# Extraction and Prediction of Compounds Migration between Containers and Contents

## Master 2 Data Science


## Gabriel WARDE

### URGO RID, Chenôve, France


From 24/03/2025 To 23/09/2025


**Company Supervisor: Bertrand GUYON**
**Academic Advisor: Dr. Damien SILEO**



**Abstract**

Recent studies show that toxic molecules like microplastics are increasingly present in the human body, raising serious health concerns, including possible effects on fertility, hormone disruption, chronic inflammation, and even increased cancer risk. At URGO RID, the research division of URGO, one of our core goals is to identify molecules released from everyday items like bottles and cups that may contaminate food, drinks, medications, and other materials. To support this work, we built an AI Large Language model (LLM), capable of extracting interaction properties between contents and containers from large unstructured scientific articles. We also developed an AI predictive model that estimates the likelihood of thousands of compounds being released under specific conditions. The extracted data from the LLM is stored in a structured cloud-hosted database (Snowflake) and then integrated into a Python-based software tool, allowing teams to assess contamination risks, document their findings, and make informed decisions about material safety.


August 21, 2025

URGO RID RESEARCH INNOVATION & DEVELOPMENT — Université de Lille — centralelille ÉCOLE CENTRALE DE LILLE — IMT Nord Europe École Mines-Télécom IMT-Université de Lille

# Contents

# List of Abbreviations

| Abbreviation | Full Name |
|---|---|
| URGO RID | URGO Research, Innovation, and Development |
| BPA | Bisphenol A |
| PFAS | Per- and Polyfluoroalkyl Substances |
| NIAS | Non-Intentionally Added Substances |
| AI | Artificial Intelligence |
| ML | Machine Learning |
| DL | Deep Learning |
| NLP | Natural Language Processing |
| TF-IDF | Term Frequency–Inverse Document Frequency |
| RAG | Retrieval-Augmented Generation |
| LLM | Large Language Model |
| CPU | Central Processing Unit |
| VRAM | Video Random Access Memory |
| GPU | Graphics Processing Unit |
| API | Application Programming Interface |
| SQL | Structured Query Language |
| REST API | Representational State Transfer Application Programming Interface |
| Regex | Regular Expressions |
| JSON | JavaScript Object Notation |
| HTTP | Hypertext Transfer Protocol |
| FAQ | Frequently Asked Questions |
| ADASYN | Adaptive Synthetic Sampling |
| SVM | Support Vector Machine |
| SHAP | SHapley Additive exPlanations |
| CAS | Chemical Abstracts Service |
| CID | Compound Identifier |
| SMILES | Simplified Molecular Input Line Entry System |
| IUPAC NAME | International Union of Pure and Applied Chemistry Name |
| DOI | Digital Object Identifier |
| IT | Information Technology |
| OCR | Optical Character Recognition |
| BBOX | Bounding Box |
| GGUF | GPT-Generated Unified Format |
| n_ctx | Number of Context Tokens |
| FAISS | Facebook AI Similarity Search |
| KNN | K-Nearest Neighbors |
| pkl file | Pickle File |
| exe file | Executable File |
| CV | Cross-Validation |

# Acknowledgments

I would like to sincerely thank the URGO team for warmly welcoming me during my internship. I am especially grateful to my supervisor, Mr. Bertrand Guyon, for his constant guidance and encouragement, to Miss Mingyi Sun for providing the data and collaborating on the project, to Dr. Pauline Kisseleff and Mr. Christophe Lagrange for their guidance and suggestions that led to more polished outcomes. Finally, I would like to give a special thanks to my university advisor, Dr. Damien Sileo, for his guidance, valuable advice, and continuous support throughout my work.


# URGO and its Division

URGO is a private French healthcare company, founded in 1958, and recognized for its solid expertise in wound healing, medical innovation, and consumer health. With decades of experience, the company has built a reputation for developing products that serve both patients and professionals, offering solutions that range from everyday first aid to advanced wound care technologies. URGO focuses on improving health outcomes through research-driven development, clinical precision, and user-friendly applications.

Today, URGO operates in over 50 countries worldwide, with a strong presence in markets such as France, Germany, the United Kingdom, Spain, and the United States.

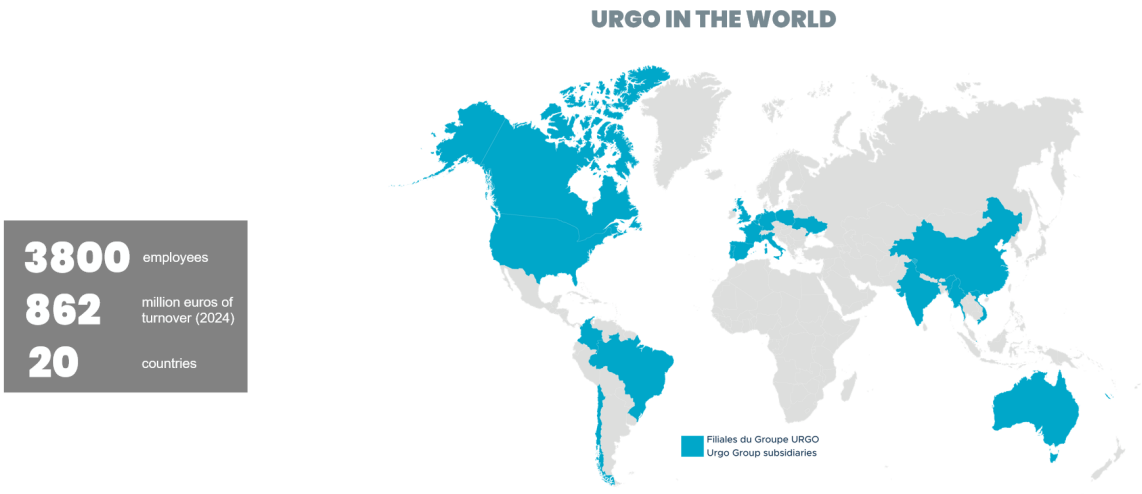



Figure 1: Global Presence of URGO

Over the years, URGO has grown into an organization with several specialized divisions. URGO Medical leads the company's efforts in advanced wound care and clinical solutions, while URGO Healthcare focuses on consumer health and over-the-counter products. The URGO RID division (Research, Innovation, and Development) drives scientific advancement and product innovation. At the core, URGO Group oversees strategic direction and international partnerships. These units are supported by other essential functions such as industrial operations, regulatory affairs, and global logistics. Together, they work in alignment to fulfill URGO's mission of advancing healthcare through innovation.

I am part of the Electro-Medical service at URGO RID, the division responsible for driving innovation through science and technology. We turn laboratory innovations into practical solutions across projects such as electro-medical devices, advanced dressings, biomaterials, safety tools, and packaging validation. We ensure the safety and reliability of materials in contact with skin or wounds by studying potential chemical migration. This work involves collaboration with regulatory teams, toxicologists, and external partners.

# 1. Introduction

This section presents the motivation behind the project and outlines its main objectives.

## 1.1. Motivation

Recent studies have shown a sharp rise in the amount of microplastics and chemical toxins that accumulate in the human body. These synthetic substances, which the body struggles to break down, are found in sources like drinking water, food containers, and medication packaging. Increasing research links these chemicals to serious health issues, including infertility, hormonal imbalances, immune disorders, and certain cancers.

Once in the body, these chemicals can travel through the bloodstream, reaching almost every major organ. Some accumulate in fat tissue, remaining for years, while others are small enough to penetrate highly protected areas like the brain or a developing fetus during pregnancy. Chemicals such as BPA and phthalates can mimic hormones or disrupt hormonal signals, affecting the body's natural balance. This disruption can affect reproduction, metabolism, mood, and brain function. Even minimal exposure, especially over time, can lead to lasting effects. These toxic molecules are often present in everyday materials, such as plastic bottles, food wrappers, cups, and containers, especially when exposed to heat, pressure, or acidic contents. Over time, they can slowly transfer into food, drinks, or medications, often without being noticed.

To address this growing concern, URGO RID launched a major initiative to investigate which harmful chemicals may migrate from containers made of various materials into the products they hold and, eventually, into the human body. The goal is to develop an AI-powered software platform called *MigraSense*, designed to assist research teams with evaluating material safety, tracking potential contamination risks, and making more appropriate and sustainable choices in packaging materials for different products.

*MigraSense* should integrate two AI models working together to support chemical safety research. The first is a Large Language Model (LLM) designed to extract specific chemical information from a wide range of unstructured scientific articles on compound migration. The second is a Machine Learning (ML) model trained on a manually labeled dataset, which will be continuously updated with additional extracted data from the LLM. This ML model predicts the likelihood of certain chemical compounds migrating under various conditions, including different materials, solvents, temperature changes, and long-term storage. Extracted information from the LLM is stored in a custom relational database, while prediction results from the ML model can be exported to Excel through the software. This setup allows researchers to identify patterns more effectively and use the predictions as a tool to validate and strengthen their own findings.

## 1.2. Objectives

The main objectives carried out during the internship are the following:

- Data collection and preprocessing conducted in collaboration with an analytical chemistry intern.
- Development of a Large Language Model (LLM) to extract specific chemical relational properties from large unstructured scientific articles.
- Implementation of a Machine Learning (ML) predictive model to estimate the likelihood of chemical compound release during container–content interactions.
- Storage and retrieval of the extracted data in Snowflake, a cloud-based database platform.
- Development of a Python software, *MigraSense*, with a human–machine interface to simulate container–content interaction scenarios.
- Integration of these AI models and cloud database functionality into the *MigraSense* software platform.
- Deployment of the complete software and documentation on URGO's internal server.

# 2. Background

In this section, we provide a brief overview of the chemical migration and toxicology problem, the fundamentals of generative AI, and the machine learning and deep learning models implemented during the project.

## 2.1. Chemical Migration and Toxicology

When products are stored in containers, unseen chemical exchanges can occur under various stimuli such as heat, light exposure, or prolonged storage as seen in figure 2.

One of these exchanges is called chemical leaching, which refers to the migration of chemical substances from the packaging into the products. These substances can be additives such as BPA, phthalates, PFAS, inks, or coatings that are not tightly bound to the main structure of the container, or other non-intentionally added substances (NIAS), such as production residues and material degradation products. This journey from container to content is not trivial: recent studies reveal that over 3,600 different chemicals have been found inside people, originating from everyday food packaging and kitchenware. From Stacey Leasca, 2024 [6].

These known hazardous compounds can potentially disrupt hormones, raise cancer risk, harm the immune system, and impair development. From Matt Fuchs, 2025 [7].

Understanding how these leaks occur and their consequences is vital for industry reform. We know that temperature, acidity, fat content and storage time strongly influence chemical migration. By learning which materials release harmful substances and under which

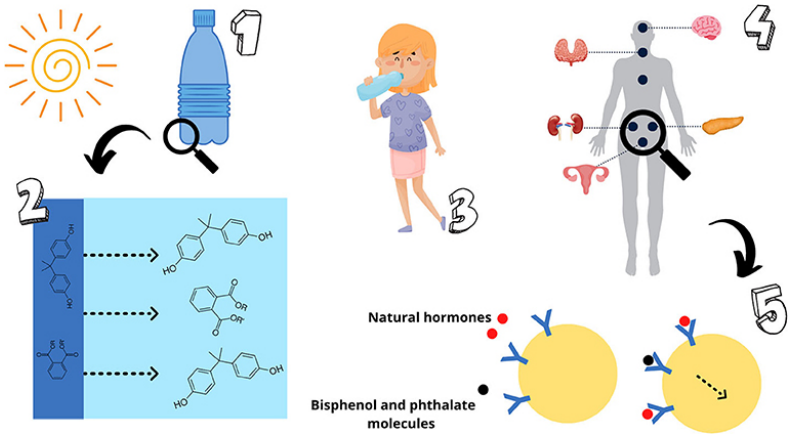



Figure 2: Endocrine disruptors like bisphenol A and phthalates can leach from plastic bottles under extreme temperatures (1,2). Once ingested (3), they may mimic hormones and interfere with the endocrine system (4), potentially triggering abnormal responses in hormone-sensitive cells (5), such as those in the reproductive system. From Helena de Oliveira Souza *et al.*, 2021 [1]

conditions, manufacturers can redesign packaging, choosing PFAS-free coatings, safer inks, or inert materials like glass or certified plastics. Regulation can then move forward, banning risky additives and mandating rigorous testing. When industries adopt these insights, they protect consumers from invisible threats and shift toward safer packaging solutions. This awareness can inspire companies to change practices, encourage policy makers to raise standards, and empower consumers to demand safer options, helping avoid irreversible harm to human health. From Anthony King, 2024 [16].

## 2.2. Large Language Models (LLMs)

LLMs are a specialized class of generative AI systems (refer to the annex A3 for more details on Generative AI) designed for processing and generating human language. Built on transformer architectures and trained on massive text corpora, LLMs leverage self-attention mechanisms to model complex dependencies across sequences. Modern LLMs, such as GPT-4 or Llama 4, contain tens to hundreds of billions of parameters and can operate over context windows ranging from 2,000 to over 10 million tokens, where the context window refers to the combined length of input and output tokens the model can process at once. This allows LLMs to handle long documents.

These models excel at a wide range of natural language processing tasks, including translation, summarization, question answering, and code generation. As language models get larger, they usually perform better and sometimes show surprising new abilities once they reach a certain size.

In healthcare, LLMs are used to generate clinical notes, extract structured data from unstructured records, assist in decision support, and power conversational agents for patient engagement or mental health. Their ability to understand domain-specific language improves when fine-tuned on biomedical corpora such as PubMed or MIMIC-III. Transfer learning allows efficient adaptation to specific applications using additional data. From Julien Simon, 2021 [11].

Despite their power, LLMs have limitations. They can hallucinate, confidently generating incorrect or fabricated facts, and suffer from issues like output truncation in long responses or degradation over large contexts. Reliability, explainability, and bias remain active areas of research. Model performance also varies depending on the prompt design, domain-specific knowledge, and the quality of the pretraining data.

One strategy to reduce hallucinations and improve factual accuracy is Retrieval-Augmented Generation (RAG). In this approach (see figure 3), the model retrieves relevant external documents, typically from a pre-indexed knowledge base, based on the user's query. The system converts the input into a vector embedding and searches for the most similar vectors in the database using a dense vector similarity search (e.g., cosine similarity). The retrieved documents are then fed into the LLM as additional context, helping it base its response on real and verifiable information.

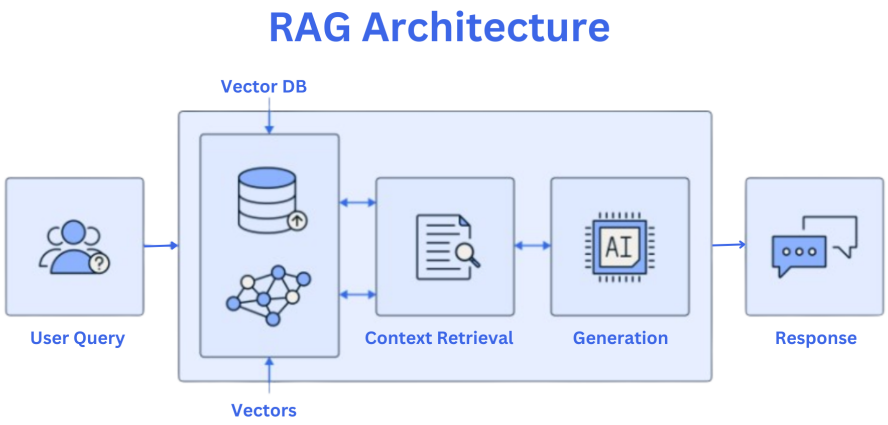



Figure 3: Diagram of a RAG Architecture. Source: freeCodeCamp.

This method helps improve reliability, especially in domains requiring high factual precision such as healthcare or legal analysis.

Deploying LLMs locally provides greater control over data and enhances security, particularly when dealing with sensitive information. However, this approach also requires significant GPU resources and a high level of technical expertise.

Scientific articles, typically found in PDF or Word document formats, often contain unstructured and implicit text information with diverse layouts, and formatting styles. As a result, common Python tools such as regular expressions (Regex) are not well-suited for reliably extracting complex information. To address this challenge, we considered using LLMs as a more powerful and adaptable solution for extracting structured interaction data between containers and their contents. Thanks to their contextual understanding of language, LLMs can effectively interpret and extract information from unstructured and inconsistently formatted documents.

## 2.3. Machine & Deep Learning Overview

Machine Learning, a branch of artificial intelligence, enables computers to improve tasks by learning patterns from data rather than following explicit instructions. It includes

three main types: Supervised Learning (using labeled data), Unsupervised Learning (discovering patterns in unlabeled data), and Reinforcement Learning (learning via trial and error with feedback). ML addresses core problems like classification (e.g., spam detection) and regression (e.g., price prediction) by optimizing model parameters during training to improve performance on unseen data. ML powers diverse applications such as fraud detection, recommendation systems, and healthcare diagnostics.

Deep Learning, a subset of ML, uses artificial neural networks inspired by the neuronal connections of the brain to analyze complex data. Unlike traditional ML, which requires manual feature extraction, deep learning models automatically learn hierarchical data representations across multiple layers, progressively identifying simple to complex features (e.g., edges to objects). This enables state-of-the-art performance in image and speech recognition, natural language processing, and autonomous systems. However, neural networks often require a large amount of data to perform well.

During the internship, various Machine & Deep Learning models were explored and evaluated to predict the likelihood of compound presence in container–content interactions. The models tested are presented in the Methodology section (refer to 4.3.2), and a technical overview of each is given in the Appendix (refer to A4).

## 2.4. AI Workflow and System Architecture of MigraSense

The diagram below illustrates the technical architecture of the MigraSense software and highlights the data flow between internal and external systems.

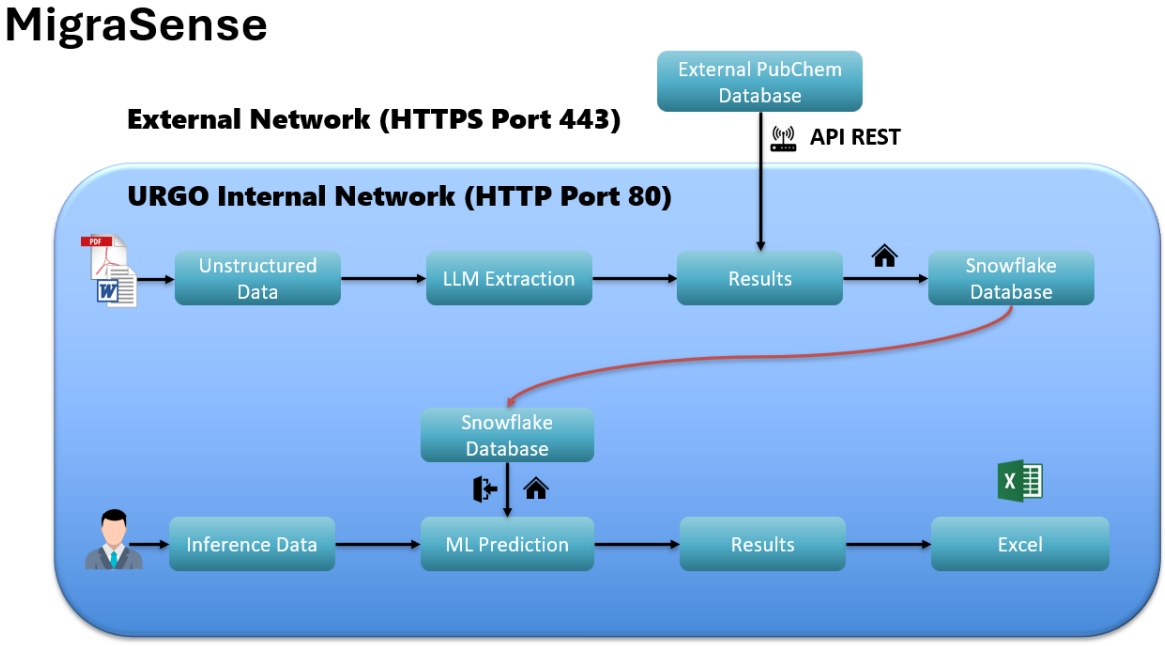



Figure 4: Main Architecture of MigraSense

As illustrated in Figure 4, the MigraSense software integrates two main AI models, each accessible from a separate section of the application. The first model processes unstructured documents in formats such as PDF or DOCX. These documents are analyzed using a LLM extractor, which identifies relevant information. The extracted content can be

reviewed and manually edited by the user within the software interface. Once validated, the user may trigger an API request to the PubChem platform, an internationally recognized chemical database, to retrieve complementary data that may not be explicitly mentioned in the original documents. The combined information is then automatically uploaded to a cloud-based Snowflake database for storage and further use.

The second model is trained on the internal URGO database. It receives inference data provided by the user and predicts the likelihood of presence for thousands of compounds listed in the database. Results are displayed in the interface and can be filtered using a configurable threshold defined by the user. If needed, the output can be exported to an Excel file for offline analysis.

The entire architecture operates on an internal system and is designed to interact securely with external resources through controlled API calls. Additional pages in the software provide access to database visualizations, user instructions, and a FAQ section.

# 3. Related Work

In this section, we review two state-of-the-art studies published prior to our research. Although we did not find any work that directly addresses the exact problem of extracting interactions between compounds and materials or predicting the likelihood of compound presence in such interactions, we still managed to identify related studies that focus on similar and related problems. These studies provided valuable insights and strategies that definitely helped guide our approach.

## 3.1. LLM-Based Extraction of Unstructured Chemical Data

A large part of chemical knowledge is written in freeform text, such as research papers or patents. This format makes it hard for computers to access and use the data. But structured information is important for faster progress in chemistry and materials science. In the past, extracting it was done by hand or with limited tools, which made the process slow and hard to scale. LLMs offer a better way. They can help turn unstructured text into structured formats that are easier to search and use.

A 2024 review by Mara Schilling-Wilhelmi *et al.,* 2025 [19], gives a full overview of how LLMs can help in chemistry. It explains how to build data extraction workflows, choose models, and check results. They emphasize that while LLMs can rapidly speed up extraction tasks that used to take weeks or months, the results still need validation using scientific knowledge and rules. Their work aligns with our use of LLMs to extract data on container–content interactions from scientific literature.

Another study by Arijit Roy *et al.,* 2024 [22], tested how LLMs perform on real chemical patents. Their goal was to improve reaction datasets used in synthesis planning. They filtered patents, used LLMs to pull out reactants, solvents, and conditions, then cleaned and validated the data. The Gemini 1.0 Pro model gave the best results. Their method found over 10,000 valid reactions and added 26% more new data than a previous method. They also corrected many earlier errors. This study shows how LLMs can improve both

the amount and quality of extracted data. It also supports our approach of using LLMs to collect detailed chemical information from complex texts.

### 3.2. ML-Based Prediction of Chemical Migration and Toxicity

A study by Shan-Shan Wang *et al.,* 2023 [24], looked at chemical migration from food packaging into food. Since lab tests are slow and costly, they trained machine learning models on 1,847 records of measured migration values. The models used molecular fingerprints, material type, and temperature as input. Their best model, based on AutoGluon, achieved high performance ($R^2 = 0.947$). While their work focuses on predicting how much chemical migrates (a regression task), our model focuses on predicting whether a compound is likely to appear at all (a classification task). Still, their methods for preprocessing features and testing model reliability were useful for our project.

Another study by Md Mobarak Hossain *et al.,* 2024 [10], focused on predicting the toxicity of chemicals in plastic packaging. They built classification models using a dataset of 6,742 compounds. To handle class imbalance, they applied sampling methods like ADASYN. They tested models such as Random Forest and SVM, and used SHAP values to explain what features influenced predictions. Their models reached high accuracy and helped screen toxic compounds more efficiently. Even though their focus is toxicity rather than migration, they dealt with similar challenges to ours, such as imbalanced data, feature selection, and data handling. Their work offered useful insights for building strong ML classifiers in our project.

## 4. Methodology

This section outlines the main methodologies explored, including the database design, LLM implementation, ML/DL techniques, and the development of *MigraSense*. The Gantt chart in Figure 5 shows the missions, tasks, and their progress from 24 March to 23 September.

PROJECT START: 24/03/2025
PROJECT END: 23/09/2025

| Tasks | March | April | May | June | July | August | September |
|---|---|---|---|---|---|---|---|
| **Phase 1: Early-Stage Project Planning** | 24th | | | | | 25th | 23rd |
| Understand project objectives and stakeholder needs. | | | | | | | |
| Analyze project architecture, pipelines, and feasibility. | | | | | | | |
| Drafting of the specifications | | | | | | | |
| **Phase 2: Software Development & Integration** | | | | | | | |
| Creation of the main pages | | | | | | | |
| Developing a human-machine interactive interface | | | | | | | |
| Integrating DB & API & AI models in the software | | | | | | | |
| **Phase 3: Database Implementation (Snowflake)** | | | | | | | |
| Snowflake training | | | | | | | |
| Table creation and database connection setup | | | | | | | |
| **Phase 4: Predictive Model Development** | | | | | | | |
| Data handling and preprocessing | | | | | | | |
| Fine-tuning ML and DL models | | | | | | | |
| Evaluating models on test set | | | | | | | |
| **Phase 5: Extraction Model Development** | | | | | | | |
| Data handling and preprocessing | | | | | | | |
| Testing different LLM architectures | | | | | | | |
| API call creation | | | | | | | |
| Models evaluation on test set | | | | | | | |
| **Phase 6: Documentation creation** | | | | | | | |
| Internship report and preparation for oral defense | | | | | | | |
| Drafting a utilization guide of the MigraSense software | | | | | | | |
| Writing a technical documentation on the code | | | | | | | |
| Protocol for software maintenance and updates | | | | | | | |

Figure 5: Gantt Chart Showing My Progress During the Internship

## 4.1. Data Modeling and Implementation in Snowflake

The selection of the database features, guided by the chemist intern's expertise, was based on an analysis of the main factors influencing migration. Three key aspects were considered important:

- the structure of the chemical compounds
- the type of the packaging materials
- the experimental conditions

Migration results were labeled in a simple way, using a binary approach (0: Not Extractable, 1: Extractable) to indicate whether the compound was detected in a given situation. Additional bibliographic information was included to ensure data traceability.

The database was organized into several logical categories. Below is an overview of the features in each category.

| **Category** | **Features** |
|---|---|
| Chemical Compounds | Cas, Name, Cid, Iupac_name, Synonym, Molar_mass, Molecular_formula, SMILES, Function, Cramer_class |
| Packaging Materials | Material, Material_detail |
| Experimental Conditions | Extraction_method, Extraction_method_detail, Solvent, Solvent_detail, Solvent_polarity, Temperature, Duration, Analysis_method |
| Publication Metadata | Title, Author, Publication_year, Doi |
| Label | Extractable |

Table 1: Database categories and their respective features

The database contains a total of 8,548 records, each with 25 features. Among them, 5,571 are labeled as Non-Extractable (negative) and 2,977 as Extractable (positive). These entries were collected from 60 distinct scientific articles.

One of the key challenges we encountered was the diversity of sources. The data came from articles published by different laboratories and countries, and while some papers described identical container-content setups, their results were sometimes contradictory and inconsistent. These differences were likely due to experimental conditions not clearly stated in the articles. As these differences could result in inaccurate interpretations by the models, a dedicated strategy was implemented to address this issue (in section 4.3.3).

The database is hosted on Snowflake, a cloud-based platform for data storage and analytics. This choice, approved in coordination with the IT department, was based on Snowflake's intuitive interface, native compatibility with Python through the snowflake-connector-python library, and strong scalability to support future needs.

## 4.2. Large Language Models Implementation

This subsection outlines the main methodology used for implementing LLMs.

### 4.2.1 Data Collection and Preprocessing

For our experiments with LLMs, we first tried using tools like pdfplumber, PyPDF2, and PyMuPDF (fitz) to extract plain text from scientific articles on compound migration, usually in PDF or Word format. These tools worked well for plain text, but struggled mainly with table data. In many cases, they failed to extract tables, or when they did, they often mixed up the content, making it unclear which values belonged to which row or column. This led to confusion and made the data hard to use. Since much of the important information is found in tables, we started looking for better solutions and found "Marker", an open-source document conversion tool that uses deep learning models to retrieve and convert content from complex documents such as PDFs into structured Markdown format.

Marker handled table extraction much more effectively, improving both readability and structure. It uses a pipeline that includes Optical Character Recognition (OCR) error detection, bounding box (bbox) detection, and scanned table recognition. By identifying and correcting OCR artifacts, Marker ensures cleaner text extraction. It then accurately detects bounding boxes around visual elements, allowing it to distinguish table cells and maintain their spatial alignment. For scanned documents, Marker applies learned models to segment and reconstruct complex tables using text-based borders, reproducing rows and columns with high fidelity. This clear formatting is essential for the LLM to correctly understand and analyze the data.

After conversion, we remove any sections that appear after the conclusion, such as references or appendices, since they do not contain useful information for our task and would only slow down processing, as shown in Figure 6 below.

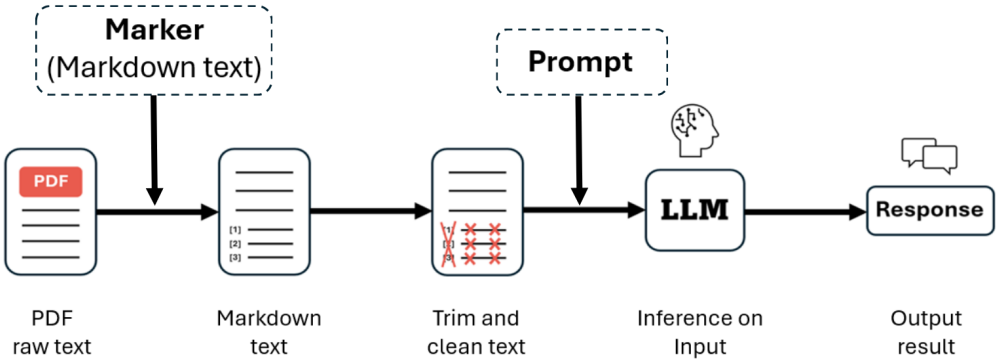



Figure 6: Data Preprocessing Pipeline for LLMs

We also clean the text by removing unnecessary elements like image tags, citation links, and leftover formatting. By doing all these, we are sure that the final input is as clean as possible before it is passed to the LLM model.

### 4.2.2 API Integration for LLM Output Completion

Once the preprocessed text is sent to the LLM along with the prompt, the model returns a JSON output (as mentioned in the prompt) that includes various details about the interactions, such as CAS, compound name, material name, extraction method, solvent, temperature, duration, analysis method, extractability (target), and some metadata of

the publication. However, in order to fill all the columns in our database, some information may be missing or not clearly mentioned in the articles. To handle this, we decided to use a REST API call to PubChem, a popular website for chemical data.

First, we need to make sure the LLM can extract the CAS number, which is a unique identifier for each compound. Once we have the CAS, we use a Python library to convert it into a CID number, which is another type of identifier. Using this CID, we can send a request to the PubChem API and retrieve more details for our database, such as the IUPAC name, molecular weight, molecular formula, SMILES, synonyms, and molecular function. These additional fields are appended to the LLM output before being stored in a dedicated table in the Snowflake database.

If the CAS number is not found in the article, the user can manually add it to the LLM output. After that, they can click the submit button to send the API request and complete the data insertion into the database.

### 4.2.3 Models Tested

During the internship, we tested several LLMs by running them locally using the GGUF format. These models were downloaded and set up on our machine, which had an RTX 3060 GPU with 8 GB of VRAM. This setup helped us to avoid using LLM APIs for data security reasons. However, inference times were sometimes slow, especially with the larger models, due to hardware limitations. The models tested included CodeLlama, DeepSeek-R1-Distill-Qwen, Gemma, Llama 2, Llama 3, Llama 3.1, Llama 4 Scout, Mistral, Nous-Hermes-2-Mistral, Phi-3 Mini, and Qwen 2.5. Testing different models helped us compare their performance and select the most suitable ones for our needs. To select these candidate models, we mainly referred to LLM ranking websites, such as LMArena.ai, and for downloading, we mainly used Ollama and Hugging Face.

### 4.2.4 Models Configuration and Parameters

As mentioned earlier, we used a local version of these LLMs in the GGUF format. The models were loaded and run completely on our machine, allowing for full control without internet dependency.

To make sure the models gave consistent and reliable answers, we carefully adjusted several key parameters. First, we set the temperature to 0.1. This parameter controls how random or creative the model responses are. A low value like 0.1 makes the model more focused and predictable, which is important for extracting structured data.

We also experimented with different context window sizes (`n_ctx`), specifically 4k, 8k, 42k, and 46k tokens, depending on the model's capabilities. The context window determines how much text the model can process at once, including both input and output. Since scientific articles often contain large volumes of information, using a higher context length was necessary to allow the model to analyze the full content without truncating important details.

In addition, n_threads was set to 8 so the model could run faster by using multiple CPU cores. We also used n_gpu_layers = 25, which allowed the model to offload the first 25

layers to the GPU for faster computation, improving overall speed and performance. On some smaller models, we used the value -1 for n_gpu_layers, which tells the system to offload all possible layers to the GPU automatically.

We used the chatml format to structure our prompts, and did not change other advanced parameters like top_p or repeat_penalty, keeping them at their default values. All model instructions and customization were done through a well-written prompt, without any further model tuning or training.

### 4.2.5 Prompt Engineering

Prompt engineering plays a crucial role in getting accurate and useful responses from LLMs. A well-written prompt guides the model reasoning and directly affects the quality of its output. There are several types of prompts, including zero-shot (where the model is given only the task, without examples), one-shot (where one example is shown), few-shot (with a few examples), chain-of-thought (which encourages step-by-step reasoning), and meta-prompting (which guides the creation of other prompts). For this project, we used a zero-shot approach, giving the model detailed, structured instructions to extract chemical data from scientific articles in a consistent JSON format. The level of clarity and specificity in the prompt was crucial for reliable extraction. For the exact prompt used, refer to Appendix A1 for further details.

### 4.2.6 Text Chunking

Some of the initial language models tested, such as Mistral, had limited context windows, preventing full article processing in a single pass. To address this, we applied a chunking approach, splitting texts into smaller parts of fixed token length and sending them sequentially to the model.

To improve context retention, overlapping tokens were added between chunks. However, this method remained insufficient. The model often lost track of earlier information, especially when key details appeared early and were referenced later, causing hallucinations.

Chunking also led to formatting issues in the model's output, particularly in JSON responses, primarily due to context loss. These limitations motivated the exploration of more robust alternatives, such as Retrieval-Augmented Generation (RAG).

### 4.2.7 Retrieval-Augmented Generation (RAG)

To address the issues we faced with chunking, we explored and implemented a Local Retrieval-Augmented Generation (RAG) approach.

Instead of sending all the chunks, a local RAG compares the user input prompt with the document chunks via vector embeddings. First, each chunk is converted into a vector using the pre-trained e5-small-v2 embedding model from the SentenceTransformers library. These vectors are then stored in a local FAISS (Facebook AI Similarity Search) vector index for efficient similarity search. When a prompt is given, it is also converted into a vector and compared with the ones in the FAISS index to find the most relevant chunks.

Only these top matching chunks are then passed to the LLM along with the prompt. This reduces the amount of data the model needs to process and helps keep the response focused and accurate. RAG also helps reduce hallucinations since it selects context that is closely related to the prompt, rather than relying on the model to remember everything.

We used a local FAISS setup to retrieve information exactly as it appears in our articles, since external databases would not provide the precise content we needed and could introduce unnecessary complexity or risks. Despite this improvement, many chunks were still considered related to our prompt without necessarily containing the exact information we needed. While RAG improved performance, it was not enough to fully resolve the main issue, and we continued to encounter incorrect JSON formats.

### 4.2.8 Huge Context Window Language Models: 128k and Beyond

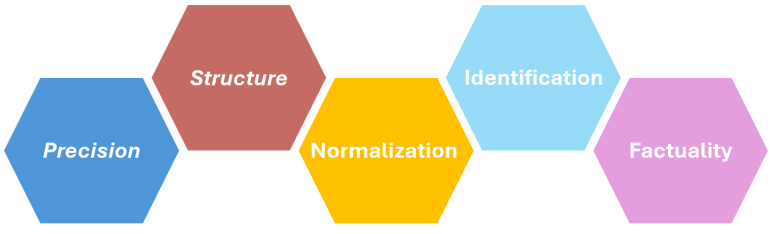
As the internship progressed, we discovered and focused on testing some of the most recent and advanced state-of-the-art LLMs with extremely large context windows. These models are very recent developments in the field, with some of them being released during the course of the internship itself. They are designed to handle much longer inputs without the need to split the data into smaller chunks. Some of the key models we explored, all evaluated under a consistent setup (n_ctx = 46,000, output max_tokens = 18,000, and temperature = 0.1), include:

- **Llama 3.1**: Released on July 23, 2024, offering a context window of **128k tokens**.
- **Qwen 2.5**: Released on January 6, 2025, offering a context size of **1M tokens**.
- **Llama 4 scout**: Released on April 5, 2025, with an impressive **10M tokens** context size. This model was released during my internship journey.

These models represent a major advancement in generative AI, as they significantly reduce or even eliminate the need for chunking. This helps preserve context across the input and leads to more accurate and coherent outputs. Since these models are built to handle long texts without the need of chunks, they offered a clear advantage for our task. Their ability to maintain full context made them an obvious choice, so we continued using them.

### 4.2.9 Evaluation Method

To assess the performance of the models, we selected two scientific articles on compound migration and manually recorded the correct extraction results for each. We then ran various language models on these same articles and compared their outputs to our reference data to determine accuracy.




The evaluation focused on five key aspects:

- **Precision**: The ability to accurately extract container–content interactions, including containers, contents, solvents, and experimental conditions.
- **Structure**: Ensuring the JSON output was correctly formatted without truncations or structural errors.
- **Normalization**: Properly handling unit conversions, such as converting time to seconds and temperature to degrees Celsius.
- **Identification**: Correctly retrieving metadata for article identification.
- **Factuality**: Returning "not mentioned" instead of hallucinating when specific data was absent from the articles.

## 4.3. Predictive ML & DL Models Design

This subsection outlines the main methodology used to design and evaluate machine learning and deep learning models for predicting the presence of compounds in container–content interactions.

### 4.3.1 Determining the Problem and Identifying the ML Task

At the very beginning of the internship, we focused on understanding how to approach the problem. We started by analyzing the problem and realized that it is a supervised learning problem. The next step was to figure out the number of classes. After discussing with chemist experts, we learned that this was not straightforward. There are thousands of possible compounds, and the list can keep growing over time. This made it clear that we could not treat the problem as a standard multi-class classification task.

After careful consideration, we decided to simplify the problem by converting it into a binary classification task with two classes: "Extractable" and "Not Extractable". The goal was to preprocess and make predictions for a dynamic list of thousands of compounds (currently 4,024) stored in a dedicated table on Snowflake. For example, in the case of the neural network model, we used a sigmoid output layer instead of softmax to output a probability score for each compound independently.

During inference, the model takes as input specific conditions such as material type, temperature, duration, and solvent. It then estimates the probability of extractability for each compound of the predefined list. A binary prediction is assigned based on a fixed threshold: compounds with a probability below 50% are classified as "Not Extractable", while those with a probability of 50% or higher are classified as "Extractable". To enhance usability, the software displays the predicted probabilities as percentages. This allows users to define their own threshold and view only compounds exceeding that value, making the output more adaptable to specific needs.

### 4.3.2 Choice of Models

During the internship, we first considered the characteristics of our dataset to guide the selection of appropriate models. Since the amount of available data was moderate and the features included a large number of categorical variables, we particularly focused on

machine learning models, especially those based on decision trees, which are well-suited for such data. However, to ensure a fair and wide evaluation of predictive performance, we also tested other model types, including neural networks.

Based on this analysis, we explored six core machine learning models: LightGBM, Logistic Regression, Random Forests, Support Vector Machines (SVM), XGBoost, and k-Nearest Neighbors (KNN). In addition, we implemented a Neural Network consisting of seven hidden layers.

Additional technical details about these six machine learning models, along with the Neural Network, can be found in the Appendix section A4.

### 4.3.3 Data Collection and Preprocessing

We took a relevant subset of features from the original dataset, used for training the machine & deep learning models, and it first contained 8 features, which are the following: Material, SMILES, Extraction_Method, Solvent, Solvent_Polarity, Temperature, Duration, and the target label Extractable. The SMILES feature is a text-based format that describes the structure of chemical compounds using ASCII characters, it acts like an ID for compounds.

As mentioned in section 4.1, we found some duplicate records with conflicting labels, originating from different articles that described similar container-content setups. These inconsistencies were likely due to unstated variations in experimental conditions across different laboratories. To address this, we developed a merging algorithm in Snowflake using SQL. The analytical chemistry intern proposed calculating, for each group of duplicates, a success rate by dividing the number of positive cases (Extractable = 1) by the total number of records in that group. If this rate was above 0.6, the group was labeled positive; otherwise, negative.

After cleaning, 3,737 unique records remained from the 8-feature dataset. We then set aside 10% of the data as an unseen test set. The remaining 90% was used for 5-fold stratified cross-validation, where each fold trained on 80% of the data and validated on the remaining 20%.

These 8 features were not enough to train our models, so we decided to add some useful chemical information for all records, we then discovered and used the RDKit Python library, which computes around 200 numerical molecular descriptors for each compound. These descriptors describe various structural and physicochemical properties. Once the descriptors were generated, the original SMILES feature was removed, and the descriptors were combined with the remaining features to form the input data for model training.

To prepare the data, we applied several preprocessing steps. First, descriptors with lots of missing values or no variance were removed to improve data quality and eliminate features that would not contribute meaningful information. We also reduced redundancy by removing one descriptor from each pair of features that had a Pearson correlation above 0.9. This step helped reduce multicollinearity and improve the efficiency of the models.

Categorical variables (Material, Extraction_Method, and Solvent) were transformed into numerical format using one-hot encoding. This process created separate binary columns for each category, allowing the models to interpret categorical information correctly.

Missing values in continuous experimental features such as Temperature and Duration were filled using the median of each column. Median imputation was chosen for its robustness to outliers and its ability to preserve the typical values.

Numerical features were divided into two types: binary (integer) and continuous (float). Continuous features were scaled using standard scaling. Binary features were left unscaled to preserve their original values. All features were then concatenated into a single matrix used for training.

Finally, to address class imbalance, since the dataset contained more Non-Extractable than Extractable samples, we applied class weighting during training. Higher weights were given to the minority class (Extractable) to ensure balanced learning and reduce bias in the model predictions.

These preprocessing steps ensured that the dataset was clean, consistent, and suitable for building accurate and reliable ML and DL models.

### 4.3.4 Model Hyperparameters and Fine-Tuning

To fine-tune the performance of the models, we optimized their respective hyperparameters using a stratified cross-validation procedure. For example, in tree-based models, we performed hyperparameter tuning over the n_estimators parameter, which controls the number of trees in the ensemble.

We used 5-fold stratified cross-validation to ensure a balanced distribution of classes across all folds. Figure 7 illustrates the complete data splitting and cross-validation setup.

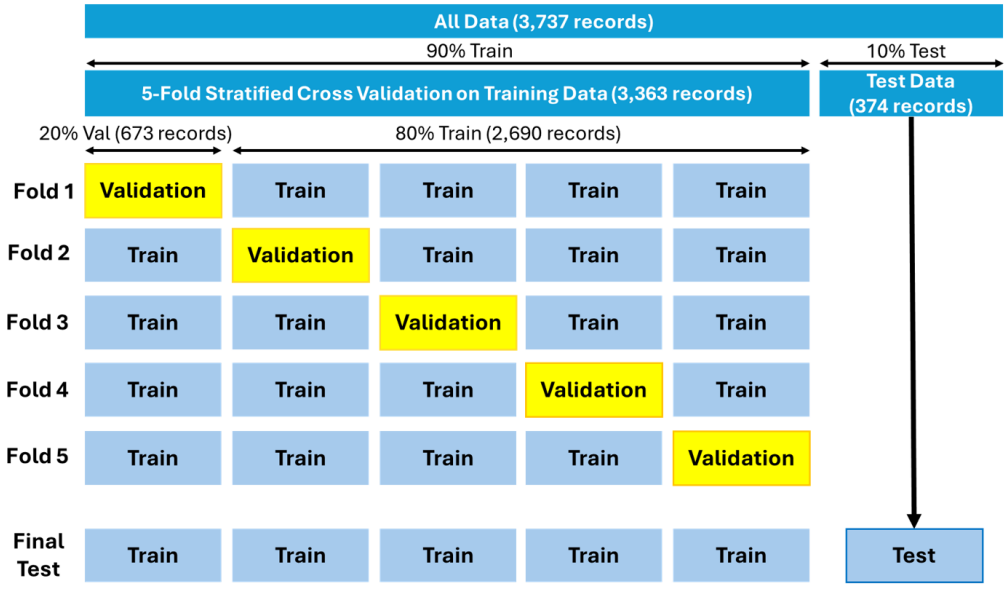



Figure 7: 5-Fold Stratified Cross-Validation and Final Test Procedure

As shown above, we first split the entire dataset of 3,737 records into a 90% training set of 3,363 records and a 10% unseen test set of 374 records. The 90% training set was then used for 5-fold stratified cross-validation, where each fold used 80% of the training data for training (2,690 records) and 20% for validation (673 records). This procedure ensured that every record was used for validation exactly once across the five folds.

The models were tested with various hyperparameter values using GridSearch or custom looping techniques to explore a wide range of possible configurations. For each configuration, we performed 5-fold stratified cross-validation, computing metrics such as accuracy, precision, recall, and F1-score for both classes (Extractable = 1 and Not Extractable = 0) on each fold. These metrics were averaged across the folds to estimate the model's generalization performance. The hyperparameter configuration achieving the highest average validation accuracy was selected, and the corresponding model was retrained on the entire 90% training set using those optimal settings.

Finally, the selected model was evaluated on the held-out 10% test set, which had remained completely unseen during training and validation.

This approach ensured that our models were both well-optimized and generalizable, minimizing the risk of overfitting and making full use of the available training data.

### 4.3.5 Evaluation Metrics

To evaluate the performance of the models, we used overall accuracy and log loss, together with precision, recall, and F1-score calculated for each class. The class-level metrics helped us assess performance in both the majority and minority classes, addressing the issue of class imbalance. All metrics were evaluated during cross-validation and later on the unseen test set to ensure robustness and generalization.

**Accuracy** gives the overall correctness of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

$TP$: True Positives

$TN$: True Negatives

$FP$: False Positives

$FN$: False Negatives

**Precision**, **recall**, and **F1-score** are key metrics used to evaluate classification models. Precision measures how many of the predicted positive cases are actually correct, while Recall indicates how many of the actual positive cases were correctly identified. The F1-score combines both precision and recall into a single metric by calculating their harmonic mean, providing a balanced measure when there is a trade-off between the two.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \qquad \text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

**Log Loss** evaluates the confidence of probabilistic predictions. It penalizes incorrect predictions with high confidence, helping assess calibration.

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

Precision, recall, and F1-score were calculated separately for each class to make sure that the evaluation was fair, especially since the classes were not balanced. We also reported macro F1 scores, which give equal importance to each class, no matter how many samples it has. This helped us better understand how well the model worked on the minority class.

### 4.3.6 Models Handling

During training, we saved each model's key components to make future use easier. For every trained model, we saved its learned weights as a .pkl file and its feature scaler as a separate .pkl file too. We also stored important configuration details like selected features, class distributions, and performance scores in a .json file. Once all models were evaluated on the test set, we selected the one with the best performance. Its saved weights, scaler, and metadata were then loaded and connected to the software's inference page, so it can be used directly for predictions in real time.

### 4.3.7 Visualization Techniques

To better understand the models performance, we used visualizations. First, we plotted a graph showing how the models performed across different values of the main hyperparameter using cross-validation accuracy. This helped us understand the models behavior across different configurations and showed us which configuration worked best. Additionally, for each trained model, we generated a confusion matrix based on its predictions on the test set, allowing us to detect common misclassifications and compare class-wise performance.

## 4.4. Software Development and Deployment

In this part, we briefly describe the software called MigraSense, which was developed during this internship. The software was written in Python using the PySide6 framework. It was designed to help with the analysis of chemical compounds in packaging by making the process easier and more user-friendly.

### 4.4.1 Software Architecture

The software is made up of six main pages: the Menu Page, Extraction Page, Prediction Page, Database Page, Guide Page, and FAQ Page. Users can also switch between light and dark themes depending on their preferences. The software was packaged and installed as an .exe file with a custom AI-generated icon. All the necessary files were included in the installation folder to make sure it runs smoothly.

### 4.4.2 DataBase Integration Into The Software

We created a connection between the database and MigraSense using the Snowflake Python connector. This allowed the software to access and interact with the online

database. Inside the app, users can filter the data, search for specific records, and refresh the page to see the latest updates. Everything was set up to ensure that the database works smoothly and gives users easy access to the information they need.

### 4.4.3 AI Integration Into The Software

We added the best-performing AI models into MigraSense, one for extraction tasks, and one for prediction. For extraction, we directly included the model file (in GGUF format) within the software. For prediction, we saved the weights of the best ML model and made sure that the input data is processed in the same way as it was during training.

# 5. Results

This section provides a detailed overview of the results achieved so far across the main tasks of this internship.

## 5.1. Performance of the Large Context Language Model

In this subsection, we provide an overview of the performance of the three selected large-context language models when applied to two external scientific articles related to compound migration.

### 5.1.1 Llama 3.1 Results

In terms of precision, Llama 3.1 was able to retrieve accurate extractions, identifying 22 compound interactions out of 34 in the first article, and 10 compound interactions out of 18 in the second.

As for structure, the output was always correct, following a well-formed JSON format.

Regarding normalization, Llama 3.1 failed to perform the required unit conversions in both articles, keeping the durations in hours instead of automatically converting them to seconds as specified in the prompt.

For the articles metadata, the results were inconsistent: it correctly retrieved the metadata of the second article, but for the first, it only extracted approximately half of the article's title and failed on the DOI.

When certain data were not mentioned in the preprocessed text of the article, Llama 3.1 correctly marked the information as "not mentioned", avoiding hallucinations.

### 5.1.2 Qwen 2.5 Results

Qwen 2.5 accurately extracted the required data, identifying 30 compound interactions out of 34 in the first article and 10 out of 18 in the second. The model showed very strong performance on the first article, whereas its performance was lower on the second. This difference in ratios indicates some instability in performance across different articles.

The output from Qwen 2.5 was also well-structured in JSON format for both articles.

Regarding normalization, Qwen 2.5 correctly performed all the unit conversions as stated in the prompt for both articles.

When retrieving article metadata, Qwen 2.5 correctly extracted the metadata for both articles but failed to retrieve the correct DOI for the second.

Regarding factual accuracy, Qwen 2.5 interpreted the DOI as missing from the second article and even generated a fabricated DOI (a case of hallucination). This behavior is problematic and should be regarded as a serious warning sign.

### 5.1.3 Llama 4 Results

Llama 4 successfully and accurately identified 25 compound interactions out of 34 in the first article, and 12 out of 18 in the second.

The model consistently produced correctly formatted JSON outputs without any errors or anomalies across all articles.

Regarding unit conversions, Llama 4 performed exceptionally well, accurately converting all units to the desired format in all cases.

For metadata extraction, it successfully retrieved the metadata from both articles, but had a difficulty retrieving the DOI in the second article.

Finally, when specific data was missing, Llama 4 correctly marked it as "not mentioned," avoiding any hallucinations.

### 5.1.4 Evaluation Overview of Large Context Language Models

Based on the evaluation results, and as we can see in Figure 8, the Llama-4-Scout-17B-16E-Instruct-UD-TQ1_0.gguf model demonstrated the best overall performance across the different evaluation criteria. Although it occasionally missed some compound interactions and did not capture all combinations, it remains one of the most recent and advanced models available for local deployment. Qwen 2.5 was a bit faster and achieved superior precision on certain articles; however, its inconsistent results and hallucinations on metadata made it a secondary option. Additionally, URGO's IT department strongly opposed the use of Qwen due to data governance and compliance concerns, expressing a clear preference for Llama's META policies.

While generative AI shows strong potential for scientific applications, most locally operated models are compressed versions of larger cloud-based models. This compression can introduce limitations, especially for complex and highly specialized tasks such as the one addressed here. Although the results were not perfect, this challenge is common and widely recognized internationally in the use of large language models. Nevertheless, the field is advancing rapidly, with ongoing improvements in model architectures, training methods, and the use of larger, higher-quality datasets, resulting in promising and more capable models in the near future.

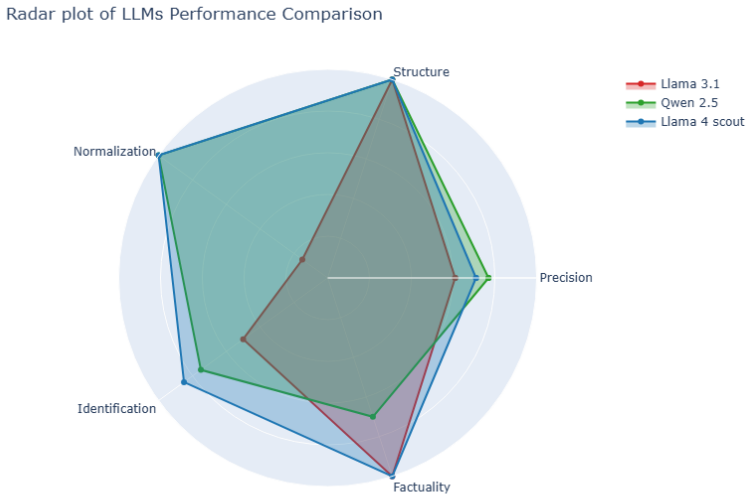



Figure 8: Radar chart comparing LLMs performance.

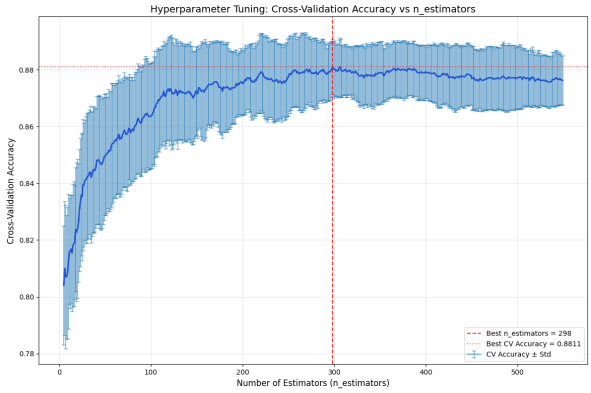
## 5.2. Evaluation of the Predictive Model

In this subsection, we present a detailed evaluation of the ML and DL models used to predict the presence of chemical compounds, based on various classification performance metrics.

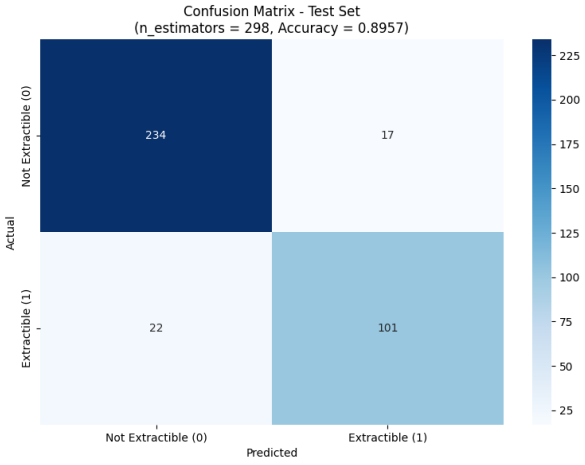
### 5.2.1 LightGBM Results

**Validation (CV):** Best n_estimators = 298, Best CV accuracy = 88.11%.

**Test Set:** Accuracy = 89.57%, Macro F1 = 88.06%, Log Loss = 0.2962.
On Not Extractable class: Precision = 91.41%, Recall = 93.23%, F1-score = 92.31%.
On Extractable class: Precision = 85.59%, Recall = 82.11%, F1-score = 83.82%.




**Validation Set:** Cross-validation accuracy across 5 folds. Best performance at 298 estimators.

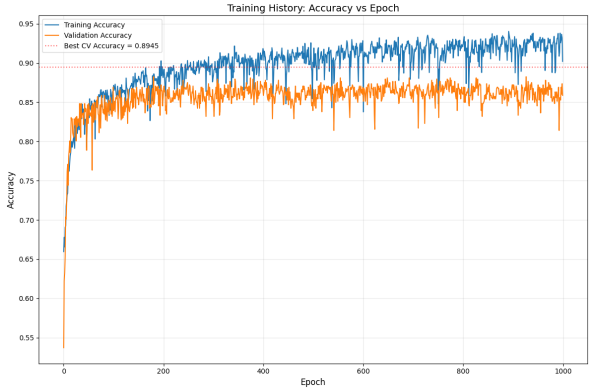



**Test Set:** Confusion matrix on test set using LightGBM.

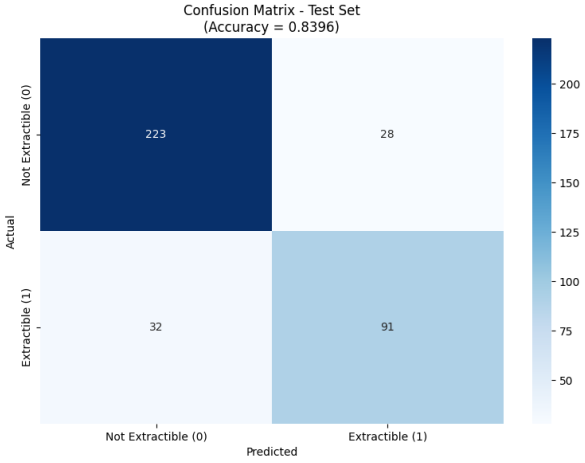
### 5.2.2 Neural Network Results

**Validation (CV):** Best CV accuracy = 89.45%.

**Test Set:** Accuracy = 83.96%, Macro F1 = 81.67%, Log Loss = 0.5323.
On Not Extractable class: Precision = 87.45%, Recall = 88.84%, F1-score = 88.14%.
On Extractable class: Precision = 76.47%, Recall = 73.98%, F1-score = 75.21%.




**Validation Set:** Training history of the best CV accuracy across 5 folds. Best performance in Fold 1 with 89.45%.

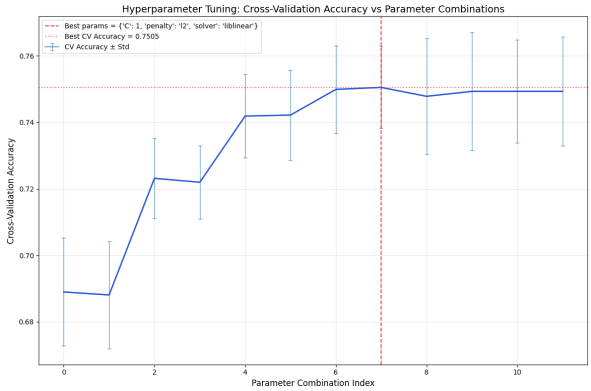



**Test Set:** Confusion matrix showing 83.96% overall accuracy.

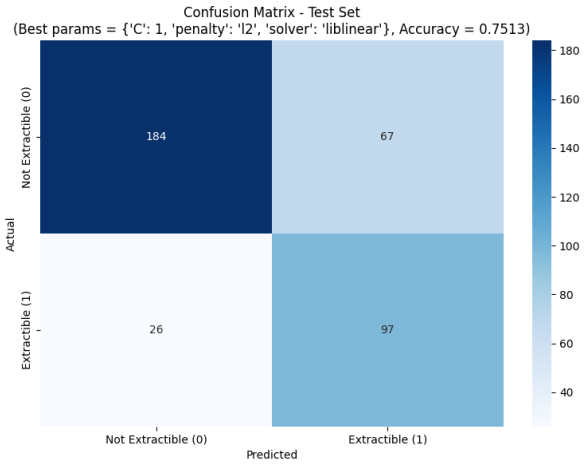
### 5.2.3 Logistic Regression Results

**Validation (CV):** Best parameters = {C = 1, penalty = l2, solver = liblinear}, Best CV accuracy = 75.05%.

**Test Set:** Accuracy = 75.13%, Macro F1 = 73.71%, Log Loss = 0.5435.
On Not Extractable class: Precision = 87.62%, Recall = 73.31%, F1-score = 79.83%.
On Extractable class: Precision = 59.15%, Recall = 78.86%, F1-score = 67.60%.




**Validation Set:** Cross-validation accuracy across 5 folds. Best performance at C = 1 using liblinear solver.




**Test Set:** Confusion matrix on test set using Logistic Regression.

### 5.2.4 Random Forest Results

**Validation (CV):** Best n_estimators = 128, Best CV accuracy = 86.41%.

**Test Set:** Accuracy = 87.70%, Macro F1 = 85.42%, Log Loss = 0.2968.
On Not Extractable class: Precision = 87.82%, Recall = 94.82%, F1-score = 91.19%.
On Extractable class: Precision = 87.38%, Recall = 73.17%, F1-score = 79.65%.

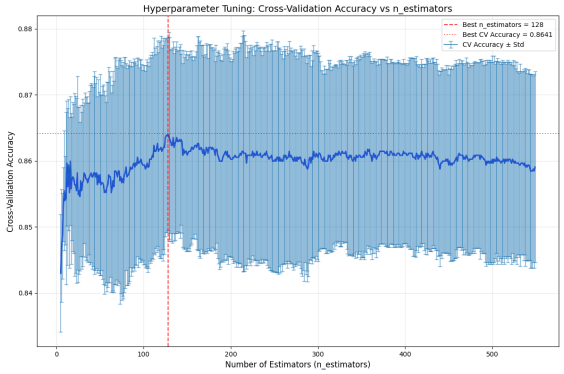



Figure 9: Cross-validation accuracy on different n_estimators values. Best accuracy at n_estimators = 128.

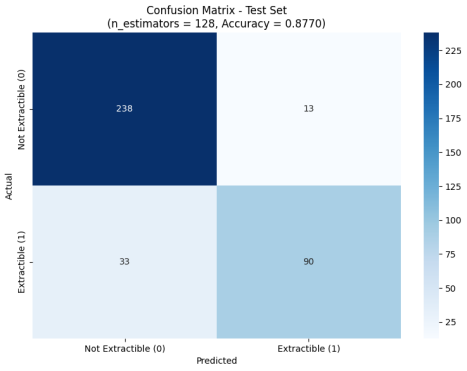



Figure 10: Confusion matrix on test set using Random Forest.

### 5.2.5 SVM Results

**Validation (CV):** Best parameters = {C = 100, kernel = rbf, gamma = 0.1, degree = 2, class_weight = balanced}, Best CV accuracy = 85.04%.

**Test Set:** Accuracy = 87.97%, Macro F1 = 86.10%, Log Loss = 0.3713.
On Not Extractable class: Precision = 89.62%, Recall = 92.83%, F1-score = 91.19%.
On Extractable class: Precision = 84.21%, Recall = 78.05%, F1-score = 81.01%.

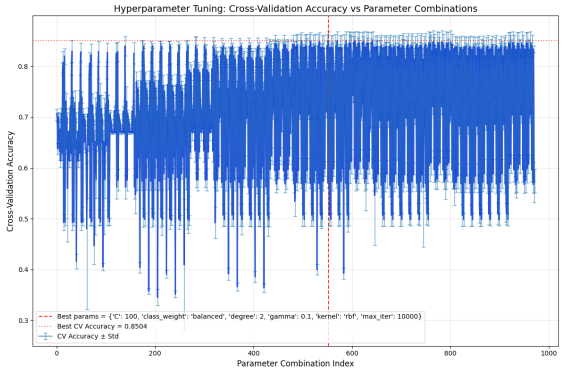



Figure 11: Cross-validation accuracy on different hyperparameter configurations.

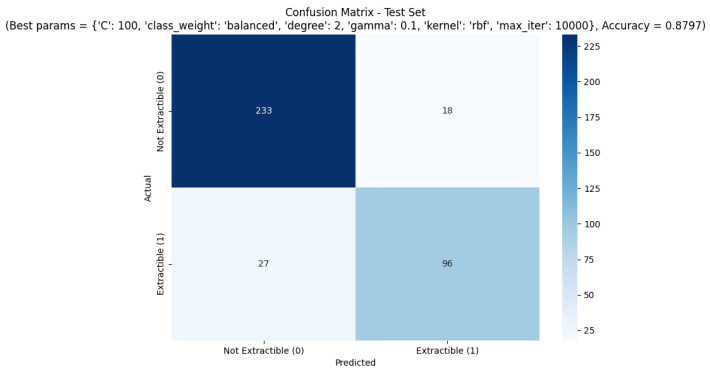
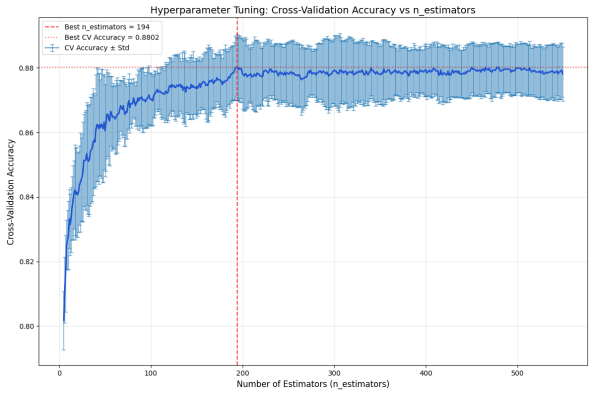



Figure 12: Confusion matrix on test set using SVM.

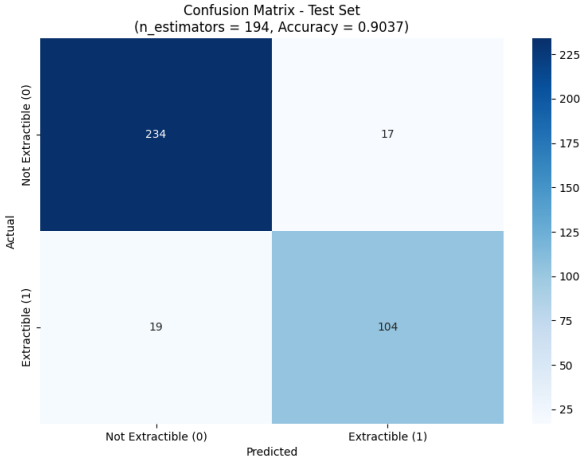
### 5.2.6 XGBoost Results

**Validation (CV):** Best n_estimators = 194, Best CV accuracy = 88.02%.

**Test Set:** Accuracy = 90.37%, Macro F1 = 89.05%, Log Loss = 0.3090.
On Not Extractable class: Precision = 92.49%, Recall = 93.23%, F1-score = 92.86%.
On Extractable class: Precision = 85.95%, Recall = 84.55%, F1-score = 85.25%.




**Validation Set:** Cross-validation accuracy across different n_estimators values. Best performance at 194 estimators.




**Test Set:** Confusion matrix on test set using XGBoost.

### 5.2.7 KNN Results

**Validation (CV):** Best n_neighbors = 4, Best CV accuracy = 84.06%.

**Test Set:** Accuracy = 86.36%, Macro F1 = 83.95%, Log Loss = 1.6661.
On Not Extractable class: Precision = 87.31%, Recall = 93.23%, F1-score = 90.17%.
On Extractable class: Precision = 83.96%, Recall = 72.36%, F1-score = 77.73%.

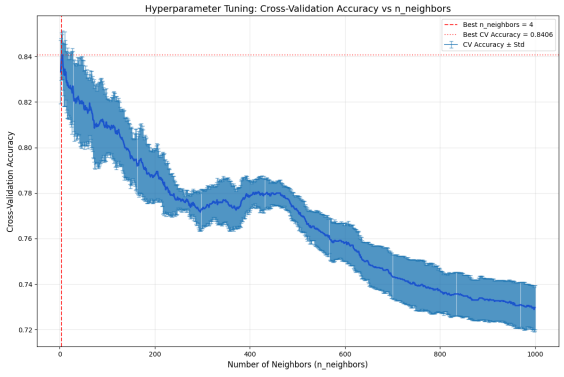



Figure 13: Cross-validation accuracy across different k values. Best at k = 4.

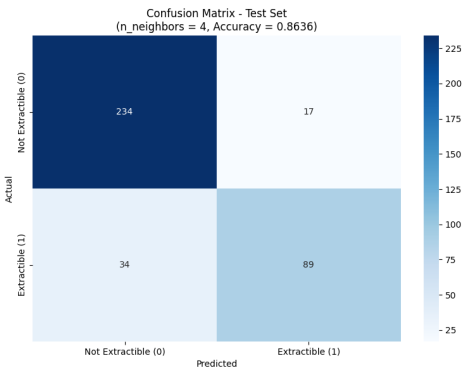



Figure 14: Confusion matrix on test set using KNN.

### 5.2.8 Overview Of The Most Relevant Hyperparameters

| Model | Key Hyperparameters |
|---|---|
| **LightGBM** | n_estimators: 298, class_weight: 'balanced', random_state: 42 |
| **Neural Network** | 7 hidden layers (512 to 8 units), Dropout: 0.2–0.7, Learning rate: 0.01, Batch size: 32, Epochs: 1000 (best at epoch 546), random_state: 42 |
| **Logistic Regression** | C: 1, penalty: l2, solver: liblinear, max_iter: 1000, class_weight: 'balanced', random_state: 42 |
| **Random Forest** | n_estimators: 128, random_state: 42, class_weight: balanced, n_jobs: -1 |
| **SVM** | C: 100, gamma: 0.1, kernel: rbf, degree: 2, class_weight: 'balanced', max_iter: 10000, probability: True, random_state: 42 |
| **XGBoost** | n_estimators: 194, random_state: 42, eval_metric: logloss, use_label_encoder: False, scale_pos_weight: ratio |
| **KNN** | n_neighbors: 4, weights: distance, random_state: 42 |

Table 2: Summary of hyperparameters used in each model.

### 5.2.9 Evaluation Metrics of All Models on the Test Set

To compare model performance and identify the most effective one, several evaluation metrics were analyzed across different models and classes. The results are summarized in the table below:

| Class | Metrics | LightGBM | Neural Networks | Logistic Regression | Random Forests | SVM | XGBoost | KNN |
|---|---|---|---|---|---|---|---|---|
| Non Extractable | Precision | 91.41% | 87.45% | 87.62% | 87.82% | 89.62% | 92.49% | 87.31% |
| | Recall | 93.23% | 88.84% | 73.31% | 94.82% | 92.83% | 93.23% | 93.23% |
| | F1-Score | 92.31% | 88.14% | 79.83% | 91.19% | 91.19% | 92.86% | 90.17% |
| Extractable | Precision | 85.59% | 76.47% | 59.15% | 87.38% | 84.21% | 85.95% | 83.96% |
| | Recall | 82.11% | 73.98% | 78.86% | 73.17% | 78.05% | 84.55% | 72.36% |
| | F1-Score | 83.82% | 75.21% | 67.60% | 79.65% | 81.01% | 85.25% | 77.73% |
| Global | Test-Loss | 0.2962 | 0.5323 | 0.5435 | 0.2968 | 0.3713 | 0.3090 | 1.6661 |
| | Test-Macro F1 | 88.06% | 81.67 | 73.71% | 85.42% | 86.10% | 89.05% | 83.95% |
| | Test-Accuracy | 89.57% | 83.96% | 75.13% | 87.70% | 87.97% | 90.37% | 86.36% |

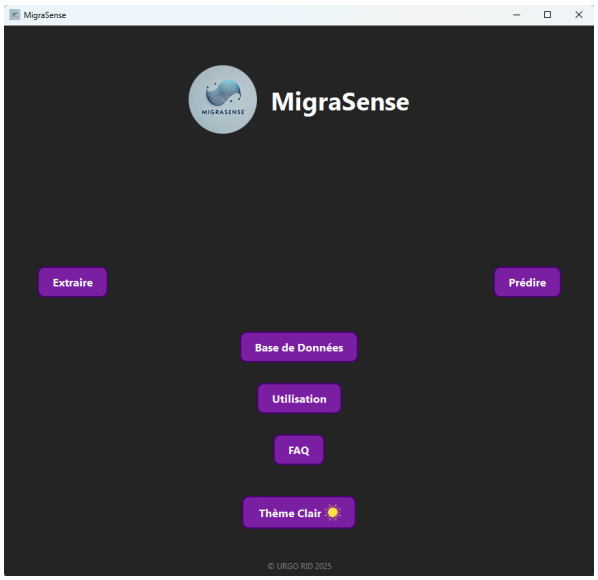
Table 3: Detailed performance metrics for each class across different models on the test set.

After analyzing the different results on an unseen Test set, the XGBoost model stood out as the best model for our task, followed by LightGBM. We noticed stable and high performance in terms of precision, recall, and F1 scores across the different classes in XGBoost. Among the seven models evaluated, XGBoost achieved the highest F1 scores for the Not Extractable (92.86%) and Extractable (85.25%) classes. It also obtained the highest Macro F1 score of 89.05%, a low log loss of 0.3090, and the highest overall accuracy

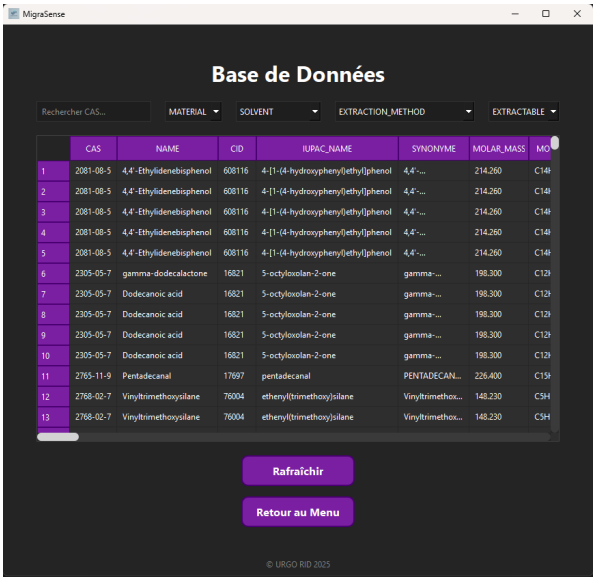
of 90.37%. Since our main priority is to ensure reliable detection of the Extractable class, recall was considered our most critical metric. Missing a potential truly toxic compound (false negative) carries a much greater risk than incorrectly predicting a compound as extractable when it is not (false positive), making the minimization of false negatives essential for safety and regulatory compliance. While balanced performance across both classes is important, the superior recall and F1 score of the XGBoost model on the Extractable class made it the most reliable choice for our application.

## 5.3. Software Implementation Outcomes

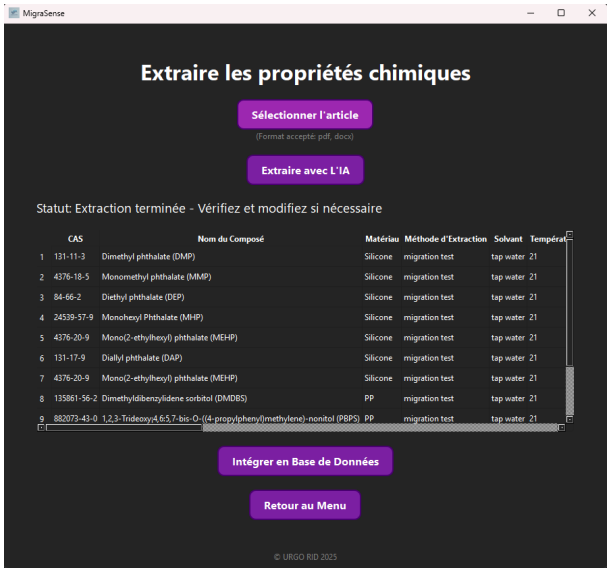
Below is a brief visual overview of the Menu, Database, Extract, and Predict pages of the application.



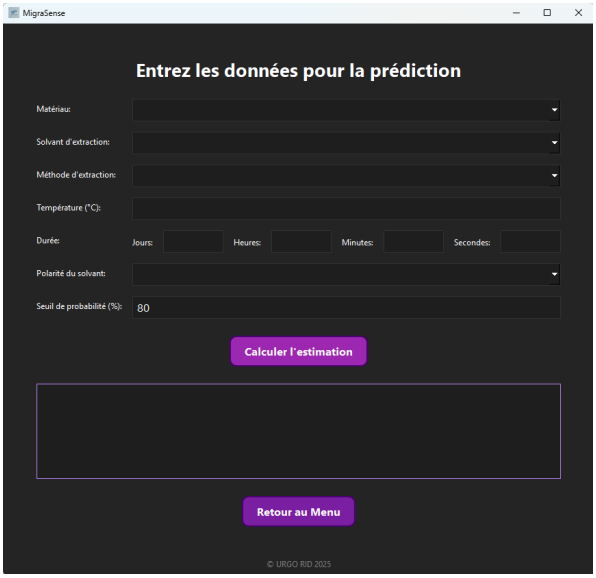

Menu page view.




Snowflake database interface.




Extraction process view.




Prediction interface for compound migration.

As shown above, the interface is designed to be simple, user-friendly, and intuitive, making it well-suited for use by chemists.

On the extraction page, we implemented a clear table-based visualization of the JSON-formatted outputs generated by the LLM. This design greatly enhances usability for chemists who may not be familiar with reading or interpreting raw JSON structures.

By the end of this internship, we successfully developed a full-stack software application. The following figure 17, illustrates the achieved work, highlighting the interactions and specific objectives of each page within the MigraSense platform.

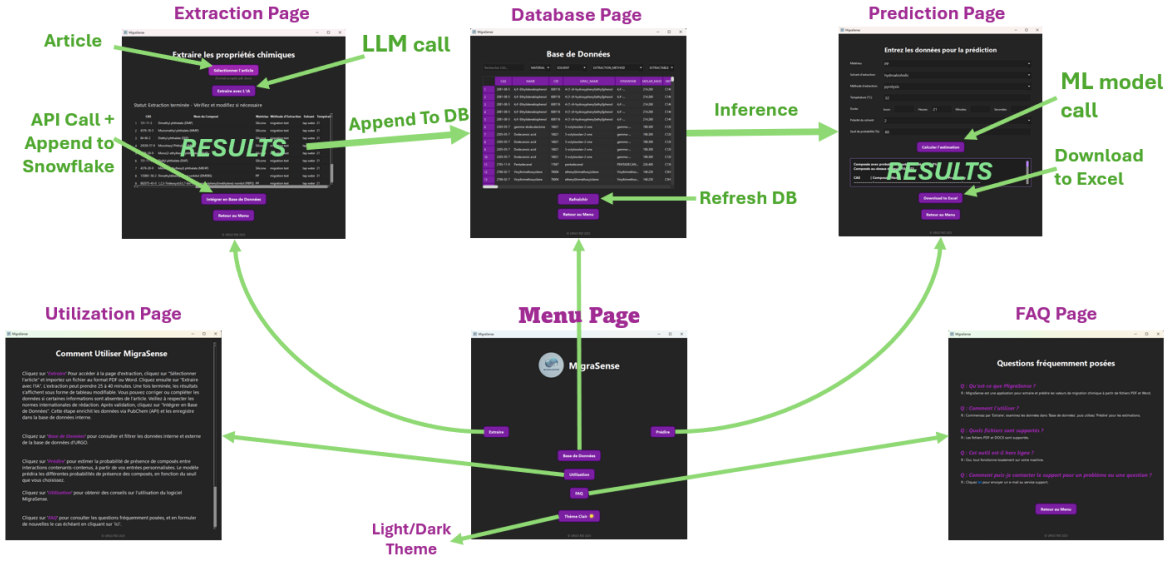



Figure 17: Schema showing the structure, interactions, and pages realized of MigraSense.

In summary, the user can access the extraction page and upload a scientific article related to compound migration. This article is first processed through a data preprocessing step before being passed to a large language model, specifically Llama-4-Scout-17B-16E-Instruct-UD-TQ1_0.gguf. The model then performs inference on the preprocessed text using a specially designed prompt that guides the extraction of structured information.

The extracted data is displayed in a table for clarity and can be directly edited within the software, and any errors can be corrected during verification, highlighting the importance of this step. Once verified, the user can trigger an API call to PubChem to retrieve additional information about the identified compounds. This enriched data is then stored in a dedicated table within the internal Snowflake database.

The Database page allows users to view the entire dataset, including all newly extracted entries. A refresh button is also available to update the table with the latest data.

Additionally, users can navigate to the Prediction page to perform inference using a trained ML model, XGBoost. By giving input data and selecting a desired threshold, users receive a list of compounds with corresponding probabilities indicating their presence. With time, as the database grows with new entries, it is recommended that URGO

periodically retrain the XGBoost model to maintain or improve prediction accuracy.

Although the interface is designed to be highly intuitive, we have also included Utilization and FAQ pages. This ensures that users without prior training can easily become familiar with the software and use MigraSense effectively.

# 6. Limitations

This project faced few limitations that affected development and performance.

The primary limitation was the limited access to high-performance computing resources. All experiments were conducted on a local machine with an RTX 3060 GPU (8 GB of VRAM), which restricted the range of models that could be used and reduced the speed and scalability of both ML and LLM training and inference.

Another limitation came from strict internal security policies enforced by URGO RID. Due to a cyberattack that occurred in 2024, the company has adopted very cautious data protection measures. As a result, we were not allowed to use external LLM APIs (like GPT-4 or Claude from Anthropic) for security reasons, or install some specific unpopular Python libraries that might connect to the internet. This restricted the use of some advanced tools and limited experimentation with certain models.

# 7. Future Works

Several improvements can be made to extend and strengthen this project.

First, using more powerful computing resources, such as investing in cloud-based GPUs, would significantly accelerate training and inference, enable fine-tuning of the LLM, and allow experimentation with larger models. These resources would also enable the exploration of Graph Neural Networks (GNNs) for predicting compound presence probabilities, which are computationally intensive and require significant GPU memory and processing power.

Access to external tools could also be expanded. Due to restrictions, we were unable to use APIs from OpenAI, Anthropic Claude, and DeepSeek. In the future, testing these tools, after appropriate ethical and security reviews, might significantly enhance performance, thanks to their immense training corpora and state-of-the-art architectures.

The MigraSense interface could benefit from regular updates and maintenance to improve usability, security, add new features, and optimize its performance.

Retraining the ML models with larger internal datasets, particularly with more samples from minority classes, could improve accuracy and generalization.

Finally, keeping up with recent advances in LLMs will enable the adoption of more effective and higher-performing versions than the current one.

# 8. Conclusion

To conclude, my six month internship at URGO, which is still ongoing until September 23rd, has so far been an incredibly rewarding journey. Under the guidance of Mr. Bertrand Guyon, I had the opportunity to apply my academic knowledge to real-world challenges involving generative AI, machine learning, deep learning, cloud databases, and software development. Each task pushed me to grow, and the learning that came with it made the experience very exciting and fulfilling.

Throughout the internship, I developed a deeper understanding of data pre-processing, especially when working with poorly described datasets for the ML models. I learned how to approach complex supervised learning tasks, even when the number of classes is very large or unknown, while applying techniques to scale relevant features from the data, manage class imbalance, and fine-tune models for optimal performance. I also gained hands-on experience with large language models, from chunking and retrieval-augmented generation (RAG) to prompting to APIs and leveraging recent advanced state-of-the-art models capable of handling long contexts. On the technical side, I learned how to integrate cloud databases securely and turned a basic software prototype into a functional application.

I am proud to share that the developed software, *MigraSense*, is now being used internally by the URGO Research and Development team. We have even started training internal chemists to use it effectively as a tool to assist and assess their work, marking an important milestone and a valuable outcome of this internship.

Until September 23, 2025, I will continue exploring and testing new models to see if any can outperform the current one. At the same time, I will document the current methods and AI models in a clear technical report, so that future developers who might continue on this project, can easily understand the work, maintain the code, and try out newer approaches.

Overall, the supportive environment at URGO allowed me to grow both technically and personally. This exciting experience has definitely inspired me to continue exploring and contributing constantly to the fields of Data Science and Artificial Intelligence.

# References


[1] Bacteria and chemicals in my bottled water?! *Frontiers for Young Minds*, 2021.

[2] Yahya Ansari. Understanding logistic regression: A beginner's guide, June 2023.

[3] Avanwyk. An overview of lightgbm, May 2018.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[5] DeepAI. Generative ai, September 2023.

[6] Food & Wine. More than 3,000 chemicals enter our bodies through food packaging — here's what you can do about it, 2025.

[7] Matt Fuchs. Why food chemicals are a problem—and how to reduce your exposure. *TIME*, March 2025.

[8] GeeksforGeeks. K-nearest neighbor (knn) algorithm, April 2017.

[9] GeeksforGeeks. Support vector machine (svm) algorithm, January 2021.

[10] Md Mobarak Hossain and Kunal Roy. The development of classification-based machine-learning models for the toxicity assessment of chemicals associated with plastic packaging. *Journal of Hazardous Materials*, 484:136702, 2025.

[11] Hugging Face. Large language models: A new moore's law?, 2025.

[12] IBM. What is nlp (natural language processing)?, August 2024.

[13] Abhishek Jain. Everything about random forest. `https://medium.com/@abhishekjainindore24/everything-about-random-forest-90c106d63989`, September 2024.

[14] Rahul Jayawardana and Thusitha Bandaranayake. Analysis of optimizing neural networks and artificial intelligent models for guidance, control, and navigation systems, 04 2021.

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[16] Anthony King. Concerns raised over health effects of chemicals leaching from food packaging. *Chemistry World*, March 2024.

[17] Massachusetts Institute of Technology. Introduction to machine learning – chapter 5: Logistic regression. `https://openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019/courseware/Week4/logistic_regression/?activate_block_id=block-v1%3AMITx%2B6.036%2B1T2019%2Btype%40sequential%2Bblock%40logistic_regression`.

[18] Marianna Rapsomaniki et al. Ai for drug discovery. `https://research.ibm.com/publications/ai-for-drug-discovery`, 2024.

[19] Mara Schilling-Wilhelmi, Martiño Ríos-García, Sherjeel Shabih, María Victoria Gil, Santiago Miret, Christoph T. Koch, José A. Márquez, and Kevin Maik Jablonka. From text to insight: large language models for chemical data extraction. *Chem. Soc. Rev.*, 54:1125–1150, 2025.

[20] Stataiml. How to implement k-nearest neighbors (knn) in python, December 2023.

[21] Tech-AI-Math. Support vector machines (svm) in depth (part 1). `https://ai.plainenglish.io/support-vector-machines-svm-in-depth-part-1-882448c8310c`, July 2023.

[22] Sarveswara Rao Vangala, Sowmya Ramaswamy Krishnan, Navneet Bung, Dhandapani Nandagopal, Gomathi Ramasamy, Satyam Kumar, Sridharan Sankaran, Rajgopal Srinivasan, and Arijit Roy. Suitability of large language models for extraction of high-quality chemical reaction dataset from patent literature. *Journal of Cheminformatics*, 16(1):131, November 2024.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[24] Shan-Shan Wang, Pinpin Lin, Chia-Chi Wang, Ying-Chi Lin, and Chun-Wei Tung. Machine learning for predicting chemical migration from food packaging materials to foods. *Food and Chemical Toxicology*, 178:113942, 2023.

[25] Yuanchao Wang, Z. Pan, J. Zheng, L. Qian, and Li Mingtao. A hybrid ensemble method for pulsar candidate classification, 08 2019.

[26] Wikipedia contributors. Neural network (machine learning). `https://en.wikipedia.org/w/index.php?title=Neural_network_(machine_learning)&oldid=1305375655`, August 2025.

[27] Wikipedia contributors. Random forest. `https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=1297666518`, June 2025.

# Appendix

## A1. Prompt Used for Extraction

Below is the exact prompt used for the LLM model:

```
You are an AI assistant specializing in extracting precise chemistry data from
scientific articles.
Your task is to extract ALL instances of studied compounds and their associated
extraction details from the given text.

OUTPUT FORMAT: Return a valid JSON array where each element is an object
representing a compound record with the following fields:

- "compound_cas": studied compound cas number
- "compound_name": studied compound name in the article
- "material_detail": studied packaging material with detail
- "extraction_method_detail": set as 'migration test' for migration experiments
even if extraction methods are also involved. For extraction experiments,
precise the method
- "solvent_detail": solvent or simulant with concentration
- "temperature": test temperature in Celsius
- "duration": test duration in seconds
- "analysis_method": analysis method
- "title": title of the given article
- "author": name of the principal author
- "publication_year": e.g., publication year of the article
- "doi": DOI link for the article starting with 'http'
- "extractible": if the compound is detected under the previous conditions,
mark as 1, otherwise 0

IMPORTANT:
1. Extract ALL combinations of compounds, materials, conditions, and sources
in the text.
2. Exclude composite materials (e.g., 70%HDPE/30%LDPE) and recycled materials.
3. Consider materials with different features distinctly.
4. For any missing information, set the field to "not mentioned".
5. Do any necessary unit conversions.
6. Return only the valid JSON array (no other text).
```

## A2. Advances in Natural Language Processing

Natural Language Processing (NLP) is a subfield of computer science and artificial intelligence concerned with enabling machines to understand and generate human language. The field originated in the 1950s, with foundational ideas from Alan Turing and early experiments in machine translation.

Typical NLP pipelines begin with preprocessing steps such as tokenization, which divides text into words or sentences, and part-of-speech (POS) tagging, which assigns grammatical categories (e.g., nouns, verbs) to each token. Stop word removal filters out commonly used words like "the" and "is" that carry little semantic value. Named entity recognition (NER) is another key step, identifying proper names of people, organizations, and locations within text.

After preprocessing, textual data is transformed into numerical representations to enable computational analysis. Traditional vectorization methods include Term Frequency–Inverse Document Frequency (TF–IDF) and Word2Vec embeddings.

These vectorized representations support a wide range of NLP tasks, including sentiment analysis, text summarization, keyword extraction, and document clustering. In the context of scientific literature, NLP helps researchers summarize papers, extract key findings, cluster related studies, and uncover hidden relationships between concepts. This streamlines the management of large document collections and supports efficient data-driven research. Based on IBM, 2024 [12].

# A3. Overview of Generative AI

Generative AI refers to models capable of producing new content, such as text, images, audio, video, or even molecular structures, by learning patterns from large datasets. Early approaches like Markov chains or probabilistic grammars struggled to capture long-range dependencies, limiting their realism. A major advancement came in 2017 with the Transformer architecture, introduced in "Attention Is All You Need" by Vaswani et al. [23]. Unlike Recurrent Neural Networks (RNNs), Transformers rely entirely on self-attention mechanisms, which allow the model to consider different parts of the input sequence in parallel. This shift enabled much faster training and better performance on tasks that involve long-range dependencies and context-aware generation, forming the foundation of powerful models like BERT and GPT.

Today, generative AI powers a wide range of tools. Text-based models such as GPT-4 are capable of generating human-like responses. In image generation, GANs (Generative Adversarial Networks) oppose two neural networks, a generator and a discriminator, against each other to produce highly realistic visuals. Diffusion models like Stable Diffusion generate images by gradually denoising random noise into coherent pictures through reverse diffusion processes. In audio, tools like OpenAI's Jukebox or ElevenLabs synthesize music and mimic human voices with remarkable fidelity. Video generation is also emerging, with models like Sora and Veo producing short, realistic clips from simple prompts.

These innovations have also reached healthcare and chemistry: generative models analyze patient records, draft medical documentation, assist in diagnosis, and even design novel drug candidates by predicting molecular properties or simulating reactions. From Marianna Rapsomaniki *et. al.*, 2024, and Stefan Feuerriegel *et. al.*, 2023 [18, 5].

# A4. Machine & Deep Learning Overview

This appendix section provides technical explanations of the various machine learning and deep learning models used during the internship.

## A4.1. Logistic Regression

Logistic regression is a fundamental algorithm in supervised learning, originally developed in the 1940s to model binary outcomes. Unlike linear regression, which predicts continuous values, logistic regression is used for classification tasks where the output is categorical, typically binary. It models the probability that an input vector belongs to a particular class.

Mathematically, logistic regression estimates the conditional probability of the target variable given the input features using the logistic (sigmoid) function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = w^T x + b,$$

with $w$ as the weight vector, $x$ the feature vector, and $b$ the bias term. The sigmoid function ensures the output is a valid probability, constrained to the (0,1) interval. A threshold, commonly 0.5, is used to assign class labels: probabilities below 0.5 map to class 0, and those at or above 0.5 map to class 1. This threshold can be adjusted for specific tasks, such as imbalanced datasets.

During training, the weights are optimized by maximizing the likelihood of the data, equivalent to minimizing the binary cross-entropy loss:

$$p_i = \sigma(w^T x_i + b), \quad L(\theta) = -\sum_{i=1}^{n} \left[ y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right],$$

where $y_i$ is the true label. Optimization is typically performed using iterative methods like gradient descent.

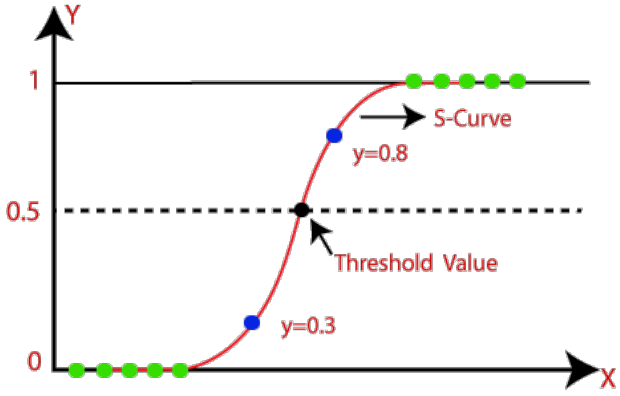



Figure 18: Diagram of the logistic regression algorithm. From Yahya Ansari, 2023, [2].

It is interpretable due to its linear decision boundary in the feature space ($w^T x + b = 0$) and has a convex loss function, ensuring a global minimum during optimization. It scales well with large datasets and can be extended to multiclass problems using techniques like one-vs-rest or softmax regression. However, it assumes linear separability and may require feature engineering for complex, non-linear relationships. From MIT, 2019 [17].

## A4.2. Random Forests

Random Forests is an ensemble learning method that constructs multiple decision trees and aggregates their outputs to enhance accuracy and robustness. Introduced by Leo Breiman in 2001, it combines bootstrap aggregation (bagging) and random feature selection to reduce overfitting and improve generalization.

Each decision tree is trained on a bootstrap sample of the dataset, and at each split, a random subset of features is considered. This introduces diversity among trees. For classification, predictions are made by majority vote; for regression, the average output is used.

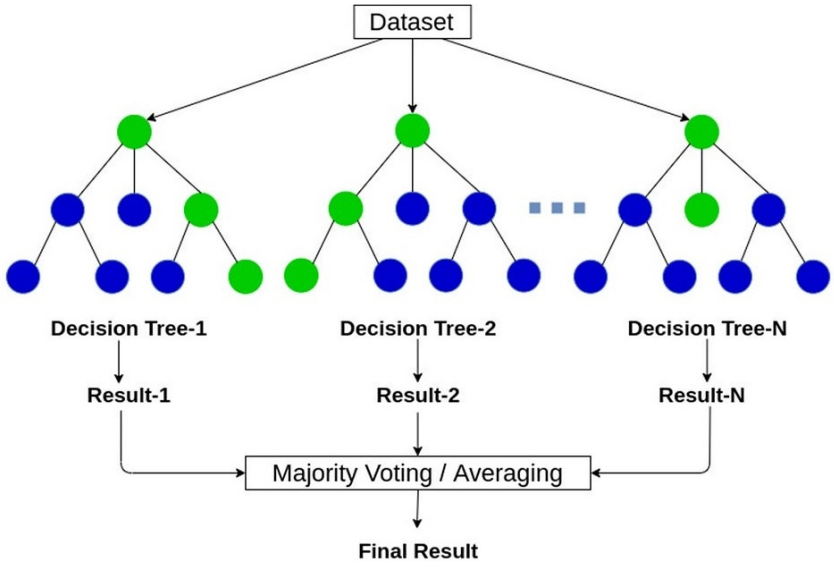



Figure 19: Random Forests Algorithm visualization. From Abhishek Jain, 2024, [13].

Decision trees partition the data by recursively selecting features and thresholds that maximize criteria like Gini impurity or information gain. While individual trees may overfit, Random Forests mitigate this through ensemble averaging.

Random Forests handle high-dimensional and non-linear data well, they are less sensitive to noise, and provide feature importance metrics based on Gini importance or mean decrease in accuracy.

Mathematically, if we denote $T_b(x)$ as the prediction of the $b$-th tree for input $x$, the final prediction for classification is:

$$\hat{y} = \text{mode}(T_1(x), T_2(x), \ldots, T_B(x))$$

And for regression:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$$

Random Forests are particularly effective in settings with noisy data or when overfitting is a concern. They also support parallel training since each tree can be constructed independently. From Wikipedia [27].

## A4.3. Support Vector Machines

Support Vector Machines (SVMs) are a set of supervised learning algorithms introduced in the 1990s, based on the statistical learning theory developed by Vladimir Vapnik. The central idea of SVMs is to find the hyperplane that maximally separates the data points of different classes. This hyperplane is defined in such a way that it maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class, called support vectors.

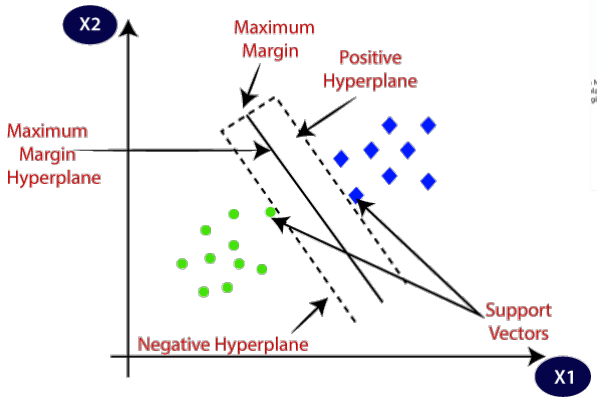



Figure 20: SVM Algorithm visualization. From Tech-AI-Math, 2023, [21].

For linearly separable data, the optimal hyperplane can be found by solving the following convex optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to } y_i(w^T x_i + b) \geq 1 \quad \forall i$$

where $w$ is the weight vector, $b$ is the bias, $x_i$ is the input vector, and $y_i \in \{-1, 1\}$ is the class label. This minimizes the inverse of the margin while ensuring correct classification.

This ensures that data points are correctly classified and that the margin is maximized.

For non-linearly separable data, the soft-margin SVM introduces slack variables to allow some misclassifications, leading to the modified objective:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

Here, $C > 0$ controls the trade-off between maximizing the margin and minimizing classification errors.

To handle complex datasets, SVMs utilize the kernel trick, which implicitly maps data into a higher-dimensional space where a linear separation is possible. Common kernels include the radial basis function (RBF), polynomial, and sigmoid kernels.

SVMs are particularly effective in high-dimensional spaces and in cases where the number of dimensions exceeds the number of samples. They are known for their robustness and theoretical guarantees on generalization. From GeeksforGeeks, 2021 [9].

## A4.4. XGBoost

XGBoost, short for eXtreme Gradient Boosting, is a high-performance implementation of gradient boosting decision trees introduced by Tianqi Chen in 2016. XGBoost builds an ensemble of decision trees in a sequential manner, where each new tree is trained to correct the residual errors of the existing ensemble.

The core idea is to minimize a regularized objective function that combines a convex loss function with a penalty term to control model complexity:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad \text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 + \alpha\|w\|_1$$

where

- $l(\hat{y}_i, y_i)$ is the loss function measuring the difference between the predicted value $\hat{y}_i$ and true label $y_i$,
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 + \alpha\|w\|_1$ is the regularization term, where $\gamma$, $\lambda$, and $\alpha$ are regularization parameters controlling the number of leaves, L2 regularization, and L1 regularization, respectively,
- $T$ is the number of leaves in the tree $f$,
- $w$ represents the leaf weights (scores) of the tree,
- $f_k$ denotes the $k$-th tree in the ensemble.

Each tree $f_k$ in the ensemble predicts a score for the instance $x_i$, and the final prediction is the sum of outputs from all trees. The algorithm uses second-order gradient information (i.e., the Hessian) to improve the accuracy of the optimization. It also includes system-level optimizations such as cache-aware access, column block compression, and out-of-core computation.

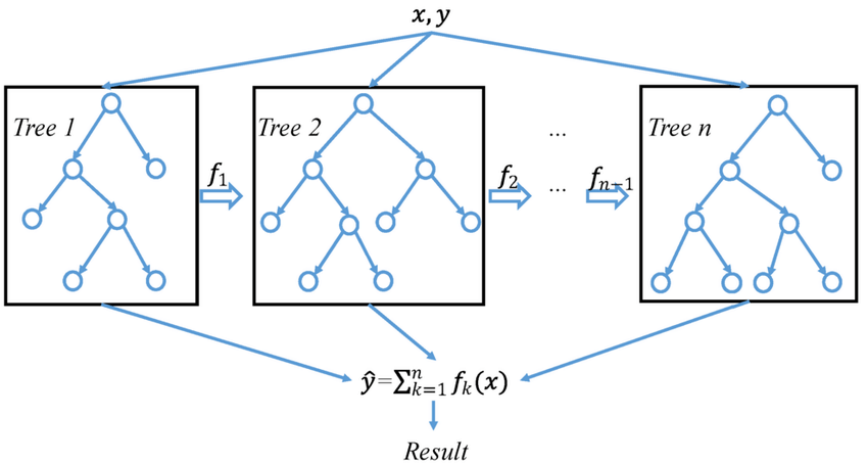



Figure 21: XGBoost Architecture visualization. From Li Mingtao, 2019, [25].

XGBoost supports both classification and regression tasks, it handles missing values internally, and it is known for its scalability and efficiency. Its success has made it a favorite model in data science competitions and production systems. From Tianqi Chen *et. al.*, 2016 [4].

## A4.5. LightGBM

LightGBM is a gradient boosting framework developed by Microsoft in 2017, designed to be more efficient than traditional boosting algorithms, especially on large datasets. It introduces two main innovations: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

Just like XGBoost, the core idea is to minimize a regularized objective function that combines a convex loss function with a penalty term to control model complexity:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad \text{where } \Omega(f) = \gamma T + \tfrac{1}{2}\lambda\|w\|^2 + \alpha\|w\|_1$$

where

- $l(\hat{y}_i, y_i)$ is the loss function measuring the difference between the predicted value $\hat{y}_i$ and true label $y_i$,
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 + \alpha\|w\|_1$ is the regularization term, where $\gamma$, $\lambda$, and $\alpha$ are regularization parameters,
- $T$ is the number of leaves in the tree $f$,
- $w$ represents the leaf weights (scores) of the tree.

GOSS keeps instances with large gradients (i.e., those that are hard to predict) and randomly samples from the rest, ensuring that the most informative examples are used to compute the information gain. EFB combines mutually exclusive features (those that rarely take non-zero values simultaneously, such as one-hot encoded features) into a single feature, reducing the number of features processed and accelerating training.

Unlike traditional gradient boosting frameworks that grow trees level-wise (splitting all nodes at the same depth), LightGBM grows trees leaf-wise. At each step, the algorithm identifies the leaf with the highest potential gain and splits it. This often leads to deeper trees with better accuracy, although it can increase the risk of overfitting if not controlled with proper regularization.

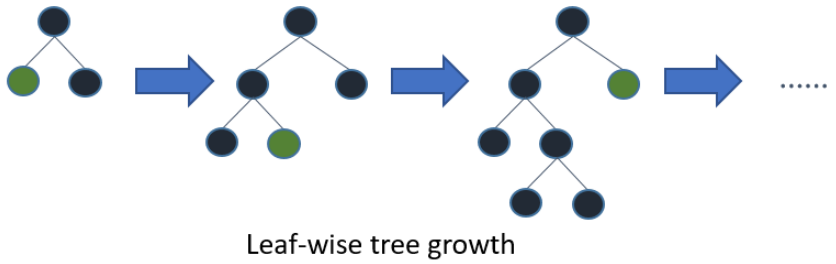



Figure 22: LightGBM Overview visualization. From Andrich van Wyk, 2018, [3].

LightGBM is highly optimized for performance, supporting GPU acceleration and distributed learning. It is widely used in production environments and consistently outperforms earlier boosting implementations, such as AdaBoost, in speed and memory usage, while remaining competitive with modern frameworks like XGBoost. From Guolin Ke *et. al.*, 2017 [15].

## A4.6. K-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm is a non-parametric, instance-based learning method that classifies a data point based on the labels of its nearest neighbors. It assumes that similar instances are likely to belong to the same class. For a given query point, the algorithm finds the $k$ closest points in the training data using a distance metric such as Euclidean distance ($d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$), Manhattan distance ($d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$), or the more general Minkowski distance ($d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$), where $p = 1$ yields the Manhattan distance and $p = 2$ yields the Euclidean distance.

Once the $k$ neighbors are identified, the algorithm performs a majority vote (for classification) or computes the average (for regression). The choice of $k$ is critical: small values make the model sensitive to noise, while large values may smooth out important patterns. It is generally recommended to use an odd value for $k$ to avoid ties during classification. Methods like cross-validation and the elbow method are commonly used to find the optimal $k$.

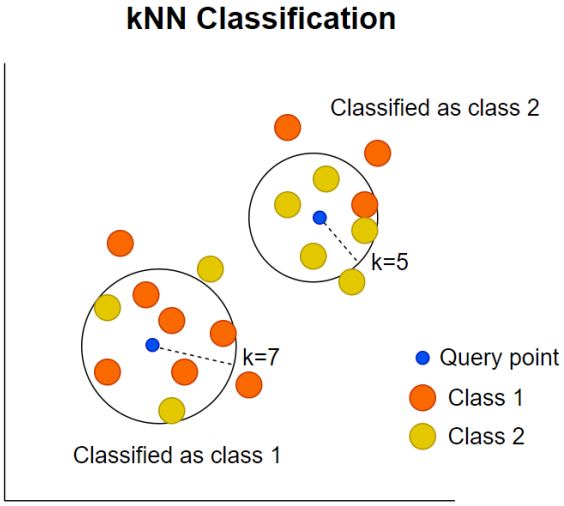



Figure 23: KNN Algorithm visualization. From Stataiml, 2023, [20].

KNN is also known as a lazy learner algorithm since it does not build a model during training but rather stores the entire dataset. Predictions are made only when a new data point needs to be classified, which can be computationally expensive on large datasets. To improve accuracy, distance weighting can be applied so that closer neighbors have more influence than distant ones.

KNN is widely used in applications such as anomaly detection, image classification, handwriting recognition, and recommendation systems. Despite its simplicity, it is effective in scenarios where the data distribution is complex and not easily captured by parametric models.

The simplicity of KNN comes with trade-offs: it is highly sensitive to irrelevant or unscaled features, and its inference time increases with the dataset size. Feature normalization (e.g., min-max scaling or standardization) is often essential for good performance. From GeeksforGeeks, 2017 [8].

## A4.7. Neural Networks

Neural Networks are a family of machine learning models inspired by the structure and function of the human brain. They consist of layers of interconnected nodes (also called neurons), where each neuron applies a linear transformation to the input followed by a non-linear activation function. The simplest form is the feedforward neural network, where information flows in one direction, from the input layer, through one or more hidden layers, to the output layer.

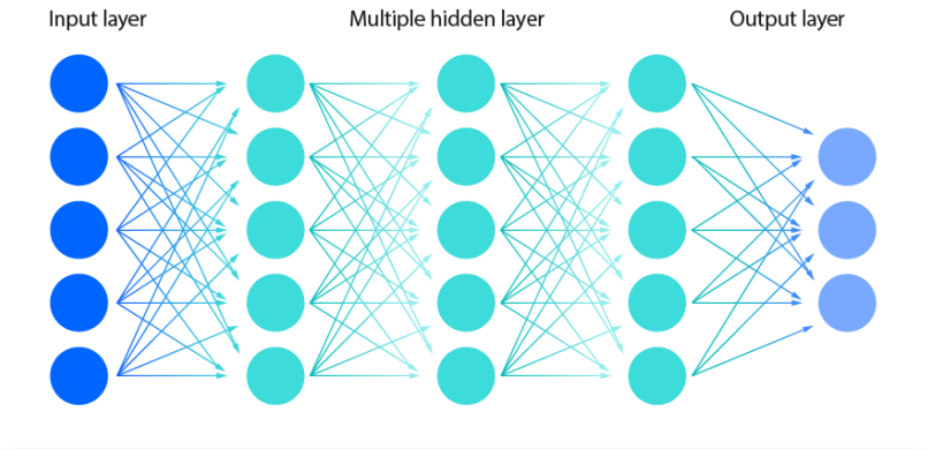



Figure 24: Simple Neural Networks Architecture

Each neuron in a layer performs the following operation:

$$z = w^T x + b, \quad a = \phi(z)$$

where $w$ is the weight vector, $x$ is the input, $b$ is the bias term, and $\phi$ is an activation function such as ReLU, sigmoid, or tanh. These activation functions introduce non-linearity, allowing the network to learn complex patterns that a purely linear model cannot. Here are the main activation functions:

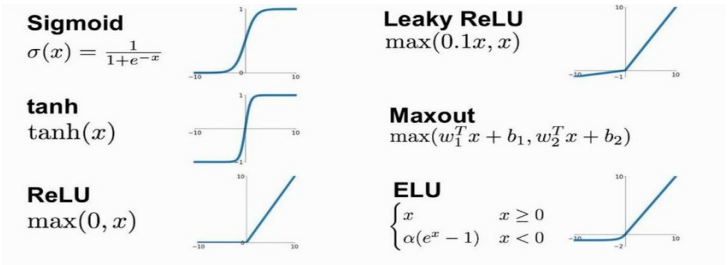



Figure 25: Various types of activation functions. From Rahul Jayawardana *et al.*, 2021 [14].

During forward propagation, input data is passed through the network layer by layer, with each layer transforming the data and extracting higher-level features. The final output layer produces the prediction based on these transformations.

After generating predictions, the network calculates the loss, a measure of how far its predictions are from the true labels. To improve performance, the network undergoes backward propagation, where the error is propagated in reverse, from the output to the

input, so that the network can compute gradients with respect to each weight. These gradients are then used to update the model parameters using optimization algorithms such as gradient descent.

The training process is repeated over multiple epochs, where each epoch represents a full pass (forward + backward propagation) through the training dataset. During training, key hyperparameters like the learning rate (which controls the size of weight updates), batch size (the number of samples processed before updating weights), and the number of hidden layers or neurons per layer significantly influence model performance and convergence. Choosing these hyperparameters carefully is essential to prevent underfitting or overfitting.

Specialized variants of Neural Networks have been developed for different data types, e.g. Convolutional Neural Networks (CNNs) are widely used for images data, while Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are designed to handle sequential data such as time series or text.

Neural networks are considered universal function approximators, meaning they can theoretically learn any mapping between inputs and outputs given enough data and capacity. They have achieved state-of-the-art results in fields such as computer vision, speech recognition, natural language processing, and even reinforcement learning applications like game playing. However, their effectiveness often depends on access to large datasets and huge computational resources. From Wikipedia [26].