From RAG to Richness in Industry Classification

What is Industry Classification?

Industry classification is the process of categorizing companies, businesses, or economic activities into structured groups based on their primary business functions, products, or services. It helps **governments, financial institutions, investors, and analysts** systematically compare businesses within a specific market.

Why is Industry Classification required

- **Industry Overlap & Diversification**
    - Many modern businesses operate across multiple industries (e.g., **Amazon = E-commerce + Cloud Computing + AI**).
    - Classifying companies with **diverse business models** accurately is difficult.
- **Lack of Standardization Across Regions**
    - **NAICS vs. GICS vs. SIC** → Different classification systems create inconsistencies in cross-border comparisons.
    - A company might fall into different categories depending on the framework used.
- **Emerging & Disruptive Industries**
    - **AI, blockchain, and quantum computing** sectors evolve faster than classification systems can update.
    - Many new industries don't fit neatly into traditional categories.
- **Subjectivity & Human Error**
    - Industry classification often **relies on human judgment**, leading to inconsistencies.
    - **Misclassification** can impact financial analysis, regulatory compliance, and investment decisions.
- **Data Accuracy & Granularity**
    - Some businesses have **insufficient publicly available data**, making precise classification challenging.
    - Small companies or **startups** often get misclassified due to **lack of detailed reporting**.
- **Industry Convergence & Cross-Sector Businesses**
    - Many industries are **merging** (e.g., FinTech = Finance + Technology), complicating classification efforts.
    - **Tesla**: Auto industry or renewable energy? **Meta**: Social media or AI?

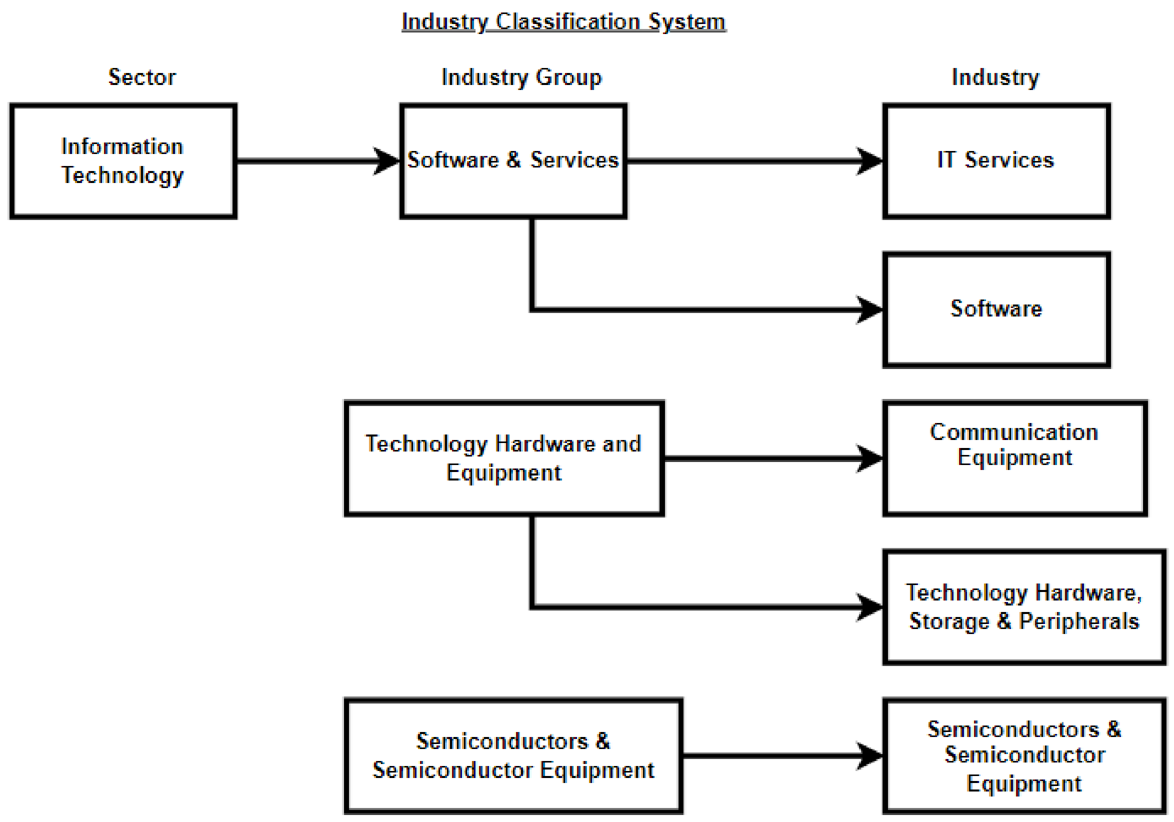
Tech Solution

- **AI & ML for Automated Industry Classification** – Using **machine learning** to classify businesses dynamically based on evolving data.
- **Regular Updates to Classification Standards** – Ensuring industry frameworks reflect **emerging business trends**.
- **Multi-Tier Classification Systems** – Allowing companies to belong to **multiple industries**, providing a more nuanced classification.
- **Improved Regulatory & Reporting Standards** – Mandating detailed company reporting for **more accurate classification**.

Common Industry Classification Standards

Several standards exist to classify industries, including:

- **North American Industry Classification System (NAICS)** – Used in the U.S., Canada, and Mexico.
- **Standard Industrial Classification (SIC)** – An older classification system still used in some sectors.
- **Global Industry Classification Standard (GICS)** – Commonly used in financial markets, especially for stock categorization.
- **Industry Classification Benchmark (ICB)** – Used for global financial and economic analysis.


Industry Classification System
Sector
Industry Group
Industry
Information Technology
Software & Services
IT Services
Software
Technology Hardware and Equipment
Communication Equipment
Technology Hardware, Storage & Peripherals
Semiconductors & Semiconductor Equipment
Semiconductors & Semiconductor Equipment

Enhancing Industry Classification with Retrieval-Augmented Generation (RAG)

Problem Statement

Ramp’s mission is to help customers save time and money, which requires a precise understanding of their industries across various initiatives, from compliance to portfolio monitoring and sales targeting. Holver, industry classification poses significant challenges due to fuzzy industry boundaries, lack of ground truth for validation, and sparse, non-uniform data distributions. For instance, a customer like Wizehire, a hiring platform for small businesses, was classified under a broad category of "Professional Services," which encompassed a wide range of businesses, from law firms to consulting firms. This lack of specificity made it difficult for

Ramp's Sales and Marketing teams to tailor their approach, and for the Risk team to accurately assess credit risk and compliance needs. Additionally, converting internal classifications into SIC and NAICS codes involved complex many-to-many mappings, sometimes requiring a single category to be mapped to 50+ NAICS codes. To address these challenges, I developed an in-house industry classification model using Retrieval-Augmented Generation (RAG), improving both customer insights and operational efficiency.

Proposed Solution

I tamed this complexity by migrating all Ramp industry classification to NAICS codes. This allows internal teams to have a consistent, expressive taxonomy while also enabling easier communication with external partners who Ire already using NAICS codes.

For NAICS code: 561311, the full hierarchy is displayed below:

Sector	56	Administrative and Support and Waste Management and Remediation Services
Sub sector	561	Administrative and Support Services
Industry Group	5613	Employment Services
Industry	56131	Employment Placement Agencies and Executive Search Services
National Industry	561311	Employment Placement Agencies

At Ramp, I needed a flexible classification system capable of providing both granular and high-level industry labels to accommodate various business needs. To transition to NAICS classification, I required a model that could accurately predict six-digit NAICS codes for all businesses. While third-party solutions offered quick, general performance, they lacked the ability to handle my complex, proprietary data, leading us to develop an in-house classification model.

My Approach: Building a Classification Model with RAG

I built my system using Retrieval-Augmented Generation (RAG), which operates in three key stages:

1. Generate Text Embeddings → Create vector representations for both queries and the knowledge base.
2. Compute Similarity Scores → Retrieve the most relevant NAICS codes based on similarity.
3. LLM-Based Prediction → Use a language model to make the final selection from retrieved recommendations.

Key Advantage → This approach constrains the LLM's output to valid NAICS codes, effectively converting an open-ended task into a structured multiple-choice problem.

Performance Evaluation: Metrics for Each Stage

Since this is a multi-stage system, I designed stage-specific evaluation metrics to ensure accurate classification.

Stage 1: Retrieval Performance

- Metric: Accuracy at k (acc@k) → Measures how often the correct NAICS code appears in the top k recommendations.
- Why it matters: If the correct NAICS code is not in the top k results, the LLM cannot select it, limiting overall accuracy.

Stage 2: LLM-Based Final Prediction

- Metric: Custom Fuzzy-Accuracy Score → Accounts for the hierarchical structure of NAICS codes.
- Scoring Logic:
    - Predictions partially correct in hierarchy (e.g., 123499 instead of 123456) receive higher scores.
    - Predictions that are completely incorrect (e.g., 999999) receive lower scores.

By integrating RAG-based retrieval with LLM-driven classification, I improved:

- Accuracy → Ensuring the correct NAICS codes are retrieved and selected.
- Interpretability → Providing structured industry classification.
- Reliability → Reducing misclassification errors in business categorization.

Generating Recommendations

Generating recommendations involves identifying the most relevant items from the knowledge base (NAICS codes) given a query (business). This part of the system has a variety of hyperparameters to choose:

- Knowledge base field to embed
- Query field to embed
- Embedding model
- Number of recommendations

For each parameter there are tradeoffs to consider. For example, certain business attributes may be more informative than others but may have higher missing rates. Additionally, different embedding models have different resmyce requirements that don't necessarily correlate with performance on the specific data I have.

In the end, I profiled the performance of different configurations and created `acc@k` curves. Note that I can't determine the optimal number of recommendations to generate without considering the downstream LLM performance. If I naively optimize for `acc@k` I would end up with a system that just recommends the whole knowledge base (guaranteed that the correct label is present if I recommend all possible labels).

I found that optimizations in this stage lead to significant performance boosts of up to 60% in `acc@k`. I also identified economical embedding models that could be used in production without sacrificing performance compared to the largest models.

Predictions Selection

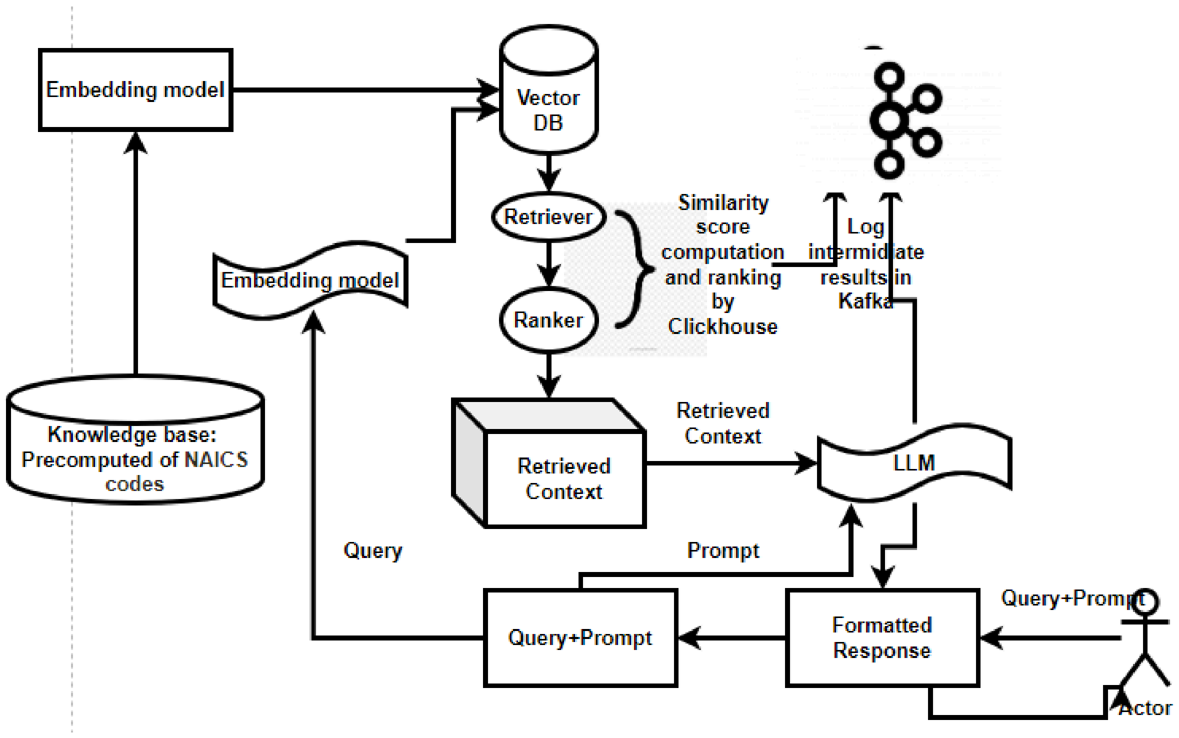
The second stage of the RAG system involves selecting a final prediction from the recommendations using an LLM. This part also has a variety of hyperparameters to choose:

- Number of recommendations
- Fields to include in the prompt (business and knowledge base)
- Prompt variations
- Number of prompts
- Structured output class and fields

Just like the first stage, there are a number of tradeoffs to consider. For example, including more recommendations in the prompt gives the LLM a better chance at finding the correct code, but it also increases the context size and can lead to degraded performance if the LLM is unable to focus on the most relevant recommendations. Likewise, longer or more descriptive information can help the LLM better understand a business or a NAICS code, but will also greatly increase the context size.

In the end I chose a two-prompt system to get the best of both worlds. In the first prompt I include many recommendations but don't include the most specific descriptions, asking the LLM to return a small list of the most relevant codes. In the second prompt, I then ask the LLM to choose the best one and provide more context for each code. For each parameter I searched, I found a `5%-15%` improvement in fuzzy accuracy after optimization.

Solution Design



1. Knowledge base embeddings are pre-computed and stored in Clickhouse for fast retrieval of recommendations using similarity scores.
2. Log intermediate results using Kafka so that I can diagnose pathological cases and iterate on prompts.

Although RAG helps constrain LLM outputs, I also have added guardrails. While hallucinations are generally negative, I've also found cases where the LLM predicts the correct code despite it not being present in the recommendations. To filter out just "bad" hallucinations, I validate that the output NAICS codes from each LLM prompt are valid.

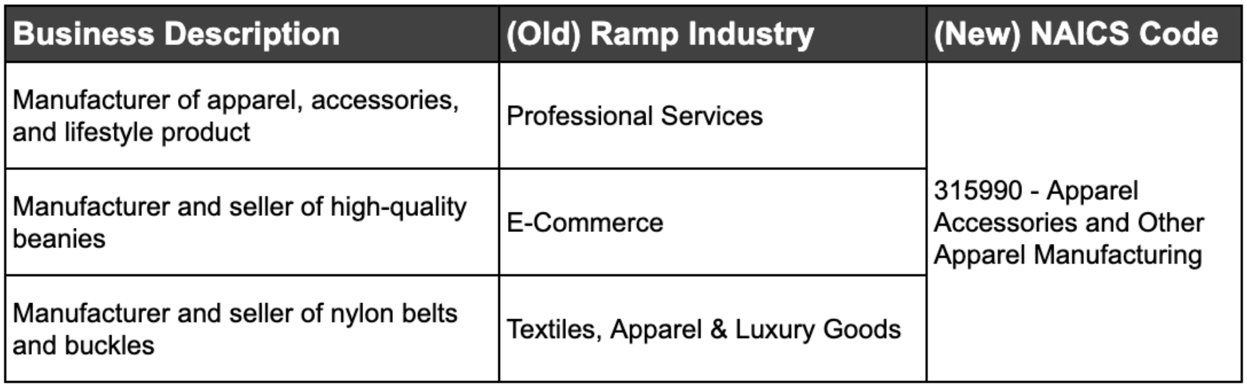
I have greater control over the algorithm, allowing us to:

1. Fine-Tune Hyperparameters → I can tweak dozens of hyperparameters to address concerns as they arise, ensuring optimal performance.

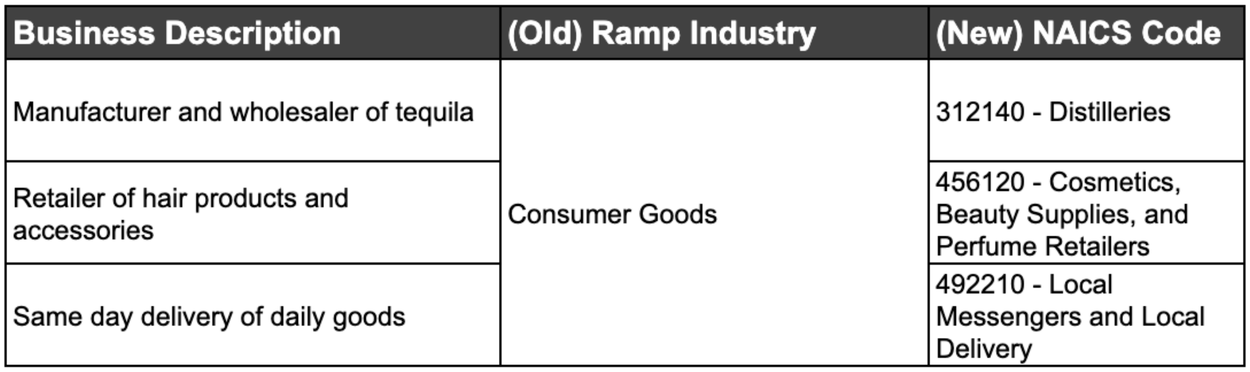
2. Diagnose & Optimize Performance → Since I log all intermediate steps, I can pinpoint issues at retrieval, re-ranking, or prediction stages and adjust the model dynamically for latency, cost, or accuracy.

3. Audit & Interpret Decisions → Unlike third-party solutions, I can audit model predictions by asking the LLM for justifications, improving transparency and trust in decision-making

Sample Industry Classifications



Business Description	(Old) Ramp Industry	(New) NAICS Code
Manufacturer of apparel, accessories, and lifestyle product	Professional Services	315990 - Apparel Accessories and Other Apparel Manufacturing
Manufacturer and seller of high-quality beanies	E-Commerce	
Manufacturer and seller of nylon belts and buckles	Textiles, Apparel & Luxury Goods	



Business Description	(Old) Ramp Industry	(New) NAICS Code
Manufacturer and wholesaler of tequila	Consumer Goods	312140 - Distilleries
Retailer of hair products and accessories		456120 - Cosmetics, Beauty Supplies, and Perfume Retailers
Same day delivery of daily goods		492210 - Local Messengers and Local Delivery

System Design

RAG-Based Industry Classification: End-to-End Pipeline

I. Data Pipeline

1. Data Acquisition & Storage
   - Tools: Snowflake, AWS S3, Azure Data Lake
2. Document Processing
   - Tools: Apache Tika, Unstructured.io, PDFMiner
3. Text Preprocessing
   - Tools: spaCy, NLTK, HuggingFace Transformers
4. Chunking & Organization
   - Tools: LangChain, LlamaIndex, Haystack

II. Feature Pipeline

1. Embedding Generation

- Tools: Sentence-Transformers, OpenAI Embeddings, Cohere Embed
- Process:
    - Generate document embeddings using domain-adapted models
    - Create specialized embeddings for industry terminology
    - Experiment with hybrid embedding approaches

2. Vector Database Implementation

- Tools: ClickHouse
- Process:
    - Index embeddings with industry-specific metadata
    - Configure vector similarity measures (cosine, dot product)
    - Implement efficient filtering by industry hierarchy

3. Feature Optimization

- Tools: FAISS, Optimum, Nvidia TensorRT
- Process:
    - Apply dimensionality reduction techniques
    - Optimize for retrieval speed vs. accuracy tradeoffs
    - Implement quantization for efficient storage

III. Training Pipeline

1. Training Data Preparation

- Tools: AWS Sagemaker Studio
- Process:
    - Create gold-standard industry classifications
    - Generate synthetic examples covering edge cases
    - Implement data augmentation for industry variations

2. Model Development

- Tools: PyTorch, TensorFlow, Hugging Face Transformers
- Process:
    - Fine-tune industry-specific classification models
    - Develop hybrid retrieval+classification architectures
    - Train evaluation models for retrieval quality assessment

3. Experimentation & Evaluation

- Tools: Weights & Biases, MLflow

- Process:
    - Track experiments with different model architectures
    - Measure performance metrics (precision, recall, F1 by industry)
    - Conduct ablation studies for feature importance

IV. Inference Pipeline

1. Query Processing

- Tools: FastAPI
- Process:
    - Implement scalable API endpoints for classification requests
    - Optimize for batch and streaming classifications
    - Cache frequent queries for performance

2. RAG Orchestration

- Tools: LangChain, LlamaIndex
- Process:
    - Implement multi-stage retrieval strategy
    - Incorporate hierarchical industry classification awareness
    - Apply re-ranking based on industry relevance

3. LLM Integration

- Tools: OpenAI API, Anthropic Claude, Llama 3, Mistral
- Process:
    - Prompt engineering for industry classification
    - Chain-of-thought reasoning for classification justification
    - Hybrid classification approaches with confidence scoring

4. Feedback Loop & Monitoring

- Tools: Prometheus, Grafana, Great Expectations
- Process:
    - Monitor classification accuracy and drift
    - Collect human feedback on edge cases
    - Implement continuous retraining pipeline

V. Deployment & Production

1. Infrastructure

- Tools: Kubernetes, Docker, NVIDIA Triton
- Process:
    - Containerize all pipeline components

    - Implement auto-scaling for variable load
    - Optimize GPU utilization for embedding generation

2. CI/CD Pipeline

- Tools: GitHub Actions, Jenkins, ArgoCD
- Process:
    - Automate testing and deployment
    - Implement canary deployments for model updates
    - Version control for models and embeddings

3. Observability

- Tools: Elastic Stack, DataDog, New Relic
- Process:
    - Monitor end-to-end latency and throughput
    - Track classification quality metrics in production
    - Alert on performance degradation or data drift

This comprehensive pipeline combines state-of-the-art RAG techniques with specialized industry classification components to deliver accurate, explainable, and scalable industry classifications.

Wrap Up
Ultimately, my model lead to increased accuracy in industry classification with full control over updates, tuning, and costs.