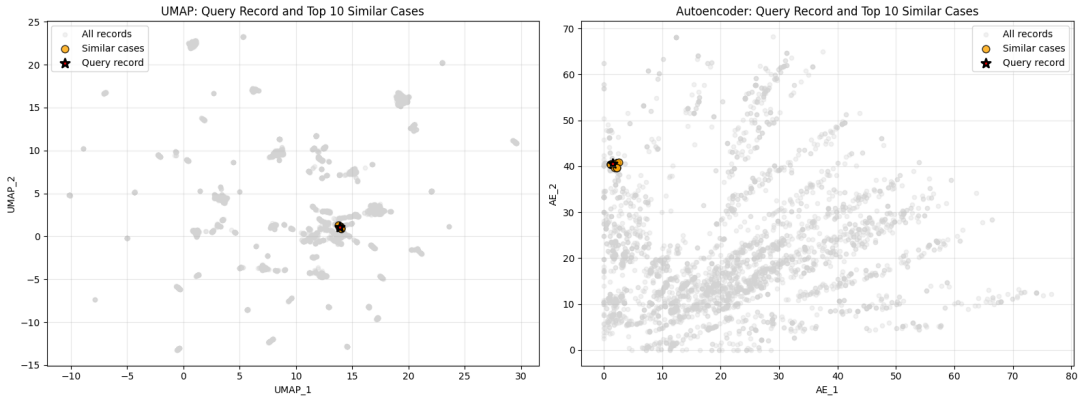



Figure 8: Practical Similarity Search Example - Demonstration of the system identifying the top 10 most similar historical cases for a new insurance quote, showing similarity scores, distances, and relevant business attributes for underwriter decision support.

# 5 Limitations and Future Work

In this section, we outline the key limitations of the current Proof of Concept and the practical steps to address them. While the RAG module and similarity engine demonstrate clear potential to streamline underwriting, their impact depends on broader document coverage, stronger data integration, and rigorous validation with underwriters. We therefore prioritize enhancements that improve reliability, explainability, and fit-for-purpose adoption: expanding data sources, hardening storage and APIs, refining recommendations with transparent scoring, and closing the loop with production-grade UI and process automation. Together, these initiatives form a pragmatic roadmap to move from promising prototype to trusted, governed, and scalable capability.

**RAG**

- Ingestion scope: expand to additional internal documents beyond the current guideline focus.
- Source diversification: incorporate curated emails and textual inputs from databases.
- Data persistence: evaluate architectures for permanent storage, embedding strategies, and durable API connections across environments.

**Final recommendation system**

- Develop a code-based history parser to enable end-to-end automation.
- Build a comprehensive comparison engine.
- Introduce a scoring mechanism that blends document-derived insights, historical textual knowledge, and renewal history into a final recommendation.

**System upgrades**

- Enable direct querying of the first-stage pricing database for real-time data access.
- Automate the transmission of quotes with recommended parameters/assumptions to the Pricing Tool.

**Similarity engine**

- Extend validation with broader scenario testing and sensitivity analyses.
- Verify input feature sets and systematically test outputs.

- Conduct qualitative assessments with underwriters (surveys and feedback).
- Extend feature coverage to additional coverages and incorporate experience/credibility considerations.
- Perform hyperparameter tuning for the autoencoder and other components.
- Research of clustering evaluation techniques.
- Exploring additional machine learning approaches for the classification task: while clustering and similarity-based methods can supply ground-truth signals, lightweight, fast tree-based models – such as XGBoost – can be effectively employed for categorization to complement and accelerate the process.

**UI**

- Advance PoC UI towards production-ready interfaces with rigorous usability testing and accessibility.

**Project management**

- Define implementation roadmap.
- Quantify the business case (freed up FTE that can be shifted towards new business)
- Implement back-testing to identify potential historical errors and estimate business impact.
- Budget and cost calculations needs for development and maintenance phases.

# 6 Appendix

## 6.1 Summary of Literature

### 6.1.1 Retrieval-Augmented Generation Systems

Retrieval-Augmented Generation has emerged as a critical technique for enhancing large language model capabilities with external knowledge bases. Lewis et al. (2020) introduced the foundational RAG architecture, demonstrating significant improvements in knowledge-intensive tasks through the combination of parametric and non-parametric knowledge sources.

Recent advances in RAG architecture have focused on improving retrieval quality through dense passage retrieval (Karpukhin et al., 2020), advanced embedding techniques (Reimers and Gurevych, 2019), and query enhancement strategies (Wang et al., 2023). However, limited work has been done on enterprise-specific RAG implementations that prioritize data privacy, domain specialization, and production scalability.

The Atlas model Izacard et al. (2023) demonstrated few-shot learning capabilities with retrieval augmentation, while recent surveys (Gao et al., 2023; Li et al., 2023) have highlighted the growing importance of retrieval-augmented approaches in practical AI applications.

### 6.1.2 Vector Database Technologies and FAISS

The landscape of vector similarity search has been dominated by specialized databases and libraries designed for high-dimensional embeddings. FAISS (Facebook AI Similarity Search) (Johnson et al., 2019, 2017) has emerged as a leading solution for efficient similarity search, providing both CPU and GPU implementations capable of handling billion-scale datasets.

Recent developments in FAISS (Douze et al., 2024) have enhanced its capabilities for production deployment, including improved indexing algorithms, memory optimization, and distributed search capabilities. On one hand the choice of FAISS over cloud-based vector databases represents a strategic decision for data privacy and control, particularly relevant for sensitive insurance applications. However in the future, we are planning to use internal infrastructure, hence the security in the cloud will be given. On the other hand though, the current phase of the project (PoC) aims to works fast on a small example, and FAISS provides good CPU implementation.

### 6.1.3 LangChain Agent Framework

The development of AI agent frameworks has accelerated with the introduction of LangChain (Chase, 2022; LangChain Inc., 2023), which provides sophisticated orchestration capabilities for AI agents, enabling complex workflows and tool integration. LangChain's agent framework (LangChain Inc., 2024) offers standardized patterns for tool selection, execution, and response formatting.

The transformer architecture (Vaswani et al., 2017) and subsequent developments like BERT (Devlin et al., 2019) and GPT models (Brown et al., 2020; OpenAI, 2023) have provided the foundation for modern language model capabilities that enable sophisticated agent behaviors in specialized domains like insurance underwriting.

### 6.1.4 Enterprise AI Integration

Enterprise deployment of AI systems requires careful consideration of security, scalability, and integration requirements. Azure OpenAI Service (Microsoft Corporation, 2024) provides enterprise-grade access to advanced language models while maintaining compliance and security standards required for financial services applications.

The combination of cloud-based LLM services with on-premises vector storage represents a hybrid architecture that balances performance, privacy, and scalability requirements typical of insurance industry applications.

## 6.2 Technical Implementation Details for Similarity Engine

### 6.2.1 Random Seed Configuration

```
import numpy as np
import tensorflow as tf
import random

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)
tf.keras.utils.set_random_seed(42)
random.seed(42)
```

### 6.2.2 UMAP Implementation

```
import umap.umap_ as umap

reducer = umap.UMAP(
    n_components=2,
    n_neighbors=15,
    min_dist=0.1,
    metric='euclidean',
    random_state=42
)
```

### 6.2.3 Autoencoder Architecture

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(128, activation='relu', input_shape=(261,)),
    Dense(64, activation='relu'),
    Dense(2, activation='linear'),  # Bottleneck layer
    Dense(64, activation='relu'),
    Dense(128, activation='relu'),
    Dense(261, activation='sigmoid')
])
```

### 6.2.4 Feature Weighting Implementation

```
def apply_feature_weights(data, weights):
    """Apply domain expert weights to categorical features"""
    weighted_data = data.copy()

    for feature_prefix, weight in weights.items():
        # Find columns matching prefix
        matching_cols = [col for col in data.columns
                         if col.startswith(feature_prefix)]

        # Apply weight to matching columns
        weighted_data[matching_cols] *= weight

    return weighted_data
```

## 6.3 Clustering result visualizations

### 6.3.1 Scenario 1

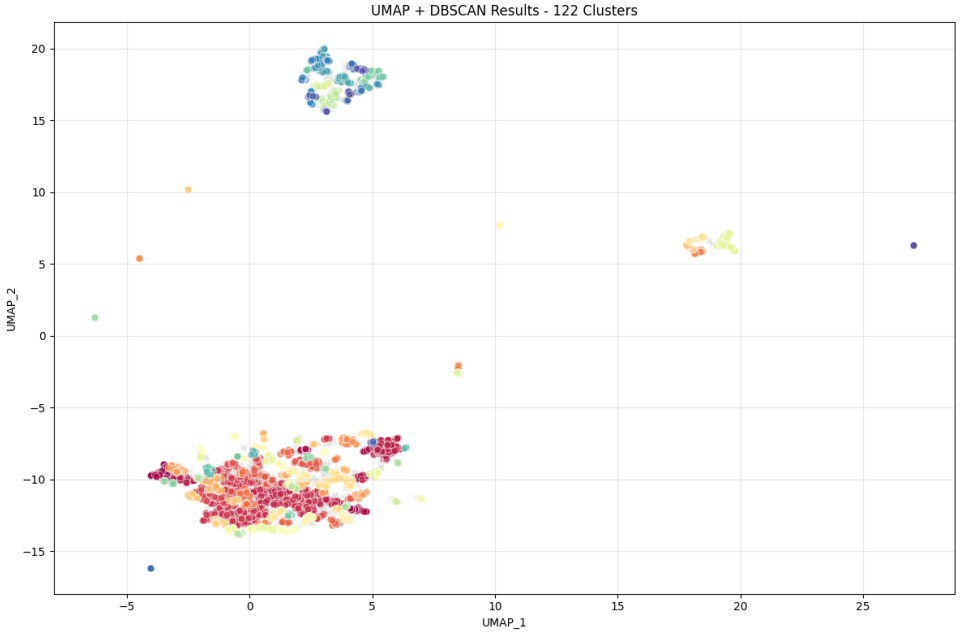



Figure 9: UMAP - DBSCAN - Scenario 1

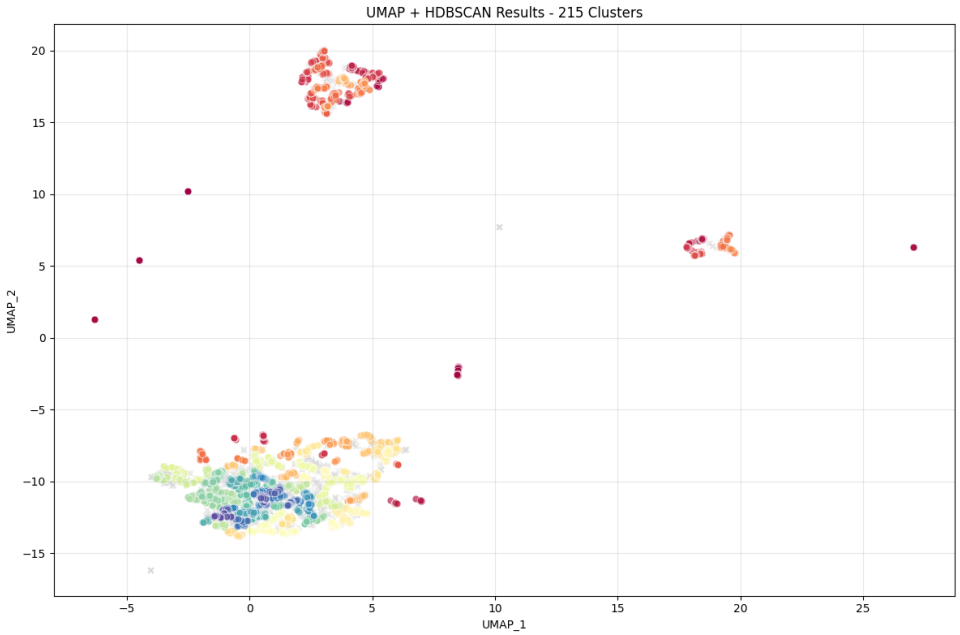



Figure 10: UMAP - HDBSCAN - Scenario 1

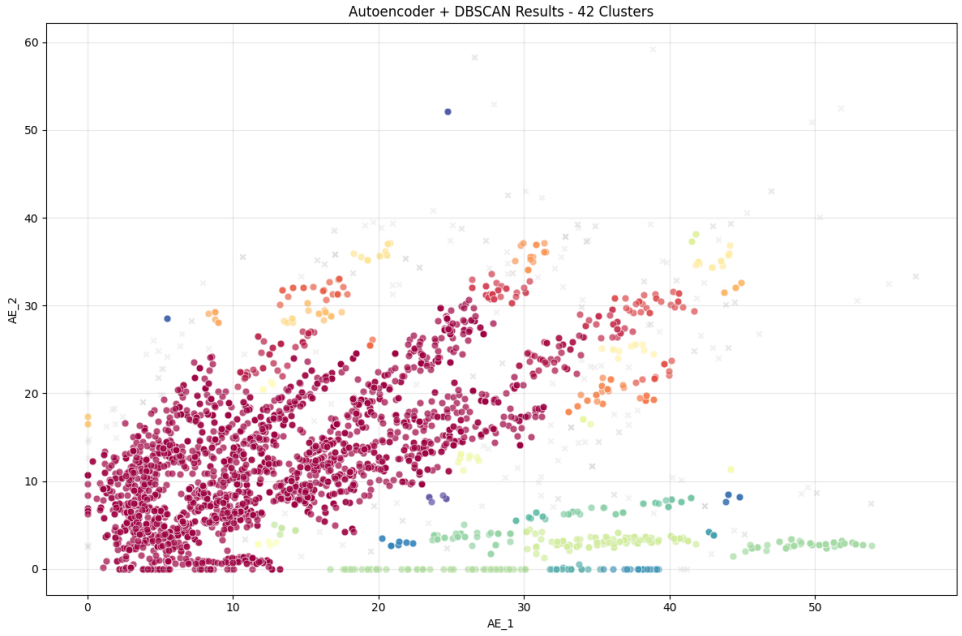



Figure 11: AE - DBSCAN - Scenario 1

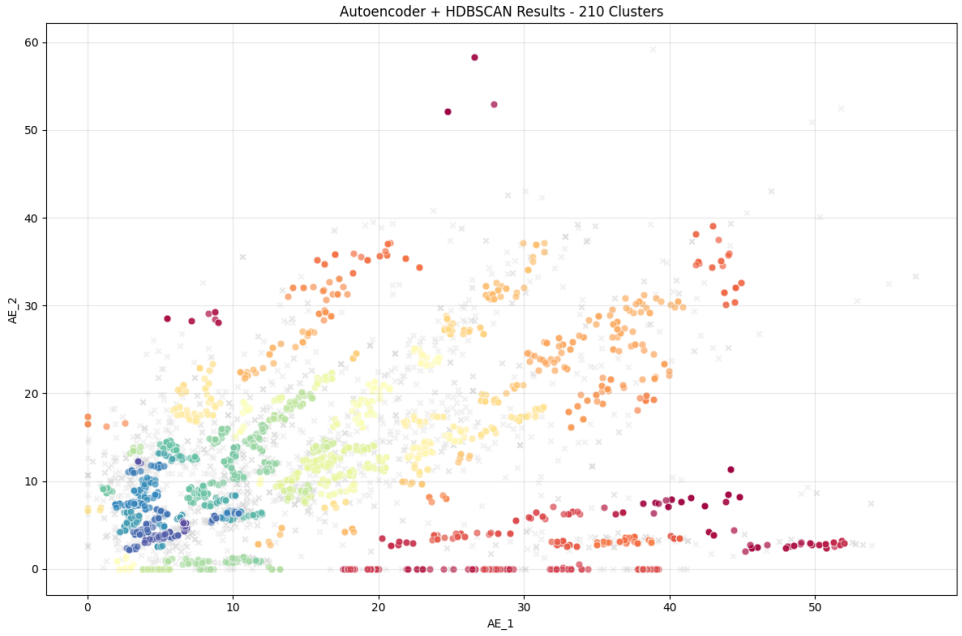



Figure 12: AE - HDBSCAN - Scenario 1

### 6.3.2 Scenario 2

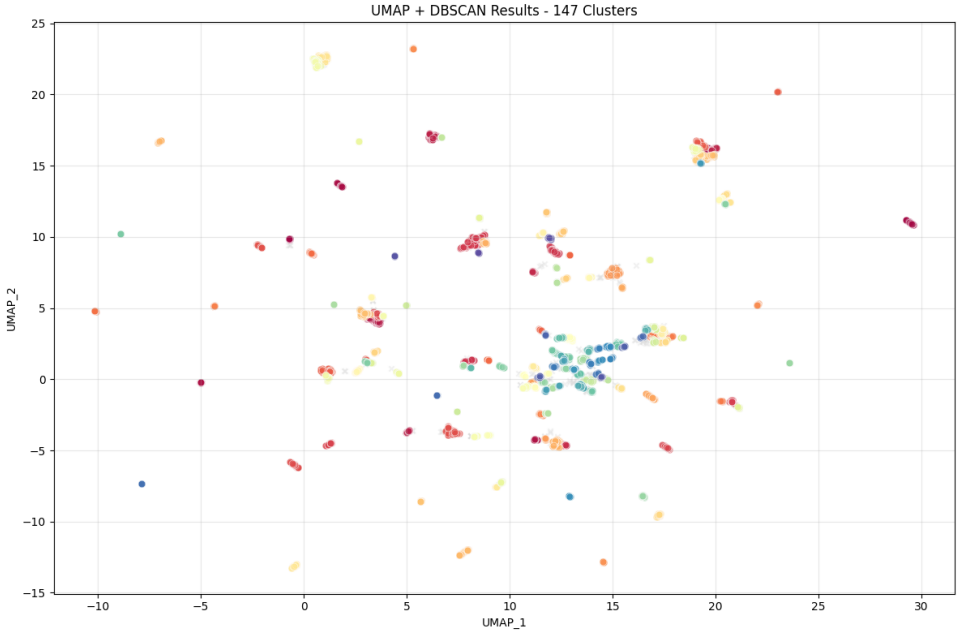



Figure 13: UMAP - DBSCAN - Scenario 2

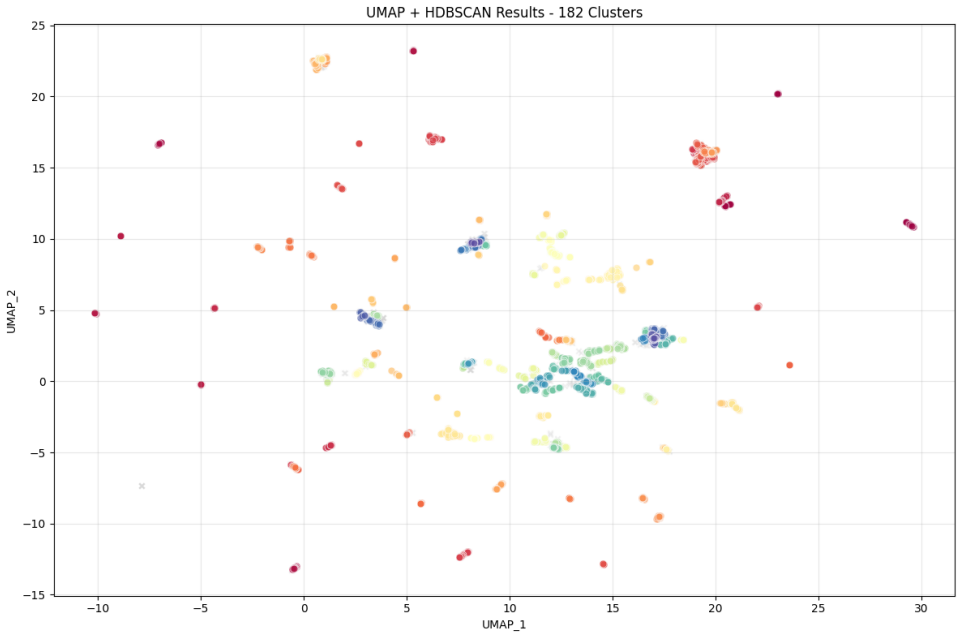



Figure 14: UMAP - HDBSCAN - Scenario 2

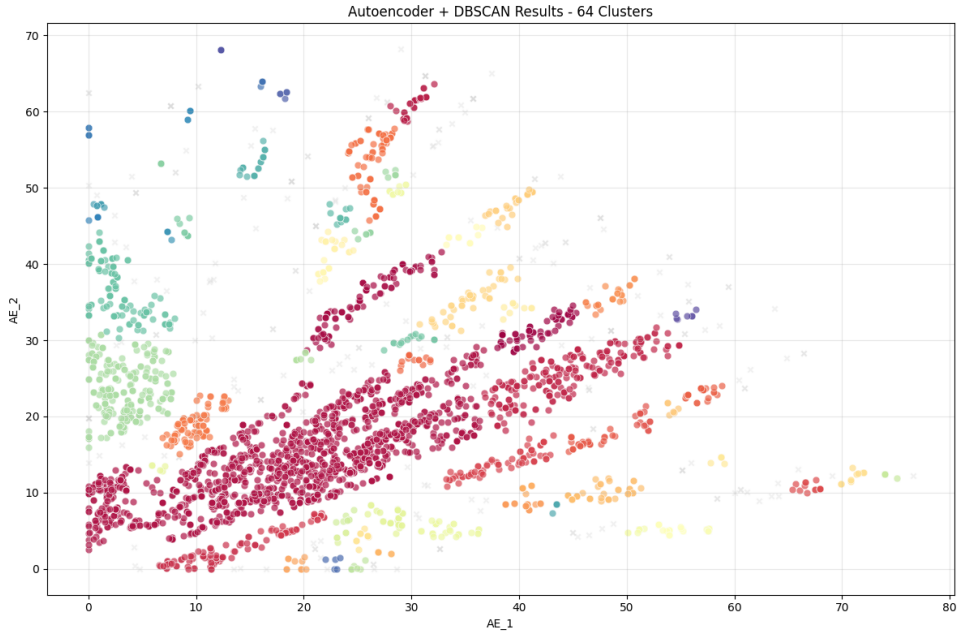



Figure 15: AE - DBSCAN - Scenario 2

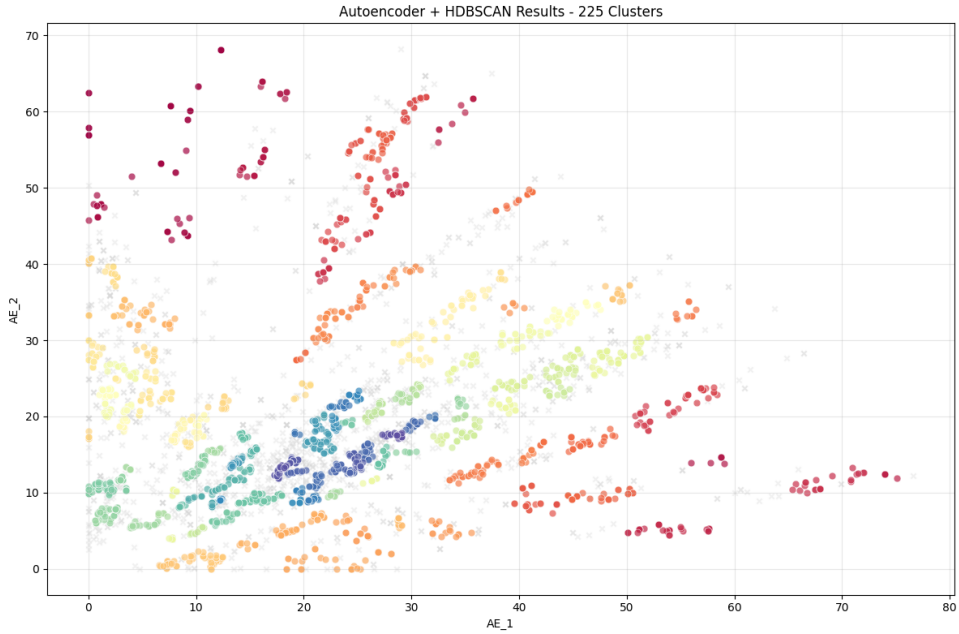



Figure 16: AE - HDBSCAN - Scenario 2

## 6.4 Performance Metrics Summary for Similarity Engine

Table 10 provides a comprehensive overview of processing performance metrics.

| **Process** | **Processing Time** |
|---|---|
| Data Loading | $< 30$ seconds |
| Feature Engineering | $< 60$ seconds |
| UMAP Reduction | 2-3 minutes |
| Autoencoder Training | 3-5 minutes |
| Similarity Analysis | $< 60$ seconds |
| Total Pipeline | 5-10 minutes |

Table 10: Processing Performance Summary

# References


Adams, D. (1979). *The Hitchhiker's Guide to the Galaxy*. Pan Books, London. The answer "42" is discussed in Chapter 27.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Campello, R. J. G. B., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*, volume 7819 of *Lecture Notes in Computer Science*, pages 160–172. Springer.

Chase, H. (2022). Langchain: Building applications with llms through composability. `https://github.com/langchain-ai/langchain`. GitHub repository.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. (2024). The faiss library. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5085–5096.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. (2023). Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

Johnson, J., Douze, M., and Jégou, H. (2017). Faiss: A library for efficient similarity search. `https://github.com/facebookresearch/faiss`. Facebook Research.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. In *IEEE Transactions on Big Data*, volume 7, pages 535–547. IEEE.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Published in ICLR 2015.

LangChain Inc. (2023). Langchain: Building applications with llms through composability. *arXiv preprint*.

LangChain Inc. (2024). Langchain documentation: Agents and tools. `https://python.langchain.com/docs/modules/agents/`. Retrieved November 2024.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Li, P., Zhang, Z., Ren, J., Jiang, Y., and Zhang, H. (2023). Retrieval-augmented generation for ai-generated content: A survey. *Proceedings of Machine Learning Research*, 202:1–15.

McInnes, L., Healy, J., and Astels, S. (2017). hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205.

McInnes, L., Healy, J., and Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*. Published in Journal of Open Source Software, 3(29), 861.

Microsoft Corporation (2024). Azure openai service documentation.

OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Prechelt, L. (1998). Early stopping - but when? *Neural Networks: Tricks of the Trade*, 1524:55–69.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, X., Zhu, Y., and Zhang, S. (2023). Query rewriting for retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5722–5735.