الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

جامعة سعيدة د. مولاي الطاهر

كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية

قسم: الإعلام الآلي

Master's Thesis in Computer Science

Specialization:

Computer Modeling of Knowledge and Reasoning

Artificial Intelligence: Principles and Applications

Theme

For the Award of Start-up Project Diploma under Ministerial Decree 1275

Doc-IN: Automated Information Extraction and Insertion into Databases from Invoices

➢ Presented by:

- Mohamed Ilyes GUERAOUI
- Aymene Mokhtar BOUHELLAL
- Mohamed BOUZIANE

➢ Supervised by:

- Dr. Mostefai Abdelkader

Academic Year 2024-2025

Abstract

In today's data-driven landscape, manual invoice processing remains a time-consuming and error-prone task for many Algerian businesses. This project presents Doc-IN, an AI-powered smart invoice reader designed to automatically extract and structure data from scanned Algerian invoices. By leveraging computer vision and machine learning techniques, Doc-IN significantly reduces human intervention in the invoice management process.

The system operates through a lightweight software agent that processes invoice images, detects key fields such as supplier information, dates, product lines, quantities, and tax details, and inserts the structured data into a connected database. Doc-IN has been developed with adaptability in mind, making it compatible with commonly used relational database systems such as MySQL and PostgreSQL, widely adopted across Algerian enterprises. Additionally, Doc-IN is easy to integrate through its SDK with any stock management system, enabling seamless automation across existing workflows.

Targeted at retail stores, warehouses, pharmacies, and similar institutions, Doc-IN addresses a critical need for automation in administrative tasks. The project integrates multiple disciplines, including computer vision, backend development, and intelligent system design, under the guidance of an academic supervisor.

Ultimately, Doc-IN aims to enhance productivity, improve data accuracy, and modernize invoice handling practices across Algeria.

Acknowledgements

All praise is due to Allah, the Most Gracious, the Most Merciful. Alhamdulillah, with His help and guidance, we were able to complete this project. We are truly grateful for the strength, patience, and determination that Allah (Subḥānahu wa Taālā) granted us throughout this journey.

We would like to express our sincere thanks to our academic supervisor, Mr. Mostefai ABDELKADER, Assistant Professor at the University of Saida Dr. Moulay Tahar. His expert guidance, encouragement, and constructive feedback played a vital role in the success of our project.

We are also thankful to the Department of Computer Science and the entire faculty for providing the knowledge, tools, and environment necessary to carry out our work.

A special thank you goes to our families for their continuous support, prayers, and motivation. May Allah reward them for their love and patience.

We also acknowledge the efforts and commitment of our project team members — Mohamed Ilyes GUERAOUI, Aymene Mokhtar BOUHELLAL, and Mohamed BOUZIANE — whose collaboration and hard work made this project possible.

Finally, we thank everyone who helped us, directly or indirectly. May Allah (Subḥānahu wa Taālā) accept this effort, and may it serve as a source of benefit to others.

This work is a collective dedication from the Doc-IN team: **Mohamed Ilyes GUERAOUI**, **Aymene Mokhtar BOUHELLAL**, and **Mohamed BOUZIANE**.

To our academic supervisor, **Mr. Mostefai ABDELKADER**, for his valuable guidance, patience, and unwavering support.
To the **Université Dr Moulay Tahar – Saïda**, for providing the environment and resources that made this achievement possible.

Dedication

Mohamed Ilyes GUERAOUI

To my beloved **parents**, whose love, sacrifices, and endless support have guided me through every challenge this achievement is as much yours as it is mine.

To my **family**, thank you for your encouragement and belief in me throughout this journey.

To my dear **friends**, your presence, laughter, and support made the hardest moments lighter and the successes more meaningful.

To the inspiring **CSS Club team**, thanks for being my academic family and for constantly pushing boundaries with me.

And to my **future wife** wherever you are may this work be a humble reflection of the dedication and passion I strive to carry into every part of life.

Dedication

Aymene Mokhtar BOUHELLAL

To my **beloved family**, for your unconditional love, encouragement, and faith in me — this achievement is as much yours as it is mine.
To my **dear friends**, thank you for your unwavering presence and for making the hardest moments lighter with your support and smiles.

Dedication

Mohamed BOUZIANE

With immense pleasure, an open heart, and deep joy, I dedicate this work to those who are most dear, most respected, and truly exceptional in my life:

To my beloved **parents**, whose unconditional love and unwavering support have been the foundation of my journey.

To my two **sisters**, for their constant affection, and to my dear **brother Zakaria**, for his fraternal love and enduring presence.

To my esteemed thesis collaborators, **Mr. Gueraoui Mohamed Ilyes** and **Mr. Bouhellal Mokhtar Aymene**, for their commitment, perseverance, and the fruitful discussions that enriched this research.

To **Haddou Abdelatif Riadh Nabil**, a cherished friend and brother at heart, for his unwavering companionship and enlightened guidance along the way.

To my dearest friends, fellow travelers on this journey of effort and reflection, for their sincere friendship and shared moments that lightened the burden of hard work: **Mr. Charaf Mohamed**, **Mr. Abdelli Cheikh**, **Mr. Charef Mohamed**, **Mr. Houacine Mohamed Amine**, **Mr. Amari Tarek**, and **Mr. Kadi Abd Essamed**.

To the members of the **ECNO 2019 cohort**, with whom I share lasting bonds and valuable memories.

May this work reflect the deep gratitude I hold for each of you, for your direct or indirect contributions to the successful completion of this endeavor.

Contents

List of Figures

List of Tables

Keywords and Concepts

- **OCR (Optical Character Recognition):** Technology for converting printed or handwritten text in scanned images into machine-readable text.
- **PaddleOCR:** An open-source OCR system developed by Baidu, used in Doc-IN for fast and accurate text recognition.
- **Document AI:** AI systems designed to understand and process documents such as invoices, receipts, and forms.
- **Layout Analysis:** Identifying and segmenting structural components of a document layout, such as headers, tables, and totals.
- **Knowledge Distillation:** Compressing a large, pre-trained model's capabilities into a smaller model to optimize performance.
- **Line Item Detection:** Automatically extracting rows of itemized data (e.g., product name, quantity, unit price) from invoices.
- **Synthetic Dataset:** Artificially generated invoices used to augment or simulate real-world data for training and evaluation.
- **LayoutLMv3:** A vision-language transformer model for document understanding, used to align text, layout, and visual features.
- **Fine-tuning:** Training a pre-existing model on a specific dataset (e.g., Algerian invoices) to specialize it for a particular task.
- **Software Development Kit (SDK):** A packaged set of tools, libraries, APIs, and documentation to help developers integrate Doc-IN features easily.
- **API Layer:** The part of an SDK that exposes the core functionality to developers, typically through functions, methods, or endpoints.
- **CI/CD:** Continuous Integration and Continuous Deployment processes that automate testing and delivery of the SDK.

Chapter 1

General Introduction

1.1 Context

Processing invoices is a routine but important task in many businesses. These documents often arrive in different formats, with varying layouts and languages, which makes automatic handling challenging. Most companies still rely on manual entry or semi-automated systems that require a lot of human supervision. This not only slows things down but also introduces errors.

Recently, there has been significant progress in AI technologies, especially in document understanding. Tools like OCR engines, layout-based models, and transformer architectures have shown that it's now possible to go beyond simple text extraction and actually understand the structure of complex documents like invoices.

This project, called **Doc-IN**, was developed to address this exact problem. It combines OCR, and transformer models such as LayoutLMv3 to extract relevant information from invoices and insert it into databases. The system also includes a custom SDK to make integration easier and more adaptable. The goal is to reduce the need for manual work and help businesses process documents faster and more accurately.

1.2 Problematic and Motivation

Despite all the progress in document automation, processing invoices remains a difficult and repetitive task for many businesses. Invoices often come in different formats, with varying layouts, fonts. This makes it hard for generic OCR tools or prebuilt models to extract accurate data without some form of manual correction or review. As a result, businesses still spend time checking and editing extracted fields to avoid costly errors [10].

Manual data entry also comes with significant operational costs. It requires dedicated human resources, increases the risk of input mistakes, and slows down financial workflows such as payment processing or tax reporting.

Another issue is the lack of flexibility in most available tools. Many invoice extraction solutions don't allow customization, or they hide their logic behind closed systems.

This creates problems when the system needs to handle regional invoice formats, adapt to new templates, or integrate with other platforms. In some cases, these tools simply fail silently or miss important information without giving clear reasons [10].

This project, **Doc-IN**, was motivated by the need for something more adaptable and transparent. We wanted to build a tool that could actually work in real-world conditions, not just ideal ones. By combining OCR, layout-aware transformer-based models like LayoutLMv3, we aimed to create a system that understands the structure of invoices more deeply. Modern OCR engines can now handle documents with diverse font styles and formats, making them more suitable for real-world invoice variability.

We also focused on making the system reusable and easy to integrate, by designing a custom SDK. The idea was not just to automate invoice reading, but to give users more control, reliability, and long-term usefulness.

1.3 Objectives

The main goal of this project is to build a solution that actually works in real conditions, not just in controlled or ideal test cases. Our focus was on creating a system that can extract key information from invoices and insert it into a structured database, while handling the kinds of problems that come up in practice—especially in places like Algeria.

One of the main challenges we noticed early on was the lack of available datasets that represent how invoices actually look here. Most public datasets come from other countries and don't match the local formats. In Algeria, the structure can vary a lot between different companies. This makes it harder for standard models to work well out of the box.

To deal with that, our objective was to design a flexible pipeline that could adapt to different layouts and document types. We used OCR tools combined with layout-aware transformer models, and we also created a custom SDK so the system can be reused or integrated into other platforms.

In short, our main objectives were:

- Build a working pipeline that can extract invoice data and insert it into a database.
- Handle local challenges like varied layouts, and inconsistent formatting.
- Compensate for the lack of Algerian datasets using synthetic data and data augmentation.
- Create an SDK to make the solution easier to reuse and maintain.

- Make sure the system performs well in realistic, everyday use cases.

Chapter 2

Background and Theoretical Foundations

2.1 Artificial Intelligence (AI)

Artificial Intelligence (AI) is a multidisciplinary field within computer science concerned with building systems capable of performing tasks typically requiring human intelligence. These tasks include reasoning, learning, perception, language understanding, and decision-making. AI combines elements from logic, probability theory, optimization, and neuroscience to model intelligent behavior [17].

2.1.1 Core Modules of AI

AI systems are typically composed of several interconnected modules, each responsible for a specific dimension of intelligence [17]:

- **Knowledge Representation:** This module is concerned with encoding information about the world in a format that a computer system can utilize to solve complex tasks such as diagnosing a medical condition or planning a route. Common approaches include:
  - *Propositional and First-order Logic:* For formal reasoning.
  - *Semantic Networks, Frames, and Ontologies:* Used in expert systems and the Semantic Web [1].

  Knowledge graphs (e.g., Google's Knowledge Graph) are a modern extension of these concepts.

- **Search and Planning:** AI systems often operate in large or infinite state spaces and must decide how to act. Search algorithms explore these spaces efficiently. Examples include:
  - *Uninformed Search:* Breadth-first, depth-first.

  - *Informed Search:* A*, greedy search.
  - *Game Trees:* Minimax and Alpha-Beta pruning in adversarial environments.
  - *Planning:* STRIPS, PDDL-based planners, and heuristic-based systems for robotic movement and logistics [6].

- **Reasoning Systems:** These enable inference—drawing conclusions from known information. There are three major forms:

  - *Deductive Reasoning:* From general rules to innovation.
  - *Inductive Reasoning:* Deriving general principles from observations.
  - *Probabilistic Reasoning:* Using Bayesian networks, Markov logic networks, and fuzzy logic to deal with uncertainty.

  Applications include expert systems, autonomous agents, and diagnostic engines.

- **Learning Mechanisms:** Learning from data is central to modern AI. This includes:

  - *Supervised Learning:* Learning from labeled datasets.
  - *Unsupervised Learning:* Pattern discovery in unlabeled data.
  - *Reinforcement Learning (RL):* Learning via interaction with an environment (e.g., AlphaGo) [19].

  These are often implemented through machine learning models like decision trees, neural networks, and support vector machines.

- **Perception** Modules: These allow AI to interpret sensory inputs like vision, sound, and touch. Examples include:

  - *Computer Vision:* For interpreting images and videos (e.g., object detection, scene understanding).
  - *Speech Recognition:* Converting spoken language to text (e.g., Siri, Alexa).
  - *Sensor Fusion:* Combining data from multiple sensors in robotics.

- **Natural Language Processing (NLP):** NLP enables machines to understand, interpret, and generate human language. This includes:

  - *Syntax and Semantic Analysis:* Parsing and meaning extraction.
  - *Named Entity Recognition (NER), Machine Translation, and Sentiment Analysis.*
  - *Large Language Models:* Like BERT, GPT, and T5 for state-of-the-art performance in a variety of tasks.

2.2 Machine Learning (ML)

Machine Learning (ML) is a subfield of Artificial Intelligence focused on the development of algorithms that enable systems to learn from data and make predictions or decisions without being explicitly programmed. ML techniques have become foundational in numerous domains such as computer vision, natural language processing, bioinformatics, and finance [2, 3].

2.2.1 Core Modules of ML

A typical machine learning pipeline consists of several critical components, each contributing to the effectiveness and robustness of the final model:

- **Data Collection and Cleaning:** The first step involves gathering relevant data from various sources such as sensors, APIs, web scraping, or databases. Raw data is often noisy or incomplete, so preprocessing techniques such as:
  - *Missing value imputation*,
  - *Outlier detection*,
  - *Normalization and scaling*,
  - *Encoding categorical variables*

  are applied to improve data quality and ensure model compatibility.
- **Feature Engineering:** Features are the measurable properties or characteristics used for learning. Feature engineering includes:
  - *Feature selection:* Identifying the most relevant inputs (e.g., using correlation, mutual information).
  - *Feature extraction:* Transforming data into new representations (e.g., PCA, TF-IDF, embeddings).
  - *Domain-specific transformations:* Applying log transforms, interaction terms, etc.

  Good features often determine model success more than the algorithm choice itself [14].
- **Model Selection:** Depending on the problem type—classification, regression, clustering, etc.—different algorithms may be considered [2]:
  - *Linear Models:* Logistic regression, linear regression.

  - *Tree-based Models:* Decision trees, Random Forests, Gradient Boosting (e.g., XGBoost).
  - *Kernel Methods:* Support Vector Machines (SVMs).
  - *Neural Networks:* Especially for deep learning tasks.

  Cross-validation and hyperparameter tuning help determine the best-performing model.

- **Training and Validation:** Models are trained by minimizing a loss function over the training data (e.g., cross-entropy, mean squared error). Key practices include:
  - *Batch training vs online learning*,
  - *Regularization:* L1/L2 penalties to avoid overfitting,
  - *Validation Split:* Separating validation data to tune parameters without leakage.

  Techniques like early stopping, dropout, and data augmentation also improve generalization.

- **Evaluation:** Performance is assessed using appropriate metrics based on the problem [14]:
  - *Accuracy, Precision, Recall, F1-score, ROC-AUC* (for classification),
  - *MSE, RMSE, MAE* (for regression),
  - *Silhouette score, Davies–Bouldin index* (for clustering).

  Confusion matrices and learning curves are often used for deeper diagnostic analysis.

- **Deployment:** After achieving satisfactory performance, the model must be integrated into real-world applications. This involves:
  - *Model serialization:* Using formats like ONNX, Pickle, or TensorFlow SavedModel,
  - *API wrapping:* Exposing the model through REST or gRPC endpoints,
  - *Monitoring:* Ensuring ongoing performance with tools like Prometheus or custom logging,
  - *Retraining pipelines:* For model updating in dynamic environments (e.g., data drift).

2.3 Deep Learning (DL)

Deep Learning (DL) is a specialized subfield of Machine Learning that focuses on algorithms inspired by the structure and function of the brain's neural networks [15]. It enables automatic learning of hierarchical feature representations through multiple layers of abstraction, making it particularly effective in complex tasks such as image classification, natural language understanding, and speech recognition [7].

2.3.1 Core Modules of DL

A typical deep learning pipeline incorporates several core components that collectively enable end-to-end learning from raw data:

- **Neural Architectures:** Deep learning models are composed of layers of interconnected nodes (neurons) [7]. Different architectures are designed for different data types and tasks:
  - *Convolutional Neural Networks (CNNs):* Primarily used for image and spatial data processing [13]. They utilize filters for local feature extraction and pooling layers for dimensionality reduction.
  - *Recurrent Neural Networks (RNNs) and LSTMs:* Suitable for sequential data such as time series and natural language. LSTMs mitigate the vanishing gradient problem in long sequences [8].
  - *Transformers:* Leverage attention mechanisms to model long-range dependencies in sequences [20, 4]. Widely used in NLP (e.g., BERT, GPT) and vision (e.g., Vision Transformers).
- **Activation Functions:** These functions introduce non-linearity into the network, allowing it to learn complex mappings. Common examples include:
  - *ReLU (Rectified Linear Unit):* Popular for its computational efficiency and performance.
  - *Sigmoid and Tanh:* Historically used, especially in early networks, though susceptible to vanishing gradients.
  - *Softmax:* Often used in the final layer for classification tasks to produce probability distributions.
- **Loss Functions:** Loss functions measure the discrepancy between predicted and true values, guiding the optimization process:
  - *Cross-Entropy Loss:* Used for classification tasks.

  - *Mean Squared Error (MSE):* Commonly used in regression tasks.
  - *Huber Loss:* Combines MSE and MAE for robustness to outliers.

- **Optimization:** Training involves adjusting weights to minimize the loss function using algorithms such as:
  - *Stochastic Gradient Descent (SGD):* A widely used baseline method.
  - *Adam (Adaptive Moment Estimation):* Combines momentum and adaptive learning rates.
  - *RMSprop:* Designed for non-stationary objectives, often used in RNN training.

  The backpropagation algorithm computes gradients of the loss function with respect to model parameters, enabling learning through iterative updates.

- **Regularization:** Deep networks can easily overfit, especially with small datasets. Regularization techniques are employed to improve generalization:
  - *Dropout:* Randomly deactivates neurons during training to prevent co-adaptation [18].
  - *Weight Decay (L2 Regularization):* Penalizes large weights to encourage simpler models.
  - *Batch Normalization:* Normalizes inputs to each layer, accelerating training and improving stability.

- **Transfer Learning:** Transfer learning leverages pretrained models (e.g., ResNet, BERT) trained on large-scale datasets like ImageNet or Wikipedia [16]. The pretrained weights are fine-tuned on domain-specific tasks with limited data, significantly reducing computational cost and training time.

2.4 Computer Vision (CV)

Computer Vision (CV) is a field of Artificial Intelligence (AI) that enables machines to interpret and understand visual information from the world. It emulates the human visual system to some extent, aiming to automate tasks that require visual cognition, such as image classification, object detection, and scene understanding.

2.4.1 Core Modules of CV

The computer vision pipeline generally consists of several key modules, each responsible for a specific transformation or interpretation of visual data:

- **Image Acquisition:** This is the first step involving the collection of visual data using devices such as CCD/CMOS cameras, scanners, or even smartphones. The quality and resolution of acquisition significantly influence downstream performance.

- **Preprocessing:** Raw images are rarely ready for direct analysis. Preprocessing enhances image quality and prepares it for further steps. Common operations include:

  - *Noise Reduction:* Using filters like Gaussian blur or median filtering.
  - *Grayscale Conversion:* Reduces complexity by converting color images to shades of gray.
  - *Resizing and Normalization:* Standardizes image dimensions and pixel values for neural networks.

- **Feature Extraction:** This step transforms images into a set of numerical descriptors capturing essential patterns. Traditional techniques include:

  - *SIFT (Scale-Invariant Feature Transform):* Detects scale- and rotation-invariant keypoints.
  - *HOG (Histogram of Oriented Gradients):* Captures edge orientations, useful for pedestrian detection.

  In deep learning, convolutional neural networks (CNNs) learn these features automatically during training [13].

- **Object Detection & Recognition:** The goal is to localize and categorize objects in the image. Prominent models include:

  - *YOLO (You Only Look Once):* Real-time object detection in a single neural network pass.
  - *Faster R-CNN:* Combines region proposal and classification in a two-stage detector.

- **Segmentation:** This involves dividing the image into coherent regions for analysis. Two major types exist:

  - *Semantic Segmentation:* Classifies each pixel into a category (e.g., road, car, sky).
  - *Instance Segmentation:* Differentiates between distinct objects of the same class (e.g., two people).

Notable architectures include U-Net, Mask R-CNN, and DeepLab.

- **OCR (Optical Character Recognition):** Extracts text from images, a vital component in document analysis. Modern OCR uses CNN-LSTM hybrids or Transformer-based models (e.g., TrOCR, Donut) for handling printed and handwritten text with high accuracy.

2.5 Building Software Development Kits (SDKs)

A Software Development Kit (SDK) is a comprehensive set of tools, libraries, documentation, and best practices designed to help developers integrate and utilize a specific service or product. SDKs abstract away low-level implementation details and expose clean interfaces, thereby accelerating development and ensuring consistency across applications [5].

2.5.1 Core Modules of SDK Design

A well-designed SDK is modular, secure, easy to use, and maintainable. The following components are typically involved:

- **API Layer:** This is the main interface exposed to developers. It typically consists of:
  - *Function calls or class methods* wrapping the business logic.
  - *REST or gRPC client interfaces* in web-based SDKs.
  - *Error objects and response handlers* for consistent behavior across platforms.

  Good SDKs offer a developer-friendly abstraction that hides the complexity of remote calls, retries, and data formatting.

- **Request Handling:** This layer processes incoming input and prepares it for execution or transmission. Core responsibilities include:
  - *Input validation and type-checking*,
  - *Serialization and deserialization* (e.g., JSON, Protobuf),
  - *Error handling and response parsing.*

  This ensures robustness and reduces integration errors in client code.

- **Authentication:** Secure interaction with APIs or resources is vital. SDKs typically implement:

  - *API Key headers or tokens*,
  - *OAuth 2.0* flows for user authorization,
  - *JWT (JSON Web Tokens)* for secure session management.

  Security should be baked into the SDK, not left for the developer to implement manually.

- **Documentation:** High-quality documentation is critical for adoption and usability. It includes:
  - *Quickstart guides and tutorials*,
  - *Inline comments and docstrings*,
  - *Code examples, FAQs, and use-case references.*

  Tools like Sphinx (Python), JSDoc (JavaScript), or Typedoc (TypeScript) are commonly used to generate API docs.

- **Versioning and Packaging:** To ensure long-term maintainability and easy distribution:
  - Semantic Versioning (SemVer) helps track breaking changes.
  - Packaging tools like `pip`, `npm`, `maven`, or `cargo` distribute SDKs.
  - Release notes, changelogs, and migration guides are essential for developer trust.

- **Testing and CI/CD:** Quality assurance is enforced via:
  - *Unit and integration tests* using frameworks like Pytest, Mocha, or JUnit,
  - *Mocking/stubbing APIs* to simulate external services,
  - *CI/CD pipelines* for automatic testing, packaging, and publishing (e.g., GitHub Actions, GitLab CI, CircleCI).

  These practices ensure reliability and speed up the release cycle without introducing regressions.

2.6 Document Understanding in AI

Understanding documents is a foundational challenge in artificial intelligence, especially in domains like business automation, healthcare, and legal technology, where unstructured or semi-structured documents are ubiquitous. Document Understanding (DU) refers to the process of extracting, interpreting, and structuring information

from diverse document types — including scanned invoices, PDFs, handwritten forms, or printed reports. Unlike plain text processing, DU must deal with a combination of textual, visual, and spatial information simultaneously.

Early efforts in document understanding focused on rule-based systems, where layout templates and regular expressions were used to locate fields in specific positions [12]. These approaches, while effective in constrained settings, were brittle and unable to handle real-world variability in layout, language, or design. As digitization accelerated, the need for models that could generalize across document styles and formats became increasingly urgent.

Modern DU systems benefit greatly from advances in deep learning, particularly the development of transformer-based models capable of understanding both text and layout. These models treat documents not just as sequences of words but as visual objects with structural hierarchies. This is especially important for semi-structured documents like invoices or forms, where information is organized spatially rather than in natural language order.

The combination of computer vision and natural language processing (NLP) has also enabled multimodal models that process both text and layout features. For example, models like LayoutLM and its successors incorporate bounding box coordinates and visual embeddings alongside word tokens, allowing them to better capture the semantics of complex layouts [22].

Despite these advances, challenges remain. Noisy data, poor scan quality, and diverse regional formats can significantly impact extraction accuracy. Therefore, DU research continues to explore methods to improve robustness, interpretability, and adaptability of these models to new domains.

Document understanding is not only a technical challenge but also a practical necessity for enabling intelligent automation in the enterprise — especially for high-volume tasks like invoice processing, which demand accuracy, and efficiency.

Algerian invoices often lack any standard layout, with key information appearing in unpredictable positions and formats. This irregularity, along with elements like stamps, and dense tables, makes them particularly difficult to process automatically.

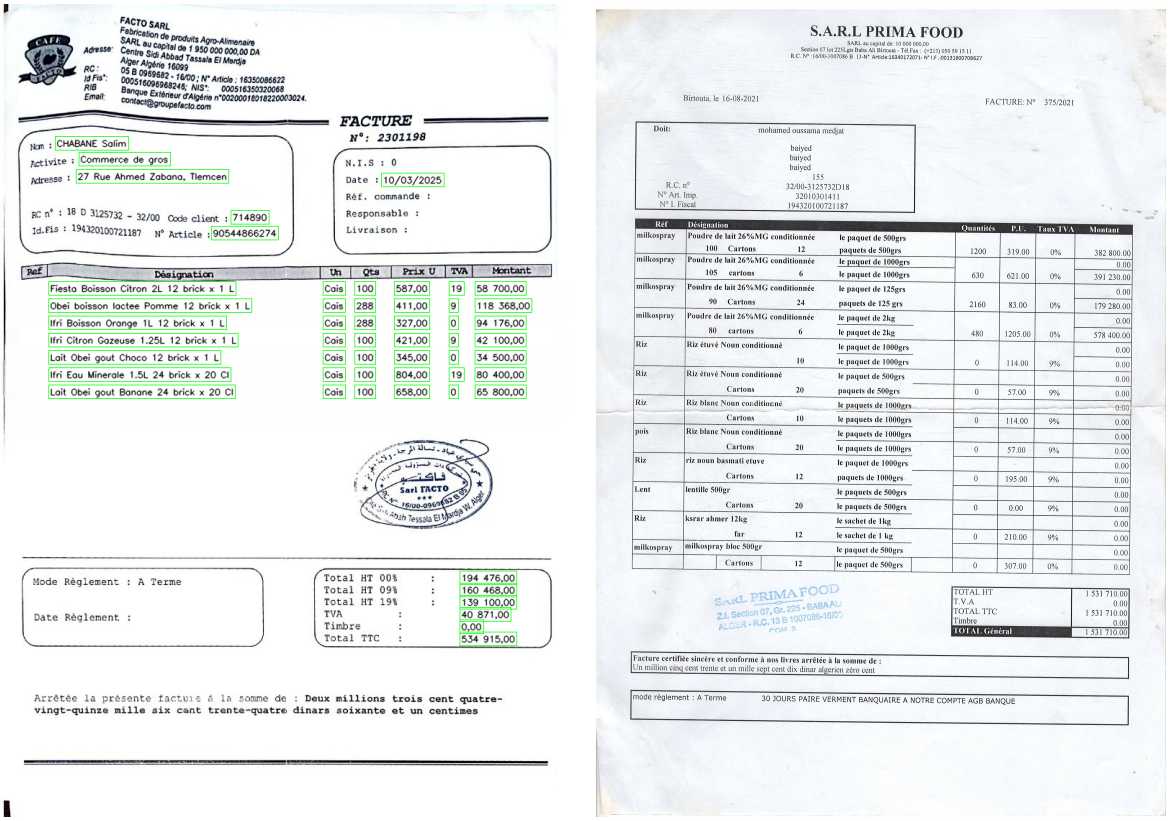

Figure 2.1: Examples of Algerian invoices.

2.7 OCR Techniques

Optical Character Recognition (OCR) is a core component in document understanding systems. It enables the automatic detection and conversion of text from scanned images, PDFs, or photographs into machine-readable formats. OCR is especially critical in the context of invoices, where key information such as item descriptions, prices, and totals often appear in various fonts, orientations, or layouts.

Early OCR systems relied on simple pattern matching and template-based approaches, which made them fast but highly sensitive to noise, font variability, and image quality. These classical methods struggled with real-world documents, particularly those containing non-standard fonts, multilingual text, or handwritten annotations.

Modern OCR techniques have seen significant advances with the integration of deep learning and computer vision. State-of-the-art systems now employ convolutional neural networks (CNNs), recurrent layers, and attention mechanisms to improve character-level recognition across diverse inputs. Frameworks like Tesseract (Google), PaddleOCR (Baidu), and EasyOCR have become widely used in academic and industrial applications. These systems are trained on large-scale datasets and are capable of handling multiple languages, irregular layouts, and even rotated or distorted text.

PaddleOCR, in particular, stands out for its combination of detection and recogni-

tion modules, support for multilingual text, and a modular architecture that integrates well into larger pipelines. It performs robustly even on noisy scans and supports text detection in complex layouts—making it a strong candidate for invoice processing systems. EasyOCR, while lightweight and easy to deploy, sometimes sacrifices accuracy for speed. Meanwhile, Tesseract remains a popular open-source solution, especially after its LSTM-based improvements in version 4, but it can struggle with layout-sensitive tasks and non-standard fonts.

One of the key challenges in OCR for invoices is that documents often include varied typography, table-like structures, and non-text elements like logos or stamps. These factors can confuse OCR models that are not layout-aware. To overcome this, hybrid approaches combine OCR output with layout models such as LayoutLMv3, which help contextualize recognized tokens based on their spatial positions in the document.

In summary, OCR is indispensable for transforming raw invoice images into usable text, but it is most effective when paired with models that understand structure. Selecting the right OCR engine depends heavily on the document's complexity, language, and layout characteristics.

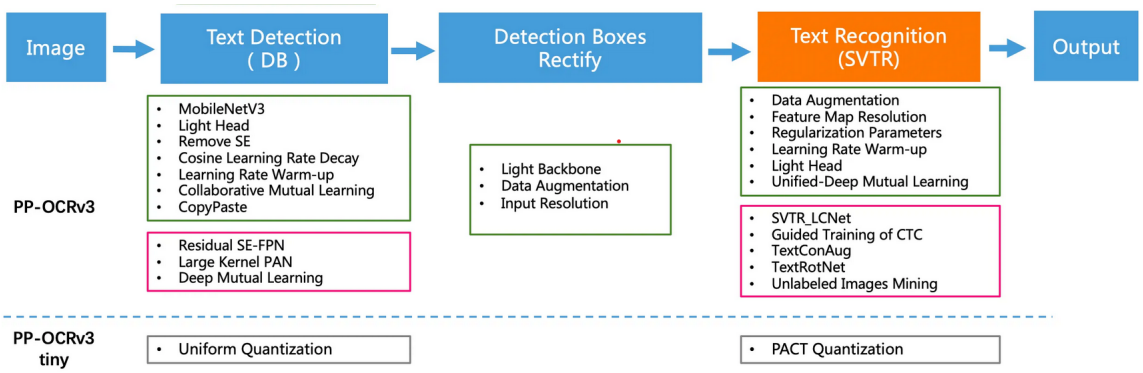

Figure 2.2: PaddleOCR pipeline architecture.

2.8 Layout Analysis with LayoutLMv3

LayoutLMv3 is a powerful model designed to enhance document understanding by jointly learning from text, layout, and visual features [21]. This makes it especially effective for structured documents like invoices. Below are the key contributions and reasons for its adoption in our project:

- **Multimodal Fusion:** LayoutLMv3 simultaneously incorporates textual, spatial (layout), and visual information, enabling it to understand not just what the words are.
- **2D Positional Embeddings:** Unlike traditional models that process text linearly, LayoutLMv3 uses two-dimensional embeddings that capture the exact

placement of tokens, making it better suited for understanding tables, columns, and key-value relationships.

- **Visual Backbone Integration:** It uses a visual encoder (e.g., ResNet or Swin Transformer) to extract image-level features [4]. This helps in identifying non-textual elements such as lines, boxes, and logos that structure the document but are invisible to OCR.

- **Better Performance on Noisy Layouts:** The model is robust to variations in fonts, orientations, and alignment—common issues in real-world invoices that often degrade the accuracy of plain OCR systems.

- **Pre-training on Document Datasets:** LayoutLMv3 is pre-trained on large-scale document datasets, enabling it to generalize well across different formats and languages without requiring massive labeled datasets.

- **Effective for Key Information Extraction:** In our use case, LayoutLMv3 significantly improved the recognition and alignment of fields like total amounts, company names, and dates—especially when paired with OCR engines.

- **Challenges and Limitations:** LayoutLMv3 is a powerful model, no doubt — but like anything, it comes with trade-offs. One of the main challenges is that it needs quite a bit of computing power to run, especially during training or real-time use. So, if you're trying to deploy it on something lightweight — like an edge device or in a live system — that could be a problem but this is not our use case.

  Another thing to keep in mind is the built-in OCR. While it's handy to have OCR integrated, it doesn't always match the performance of more specialized tools like PaddleOCR. In practice, PaddleOCR often does a better job with tricky fonts, low-quality images, or documents in different languages. So, depending on your use case, it might actually make sense to combine LayoutLMv3 with an external OCR engine to get the best of both worlds.

These capabilities make LayoutLMv3 a solid foundation for building invoice automation systems that are layout-sensitive, adaptable, and more reliable than traditional pipeline-based approaches.

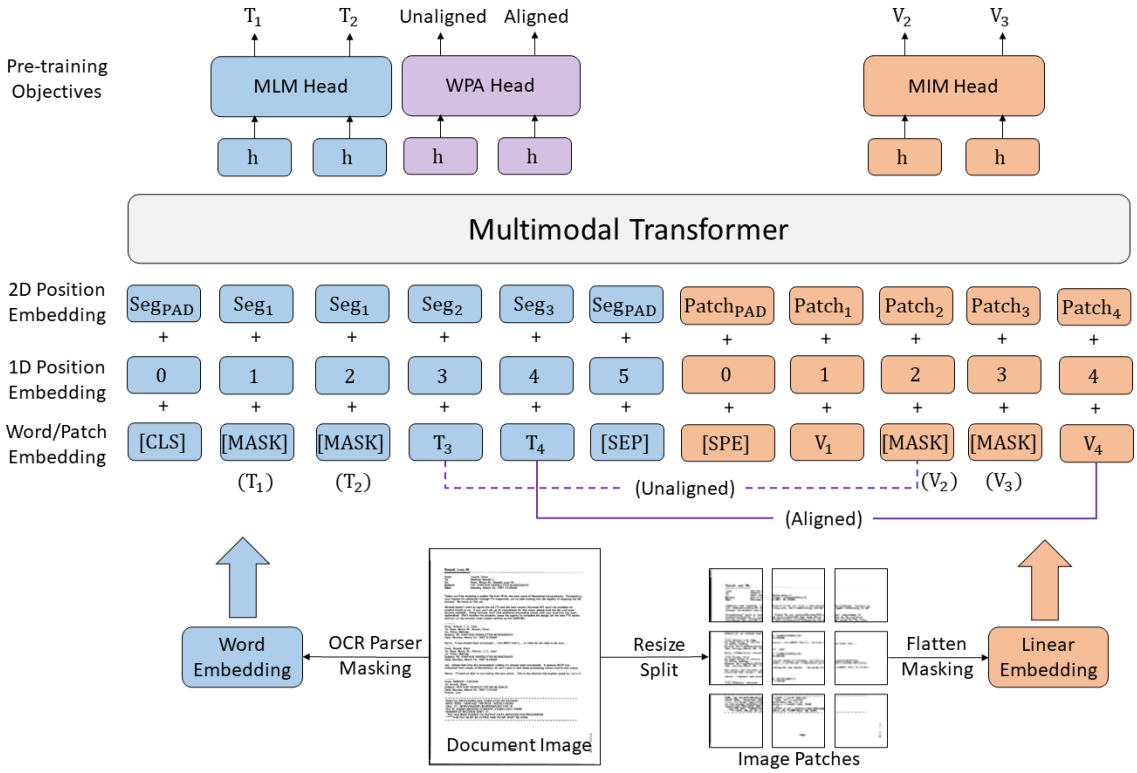

Figure 2.3: Overview of the LayoutLMv3 architecture.

2.9 Transformers for Document Processing

Transformer-based models have significantly advanced the field of document processing, thanks to their ability to capture contextual relationships across large text sequences [20]. In this project, we experimented with **Mistral**, an open-weight transformer model, to evaluate its capabilities in extracting structured information from invoices. Below is a breakdown of our rationale, process, and findings:

- **Why Transformers for Documents?** Transformers have become the foundation of modern NLP due to their scalability and attention mechanisms, which allow them to model long-range dependencies—crucial for understanding full documents.

- **About Mistral:** Mistral is a modern decoder-style transformer model optimized for efficient inference and competitive performance on a range of tasks [11]. It was a strong candidate for experimenting with personalized, domain-specific fine-tuning due to its lightweight architecture and open accessibility.

- **Initial Motivation:** Our goal was to fine-tune Mistral to understand and extract key fields from invoices in French (and optionally Arabic), using synthetic data generated from Algerian invoice formats. We hoped that Mistral's autoregressive capacity could adapt to these structured formats through targeted training.

- **Main Limitation Encountered:** Despite our efforts, the main bottleneck was the **lack of sufficient domain-specific data** required for effective fine-tuning or knowledge distillation. Unlike generic NLP tasks, document extraction tasks need detailed, annotated examples—and generating high-quality labeled invoice data at scale proved challenging.
- **Impact on Results:** Without enough real or well-annotated synthetic examples, Mistral struggled to learn field-specific patterns such as totals, TVA values, or nested product tables. Its outputs lacked precision, and often misinterpreted structural elements.
- **Comparison with Layout-Aware Models:** In contrast, models like LayoutLMv3—designed with spatial awareness—performed better out of the box, even with less supervision. This reinforced our understanding that language-only models like Mistral are not ideal for purely layout-driven tasks.
- **Key Takeaway:** Mistral can be valuable in downstream tasks (e.g., explaining or validating extracted data in natural language), but its effectiveness in structured field extraction remains limited without significant investment in training data. This insight shaped our decision to rely on hybrid pipelines combining OCR, layout-based vision models, and rule-based post-processing.
- **Future Possibility:** With access to richer datasets or domain-specific pretraining, Mistral—or similar decoder-based models—could play a stronger role in end-to-end invoice automation.

2.10 Existing Solutions and Their Limitations

Several tools and platforms have been developed to tackle invoice information extraction, each with varying degrees of accuracy, flexibility, and transparency [10]. While these systems have made strides in automating document workflows, they are not without significant limitations—especially when applied on non common formats of invoices.

- **Closed-Source Commercial Solutions (e.g., Docparser, Rossum, Kofax):**
  - Provide relatively good accuracy on common English invoice templates.
  - Often underperform on non-standard layouts, multilingual formats, or low-resource regions like North Africa.
  - Require expensive subscriptions and limit scalability for smaller businesses.

- **Modern Layout-Aware AI Models (e.g., LayoutLM, Donut):**
  - Offer better performance on complex document tasks by combining visual, textual, and layout signals.
  - Require large amounts of annotated training data, which is often unavailable for specialized domains like Algerian invoices.
  - May still struggle with generalization to unseen layouts or noisy scans.
  - High computational costs during training and inference can be a barrier for practical deployment.
- **End-to-End Systems (e.g., Google Document AI, Azure Form Recognizer):**
  - Provide APIs that work well for common invoices formats.
  - Offer limited flexibility for schema customization or user-defined fields.
  - Depend on cloud infrastructure—raising privacy, latency, and cost concerns.
- **Key Limitation Across the Board:**
  - Most systems are built and trained using Western or standardized invoice formats.
  - Many top-performing models require intricate dependency chains and custom environments, making "plug-and-play" integration difficult.

These limitations highlighted the need for a more flexible, open, and modular solution—one that could be tailored to regional business practices and integrated directly into custom platforms. **Doc-IN** was designed to address these gaps by combining layout-aware vision models, OCR, and a developer-friendly SDK architecture.

Chapter 3

Invoice Information Extraction: Problem Space

3.1 Why Invoices? Importance and Challenges

Invoices serve as the primary record of a transaction between a buyer and a seller, carrying both financial and legal significance. They are used for accounting, auditing, tax reporting, and cash flow management, making accurate data capture critical for business operations and regulatory compliance. Manual invoice processing, however, is time-consuming and error-prone—errors can lead to payment delays, incorrect tax filings, and strained supplier relationships [10].

Key challenges that make invoice processing difficult include:

- **Semi-structured Format:** Invoices are neither fully structured (like databases) nor purely free text; their layouts vary widely across vendors and regions, hindering generic parsing methods.
- **Visual Variability:** Differences in fonts, logos, color schemes, and graphical separators (lines, boxes) add noise, making text detection and segmentation more complex.
- **Document Quality Issues:** Scanned or photographed invoices can suffer from skew, blur, low contrast, or uneven illumination, all of which degrade OCR performance.
- **Data Validation Needs:** Beyond text extraction, invoices require validation of relationships (e.g., sum of line items equals total amount) to ensure data integrity and trigger human review when discrepancies arise.

3.2 Invoice Structure and Key Fields

Invoices serve as essential financial documents that formally record transactions between sellers and buyers. For automated information extraction systems, understanding the typical structure and expected key fields within an invoice is a foundational step. Although there is no universally enforced layout, certain elements are commonly present across most invoice formats, providing a semi-structured framework for AI models to learn from.

1. Document Header

The top section of most invoices contains administrative metadata that uniquely identifies the document and sets its temporal context. Key elements include:

- **Invoice Number** – A unique alphanumeric or numeric identifier.

2. Supplier and Client Information

Below or alongside the header, this section outlines the parties involved in the transaction. These fields are essential for legal and auditing purposes:

- **Supplier (Issuer)** – Name, logo, address, tax identification number, and sometimes commercial registry codes.
- **Client (Buyer)** – Name, address, and client reference codes.

This information may be organized in blocks or aligned side-by-side depending on template design.

3. Line Items (Product Table)

The body of the invoice typically contains a tabular structure listing the individual items being billed. It is the most content-heavy section and often the most error-prone in extraction tasks. Typical fields per row include:

- **Description of Product** – May contain free-form text, which varies greatly in length and format.
- **Quantity and Unit of Measure** – E.g., “2 pcs”, “1.5 kg”.
- **Unit Price** – Can appear with or without currency symbols.
- **Tax (TVA)** – Sometimes embedded in total, sometimes listed separately.
- **Total Line Amount** – Computed as Quantity $\times$ Unit Price (plus tax if applicable).

4. Totals Section

Near the bottom of the invoice, this section summarizes the financials:

- **Subtotal (HT)** – Sum of all line items before taxes.
- **Tax Amount (TVA)** – Can vary by product or be applied globally.
- **Discounts** – Sometimes listed as separate line.
- **Total Amount Due (TTC)** – Final payment amount, which must be correctly parsed for billing systems.

Visual indicators like bold fonts, boxed totals, or larger font sizes are often used here, and can assist layout-aware models.

Key Extraction Challenges:

- **Inconsistent Layouts:** Some vendors use Excel-generated invoices, others use scanned paper forms. This affects readability and extraction.
- **Non-standard Labels:** The same field might be labeled “TVA”, “Taxe”, “VAT”, or even just “T.”, depending on the business.
- **Spelling Errors and Text Inconsistencies:** Invoices often contain misspelled or inconsistent terms, such as "grek 100gram" instead of "Greec-100g", due to human error or OCR inaccuracies. These variations complicate reliable data extraction and require fuzzy matching or correction mechanisms to ensure accurate results.
- **Currency and Format Variability:** Numbers might use commas or dots as decimal separators; currencies might be implicit or indicated with symbols.
- **Structural Ambiguities in Tabular Invoice Data:** In Algerian invoices, structural inconsistencies can pose significant challenges for extraction models. Some line items show a quantity of zero and a total of zero. These are usually not real purchases but mistakes from invoice templates or internal system exports. Additionally, some invoices contain redundant entries, such as repeated item names or duplicated amounts across different rows. Although these elements may appear structurally valid, they introduce semantic noise that can lead to incorrect or duplicated extraction. Handling such ambiguities is essential for reliable information retrieval in local invoice formats.

Why Structure-Aware Models Matter:

Traditional OCR systems process text linearly, often ignoring the spatial context that is critical in invoices. Layout-aware transformers like LayoutLMv3 represent a significant

advancement by integrating textual, visual, and positional information. This allows them to infer relationships between elements like “TVA” and the number beside it, or to associate column headers with their corresponding data rows.

3.3 Review of Tools and Approaches (OCR, Layout Models, Transformers)

The problem of extracting structured data from invoices has been tackled using a variety of tools and approaches that combine optical character recognition (OCR), layout-aware models, and deep learning-based transformers. Each category of tools plays a unique role, and understanding their advantages and limitations is critical for designing a robust pipeline.

1. Optical Character Recognition (OCR)

OCR is the foundational step in document processing. It converts scanned images or PDFs into machine-readable text.

- **Tesseract OCR**: An open-source OCR engine developed by Google, known for its robustness in recognizing printed English text. However, it struggles with complex layouts or multilingual documents (especially Arabic script).
- **EasyOCR**: A lightweight, Python-based OCR library that supports over 80 languages, including Arabic and French. While easier to use, it sometimes misaligns text blocks when working with low-resolution documents.
- **PaddleOCR**: Developed by Baidu, this library supports multilingual detection and includes layout analysis capabilities. It performs well on noisy images and mixed-language documents, making it a strong choice for real-world invoice processing.
- **Limitations:** Traditional OCR engines treat documents as linear text and often fail to preserve spatial relationships, which are essential for understanding structured documents like invoices.

2. Layout-Aware Models

To address OCR’s limitations in layout understanding, a new family of models integrates both textual and spatial information:

- **LayoutLM and LayoutLMv2**: These models introduced the concept of embedding spatial coordinates into the attention mechanism of transformers [22]. They learn how words are placed in relation to one another, which improves extraction in structured formats like tables and forms.
- **LayoutLMv3**: The latest version improves performance by integrating vision transformers (ViTs) to process raw pixels alongside text and layout [21]. It shows significant gains in complex document tasks (e.g., FUNSD, CORD, and invoice parsing).
- **Donut**: A document-understanding model that skips OCR altogether by directly learning from raw images using an encoder-decoder vision transformer. While promising, it requires large-scale training and is sensitive to domain shifts.
- **Limitations:** Layout models require annotated spatial data and large GPU resources to fine-tune effectively. They are also sensitive to misalignments in OCR outputs or document noise.

3. Transformer-Based Language Models

Transformers revolutionized NLP by introducing attention mechanisms that allow models to learn long-range dependencies across text.

- **BERT, RoBERTa, and DeBERTa**: These pre-trained transformers are widely used for token classification and named entity recognition. However, they lack layout awareness and are not directly suitable for document parsing tasks.
- **Mistral**: A high-performance, decoder-only transformer optimized for generation tasks [11]. In our case, we experimented with Mistral for invoice field generation, but due to limited labeled data, fine-tuning or knowledge distillation didn't work very well.

4. Hybrid Pipelines and SDK Approaches

Modern systems combine multiple tools into cohesive pipelines:

- OCR is used for initial text recognition.
- Layout models (like LayoutLMv3) add spatial intelligence.
- Transformers handle contextual understanding and field classification.
- A custom SDK allows the system to be packaged, deployed, and integrated easily across platforms.

Conclusion:
While great progress has been made, no single approach solves all document processing challenges. Real-world deployments must balance accuracy, speed and flexibility factors we considered deeply when designing the Doc-IN pipeline.

3.4 Our Vision for Doc-IN

- **Built for Real-World Use:** Doc-IN is designed to function in realistic conditions, particularly in Algeria where invoices vary in layout and structure.
- **Hybrid Intelligence Approach:** Combines OCR with layout-aware models like LayoutLMv3 to understand the spatial arrangement of invoice elements, improving field detection accuracy.
- **Custom SDK Architecture:** Includes a reusable, modular SDK that separates components (OCR, model, postprocessing), allowing customization, easier debugging, and platform integration.
- **Future Vision:** Plans include a mobile scanner app, offline capability, and multilingual support, especially tailored for underrepresented regions like North Africa.

Chapter 4

Data Preparation and Methodology

In this chapter, we first review the theoretical foundations of transfer learning—including fine-tuning and knowledge distillation—then describe our concrete experiments with Donut, Mistral and LayoutLMv3. Finally, we present the methodology we chose, with a comparative table of all approaches tried.

4.1 Data Collection Selection and Annotation

- **Absence of Public Datasets:** One of the first challenges encountered was the complete lack of open, labeled datasets representing Algerian invoices across different formats and vendors.
- **Collection from Real World:** We gathered a small sample of real invoices from local businesses (with permissions), covering a range of industries, layouts and formats.
- **Privacy Considerations:** To ensure compliance with data privacy norms, all sensitive personal or business information was anonymized or synthetically replaced before use in training.
- **Manual Labeling:** These real invoices were annotated manually using labeling tools such as `Label Studio`, defining fields like `Client`, `Total TTC`, `TVA`, `Date`, and `Produit`.
- **Layout Diversity:** Due to limited access to large-scale annotated datasets and the time-consumption of manual data labeling, we focused on carefully selecting a smaller yet diverse set of invoice samples. This strategic curation was intended to maximize variability and support better generalization during model training.
- **Format Standardization:** All collected documents were converted to a uniform input format (e.g., PNG), and associated labels, tokens and bounding boxes were stored in JSON following a strict schema in order to train LayoutLMv3.

- **Evaluation-Ready Splits:** The dataset was divided into training and test sets using holdout approach, ensuring layout variety in each split to better evaluate model generalization.

4.2 Data Augmentation and Synthetic Data Generation

A significant challenge in training document understanding systems, especially for domain-specific tasks like invoice extraction, is the scarcity of labeled data. This is particularly true in regions like Algeria, where publicly available, annotated invoice datasets are virtually nonexistent. To overcome this limitation, our project incorporated extensive data augmentation and synthetic data generation strategies.

- **Synthetic Invoice Generation:** To scale our dataset, we built a custom script to generate hundreds of synthetic invoices. These documents simulate different real-world scenarios, fonts, templates, and noise.
- **Template-Based Synthetic Generation:** We created multiple invoice samples simulating real-world formats using various layouts, fonts (product names, prices, company names, etc.). Synthetic invoices were employed solely for model training to simulate layout and content variability. For testing, however, we relied on real-world invoice data to ensure that the evaluation metrics accurately reflect the model's performance.
- **Data Augmentation:** Using image processing techniques, we introduced distortions, noise, blur and skewing to provide larger dataset and help the model to generalize better.
- **Use of Generation Tools:** Tools such as Faker and OpenCV were combined with Python scripts to automate large-scale generation of synthetic invoices, paired with ground-truth JSON annotations.

Through these methods, we generated over 500 synthetic annotated invoices, which served as the foundation for model training and fine-tuning. This approach not only compensated for the lack of Algerian datasets but also provided a scalable way to continuously expand and adapt the dataset as the system evolves.

4.3 Dataset Schema Definition

A well-defined notation and schema are critical to ensure that extracted invoice data can be validated, stored, and consumed reliably by downstream applications. In DocIN, we represent each invoice as an image I and an associated JSON document D, following these conventions:

- **Data Preparation:** To train the LayoutLMv3 model, we first constructed a set of JSON annotation files associated with each invoice image. These JSON files followed a consistent and structured format, where each token was annotated with its corresponding bounding box and class number.
- **Token and Bounding Box Encoding:** Each token extracted from the invoice was paired with a bounding box representing its spatial position on the image. This spatial information is essential for LayoutLMv3, which leverages both textual and layout features.
- **NER Tagging:** The semantic class of each token was encoded as a numerical NER tag. These tags represent the various fields of interest (e.g., invoice number, date, total amount) in the document.
- **Label Mapping:** After the training phase, a label-to-ID mapping (label2id) was applied to associate each numerical class ID with its corresponding human-readable label. This step ensured that the model's output could be interpreted clearly by human readers and aligned with the original annotation schema.

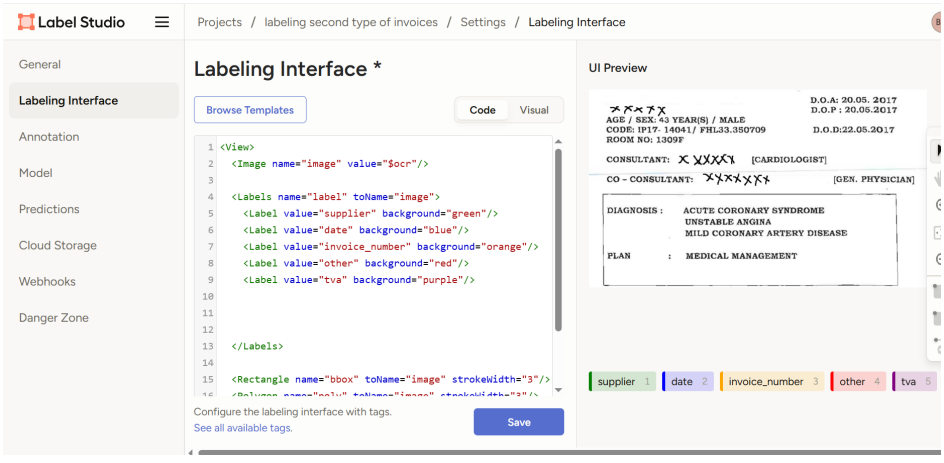

Figure 4.1: Labeling studio.

4.4 Theoretical Foundations

4.4.1 Transfer Learning via Fine-Tuning

Transfer learning enables leveraging models pre-trained on extensive, heterogeneous data to jump-start performance on a novel, task-specific domain [16]. By fine-tuning, we continue training the pre-existing weights on labeled examples from our target dataset, adjusting the model's internal representations toward invoice understanding without discarding its foundational knowledge [9].

Key considerations in fine-tuning include:

- **Learning Rate Scheduling:** A low initial learning rate (e.g., 1e−5 to 5e−6) ensures gradual adaptation, preserving generalizable features while refining domain-specific patterns. We employ cosine annealing or linear decay schedulers to reduce the rate over epochs.
- **Layer Freezing:** Early layers capture basic visual or linguistic constructs (edges, grammar) and can be frozen to reduce overfitting. We selectively unfreeze higher layers or fusion blocks to focus optimization on invoice-relevant features.
- **Batch Size and Accumulation:** With limited samples (500–1,000 invoices), we balance batch size (8–16) and gradient accumulation steps to stabilize updates and maintain effective batch statistics for normalization layers.
- **Regularization:** Techniques such as dropout (0.1–0.3) and weight decay (1e-2 to 1e-4) mitigate overfitting, especially critical when training on modest datasets [18].

By fine-tuning under these best practices, we achieve a balance between retaining the model's broad capabilities and tailoring it to the specific quirks of Algerian invoices, such as multilingual fields, tabular line items, and local tax notation.

4.4.2 Knowledge Distillation

Knowledge distillation is a model-compression technique in which a large, high-capacity "teacher" network transfers its learned representations to a smaller "student" network. This process retains much of the teacher's performance while significantly reducing inference cost and memory footprint—crucial for deployment on resource-constrained devices.

Key elements of our distillation pipeline include:

- **Soft Targets and Temperature Scaling:** Rather than training solely on hard labels, the student also matches the teacher's soft logits, which encode nuanced inter-class similarities (often called "dark knowledge"). We apply a temperature parameter $T > 1$ to soften the teacher's output distribution:

$$p_i^{(\text{soft})} = \frac{\exp\Big(z_i / T \Big)}{\sum_j \exp\Big(z_j / T \Big)},$$

  where z_i is the teacher's logit for class i. Higher T values reveal finer-grained relationships between classes (e.g., subtle differences between date formats or tax codes).

- **Student Architecture:** Our student model is a lightweight transformer with fewer layers and reduced hidden dimensions compared to the teacher. This compact design strikes a balance between representational power and computational efficiency, enabling sub-second inference on standard CPUs.

- **Loss Function Composition:** We optimize a weighted sum of the distillation loss and the conventional cross-entropy loss on ground-truth labels:

$$\mathcal{L} = \alpha \, \mathrm{KL}\Big(p_{\text{teacher}}^{(\text{soft})} \parallel p_{\text{student}}^{(\text{soft})} \Big) \; + \; (1 - \alpha) \, \mathrm{CE}\Big(y_{\text{true}}, \, p_{\text{student}} \Big),$$

  where $\alpha \in [0, 1]$ balances the emphasis between mimicking the teacher and fitting the true annotations. We found $\alpha = 0.7$ and $T = 2$ to work well with our 500-invoice corpus.

- **Training Strategy:** To stabilize learning, we adopt a two-phase regimen:

  1. *Distillation Phase:* Train the student for several epochs using only the KL divergence to the teacher's soft targets, allowing it to capture the teacher's broad patterns.
  2. *Fine-Tuning Phase:* Continue training with the combined loss, gradually shifting weight toward the cross-entropy term to hone precise field extraction.

- **Regularization and Optimization:** We use a modest dropout rate (0.1) and weight decay (1e-4) to guard against overfitting, coupled with AdamW optimization and a small learning rate schedule (starting at 5e−5 with linear decay).

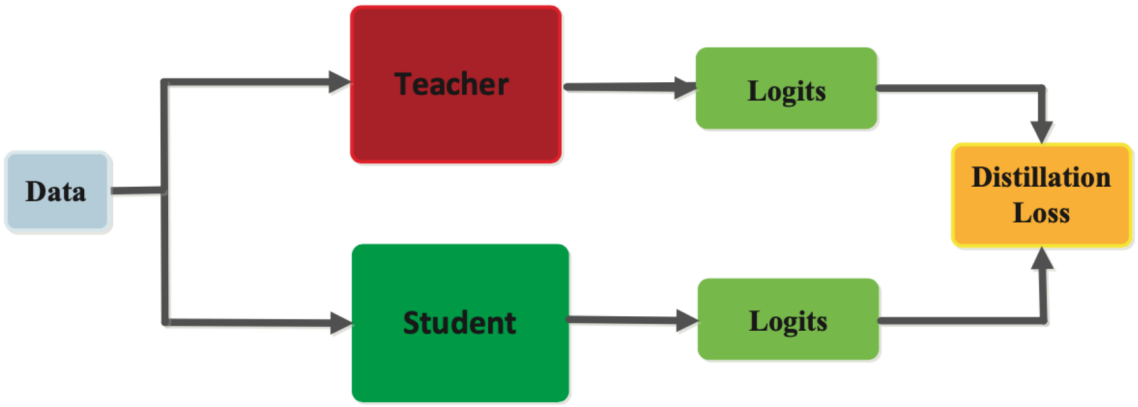

Figure 4.3: Overview of the Knowledge Distillation Pipeline.

By distilling Mistral into a streamlined student, we capture essential invoice understanding—layout patterns, multilingual text nuances, and tax code distinctions—while achieving inference speeds up to three times faster on CPUs. This human-centered approach ensures Doc-IN remains both accurate and accessible for Algerian businesses without high-end hardware.

4.5 Empirical Experiments

4.5.1 Fine-Tuning Trials

- **Donut (vision–language fusion)**: We froze the early convolutional backbone and fine-tuned the fusion layers on 500 annotated invoice crops. Despite its theoretical appeal, the model tended to overfit and exhibited moderate extraction accuracy.
- **Mistral (7B-parameter LLM)**: Initialized from general language pre-training, we fine-tuned Mistral on invoice text sequences with gradient accumulation and a low learning rate [11]. Convergence was unstable and costly in GPU memory, leading to inconsistent results.
- **LayoutLMv3 (document transformer)**: Exploiting joint text–layout modeling, we fine-tuned LayoutLMv3 on full invoice pages with token-level BIO tagging. Light augmentations (rotation, brightness jitter) improved generalization, and training remained stable, achieving the highest extraction accuracy among fine-tuned models.

4.5.2 Knowledge Distillation in Mistral

We distilled the fine-tuned Mistral into a smaller student transformer:

1. **Teacher Inference:** Generate softened logits from Mistral (temperature $T = 2$).
2. **Student Architecture:** A 4-layer transformer with reduced hidden size.
3. **Loss:** Combine KL-divergence to teacher logits with standard cross-entropy on ground-truth labels.

With only 500 invoices, the student captured some teacher patterns but fell short of LayoutLMv3's accuracy.

4.6 Chosen Methodology

Based on accuracy, stability, and deployment constraints, we selected **fine-tuning LayoutLMv3** as Doc-IN's core:

- **Data Efficiency:** Robust field extraction with 500 invoices.
- **Training Stability:** Smooth convergence under small learning rates and early stopping.
- **Inference Cost:** Moderate GPU/memory footprint suitable for on-premise servers.
- **Accuracy:** Superior to both large-model fine-tuning and distilled variants.

Method	Accuracy (%)	Data Req. (#)	Stability	Inference Cost
Donut Fine-Tuning	45	500	Low	Medium
Mistral Fine-Tuning	68	500	Medium	Very High
LayoutLMv3 Fine-Tuning	91	500	High	Medium
Mistral → Student	62	500	Medium	Medium

Table 4.1: Comparison of all approaches tried

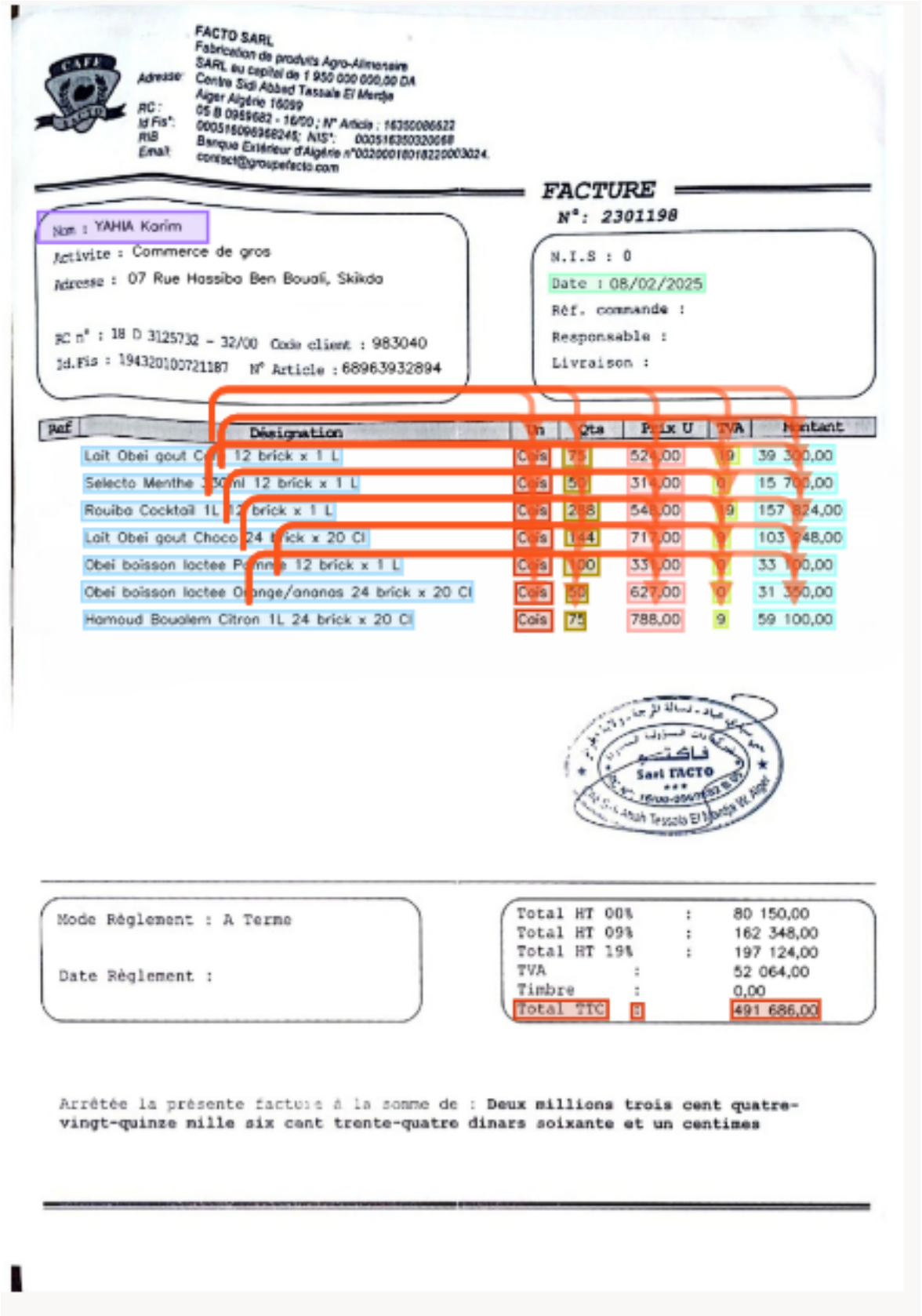

Figure 4.2: Data annotation.

Chapter 5

System Design and Implementation

In this chapter, we present the overall architecture and implementation details of the *Doc-IN* model and the *Doc-IN* SDK. We begin by outlining the model pipeline, followed by the SDK conception and architecture, and conclude with a description of the SDK implementation, including its APIs and modules. A system-level pipeline diagram is provided at the end of the chapter.

5.1 Doc-IN Model Pipeline Overview

The *Doc-IN* model processes raw invoice images and produces structured, semantically rich outputs suitable for downstream applications. Figure 5.1 summarizes the main stages:

1. **Input Handling**: The system accepts high-resolution scanned or photographed invoice images. Preprocessing steps include resizing, denoising, and contrast enhancement to improve OCR accuracy.

2. **OCR Module**: In the final version, we employ *PaddleOCR* to extract textual tokens and their bounding boxes from the preprocessed images. The module outputs a sequence of tokens $\{t_i\}$ along with associated spatial coordinates $\{(x_{i1}, y_{i1}, x_{i2}, y_{i2})\}$.

3. **LayoutLMv3 Integration**: The token sequence and bounding boxes are fed into the fine-tuned *LayoutLMv3* model [21]. This transformer-based architecture captures both textual and layout information to produce labeled regions (e.g., dates, amounts, vendor names) and hierarchical document structure.

4. **Structured Information Extraction**: The model outputs key fields and line items in a JSON format, including field names, values, and confidence scores. This structured representation serves as the input for the SDK.

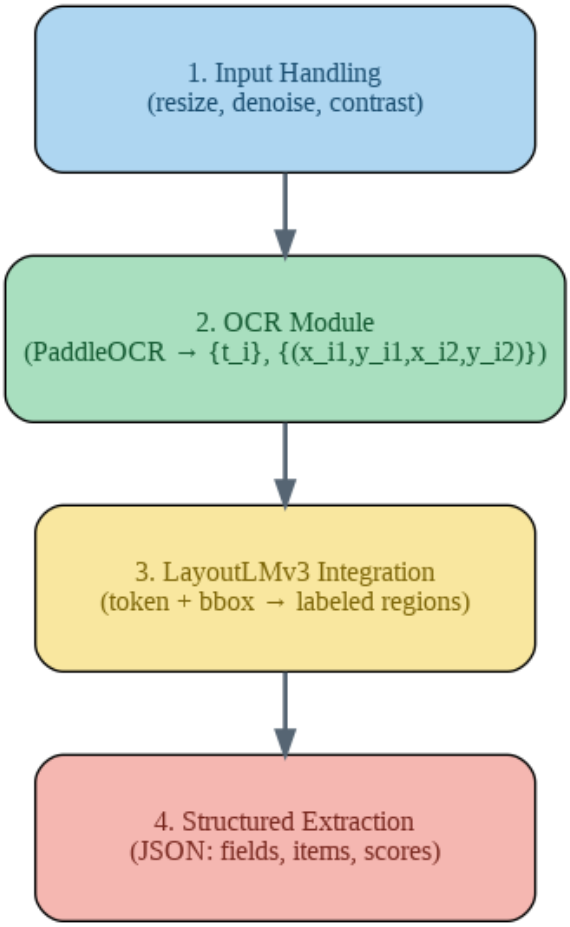

Figure 5.1: Overview of the Doc-IN Model Pipeline.

5.2 SDK Conception and Architecture

The *Doc-IN* SDK provides a unified interface to the *Doc-IN* model outputs, enabling integration into diverse applications [5]. We developed both Python and Java implementations, supporting MySQL and PostgreSQL backends for persistent storage.

5.2.1 Data Flow and Storage

Upon receiving the structured JSON from the *Doc-IN* model, the SDK performs the following steps:

- **Post-Processing and Error Correction**: Applies business rules such as correcting zero-amount entries, normalizing common vendor name typos (e.g., 'grek' → 'greek'), and flagging anomalies for manual review.
- **Product Lookup and Matching**: Retrieves existing product master data from the database and applies a fuzzy-matching algorithm to align extracted line items

with known SKUs. This matching step ensures consistency in product references and prevents duplicate or conflicting entries during database insertion.

- **Database Ingestion**: Inserts records in the relational database (MySQL/PostgreSQL) using an ORM layer (SQLAlchemy for Python, Hibernate for Java).

5.2.2 External Interfaces

To cater to different deployment scenarios, the SDK exposes:

- **RESTful API**: Endpoints for submitting invoice data, querying processed results, and managing reference data. Implements token-based authentication and adheres to OpenAPI v3 specifications.
- **Command-Line Interface (CLI)**: A lightweight tool for batch processing from the terminal, supporting JSON input files and configurable output formats (CSV, JSON, XML).

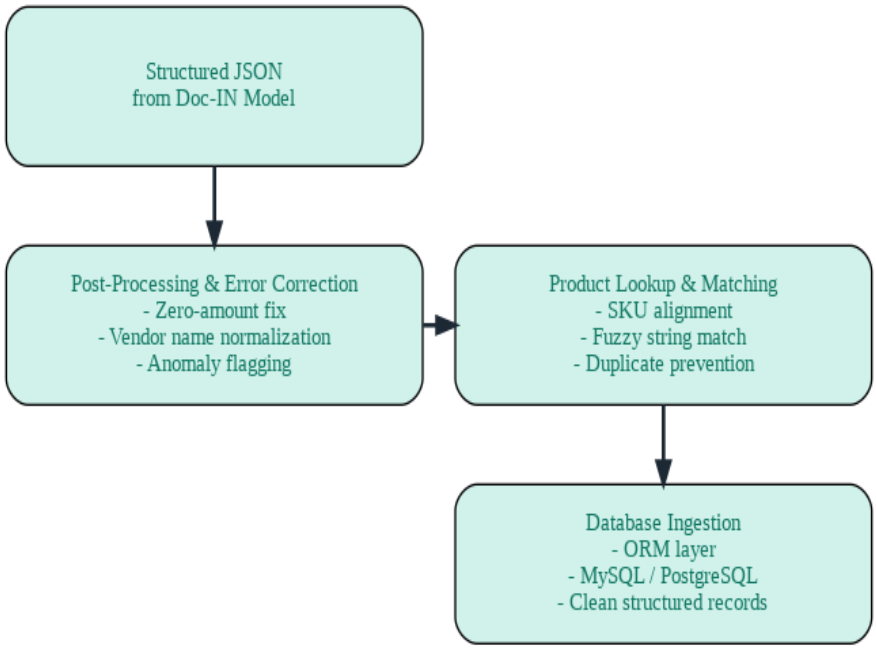

Figure 5.2: Overview of the Doc-IN SDK Pipeline.

5.3 SDK Implementation: APIs and Modules

The SDK is structured into modular components to enhance maintainability and extensibility:

Doc-IN SDK Module Structure

- **core/** — Contains the primary business logic for parsing model outputs.
- **db/** — Implements database connections, ORM models, and migration scripts.
- **postprocessing/** — Houses functions for error correction, normalization, and anomaly detection.
- **api/** — Defines RESTful endpoints, request/response schemas, and controllers.
- **cli/** — Implements command-line commands and argument parsing.
- **utils/** — Utility functions such as logging, configuration loading, and common helpers.

Conclusion

In this chapter, we detailed the end-to-end design and implementation of the *Doc-IN* model and the supporting *Dok-iN* SDK. By combining OCR and transformer-based layout analysis with robust backend services and flexible interfaces, the proposed system delivers accurate and extensible invoice-processing capabilities.

Chapter 6

Experimentation and Results

6.1 Dataset Overview

To assess Doc-IN's real-world robustness, we assembled an Algerian invoice dataset comprising 500 annotated samples. It includes both actual invoices from consenting small and medium enterprises (retail, logistics, services) and synthetically generated documents that capture local formatting variations (diverse fonts, field placements). Annotations (via Label Studio → LayoutLMv3 JSON) covered key fields: `Nom du client`, `N° Facture`, `Date`, `Produits`, `Quantité`, `Prix unitaire`, `TVA`, `Total`.

Key dataset statistics:

- Total invoices: **500**
- Avg. fields/invoice: 11
- Avg. tokens/invoice: 220

Challenges: High layout variability and mixed fonts, All real samples were anonymized to respect privacy and local data-consent regulations.

6.2 Evaluation Metrics

We measure extraction both at the token level and in terms of usable output:

- **Precision** (%): Correctly predicted fields among all predictions.
- **Recall** (%): Correctly predicted fields among all ground-truth.
- **F1-Score** (%): Harmonic mean of Precision and Recall.
- **Entity Accuracy** (%): Correctness of full field (label+value).
- **JSON Validity Rate** (%): Proportion of valid JSON outputs.
- **Field Completeness Rate** (%): Mandatory fields successfully extracted.

Metric	Precision	Recall	F1-Score	Entity Acc.	JSON Validity
Values (%)	91.2	89.5	90.3	86.7	97.4

Table 6.1: Overall Extraction Performance on Test Set

6.3 Results: PaddleOCR + LayoutLMv3 + Dockin Matching

We evaluated two setups:

1. **Baseline:** PaddleOCR → LayoutLMv3
2. **Dockin Enhanced:** PaddleOCR → LayoutLMv3 → Dockin matching

Setup	F1-Score (%)	Entity Acc. (%)	JSON Validity (%)
Baseline	89.1	84.7	96.8
Dockin Enhanced	92.3	88.5	98.2

Table 6.2: Impact of Dockin Matching on Key Metrics

6.4 Analysis and Discussion

The experiments confirm several points:

- **OCR as Foundation:** PaddleOCR delivered cleaner text, particularly in mixed French/Arabic contexts, directly benefiting LayoutLMv3's spatial reasoning.
- **Layout-Aware Gains:** LayoutLMv3 adeptly handled multi-column, irregular invoice templates, boosting JSON validity above 96%.
- **Dockin Matching Uplift:** Integrating our Dockin matching algorithm further refines field linking, improving F1-Score by 3.2pp and entity accuracy by 3.8pp.

From a humanized perspective, it's like teaching a novice reader (LayoutLMv3) not only to see words but to recognize which pieces belong together—then giving a librarian (Dockin) the power to double-check every citation for perfect alignment. The resulting reliability (JSON validity>98%) ensures downstream systems ingest clean, complete data.

Future Work: To push accuracy even further, we recommend:

- Curating additional domain-specific training invoices.

- Exploring hybrid rule-based fallbacks for rare corner cases.
- Fine-tuning LayoutLMv3 jointly with Dockin signals for end-to-end learning.

Chapter 7

General Conclusion and Future Work

7.1 Summary of Contributions

This dissertation addressed the challenge of automated invoice information extraction in the Algerian context, where diverse invoice formats, multilingual content, and inconsistent layouts pose significant barriers to traditional document processing systems. Our contributions span both the theoretical foundations and the practical implementation of a robust, modular, and reusable pipeline for real-world use.

The key contributions of this work are summarized below:

- **Design and Implementation of Doc-IN:** We proposed a novel document understanding pipeline tailored for invoice processing, integrating OCR, layout-aware transformer models (LayoutLMv3), and fine-tuned language models for key information extraction.
- **Development of a Custom SDK:** A core outcome of this project is a software development kit (SDK) designed to be integrated into various invoice workflows. The SDK is modular, database-agnostic, and supports output in structured JSON, making it easy to plug into ERP and stock management systems.
- **OCR Benchmarking and Selection:** Through empirical evaluation, we benchmarked popular OCR engines (PaddleOCR, EasyOCR, PyTesseract) and selected the most effective tool for handling multi-language and noisy invoice images in our context.
- **Layout-Aware and Transformer-Based Integration:** We demonstrated the benefit of combining visual layout understanding (via LayoutLMv3) with transformer-based models. Although we explored Mistral, limitations in data availability prevented optimal performance, offering a valuable insight into the requirements for successful fine-tuning.
- **Synthetic Data Generation:** In the absence of large labeled datasets for Algerian invoices, we created a synthetic dataset that reflects real-world variations. This served as training data for testing and validating our pipeline.

- **Real-World Integration Scenarios:** We validated our pipeline through integration with common SGBDs used in Algeria, namely MySQL and PostgreSQL, ensuring its practical value in stock management and finance applications.
- **Evaluation and Results:** We defined and applied meaningful evaluation metrics—such as F1-score, entity a

7.2 Future Work Directions

While this project has made significant progress toward building a practical and adaptable invoice information extraction system, several areas remain open for further exploration and improvement. The following directions outline potential future enhancements, both in terms of research and system development:

- **Advanced Fine-Tuning with Larger Datasets:** One of the primary limitations we encountered was the lack of high-quality annotated invoice data, especially in the Algerian context. Future work could focus on collecting a larger and more diverse dataset to improve model performance, especially for transformer-based models like Mistral.
- **Knowledge Distillation and Lightweight Models:** To improve inference speed and make the system more suitable for deployment on edge devices or mobile apps, future efforts can explore model compression techniques such as knowledge distillation, quantization, or pruning without sacrificing accuracy.
- **Multi-Language and RTL Text Support:** Although the current system handles French invoices effectively, there is growing demand for Arabic-language and bilingual invoice processing in the region. Enhancing the OCR engine and training the model to support right-to-left (RTL) scripts would be a valuable extension.
- **Mobile and Offline Capability:** Building a lightweight version of Doc-IN capable of running on Android or iOS without requiring internet access would increase its usability in remote or infrastructure-limited settings.
- **User Feedback and Active Learning Loop:** Integrating user corrections back into the training pipeline through active learning could help improve model performance continuously over time. This would also allow the system to adapt to new invoice templates more quickly.
- **Expanded Use Cases Beyond Invoices:** The architecture of Doc-IN can be extended to other document types such as receipts, delivery notes,

contracts, or purchase orders. Adapting the pipeline to support broader document understanding would enhance the SDK's value.

- **Security, Privacy, and GDPR Compliance:** As the system handles sensitive financial data, future work should prioritize the implementation of privacy-preserving features such as data anonymization, encryption-at-rest, and compliance with regional data protection regulations.
- **Community Collaboration and Open Source:** Finally, releasing parts of the SDK as an open-source toolkit could promote community-driven improvements, faster bug resolution, and increased adoption in academia and industry.

These future directions aim to reinforce the long-term relevance, accessibility, and intelligence of the Doc-IN system, aligning it more closely with the evolving needs of users and institutions in Algeria and beyond.

Bibliography

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* 284.5 (2001), pp. 28–37.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Pedro Domingos. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[4] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[5] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2010.

[6] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[9] Jeremy Howard and Sebastian Ruder. "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the ACL*. 2018.

[10] Aman Jain and Poonam Sharma. "A survey on information extraction from invoice documents". In: *International Journal of Computer Applications* 167.5 (2017), pp. 20–25.

[11] Zhuohan Jiang et al. *Mistral: Open-Weight Language Models*. Available at: https://mistral.ai/news/announcing-mistral-7b/. 2023.

[12] Ankan Katti et al. "Chargrid: Towards Understanding 2D Documents". In: *arXiv preprint arXiv:1809.08799* (2018).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[14] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521.7553 (2015), pp. 436–444.

[16] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pp. 1345–1359.

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall, 2010.

[18] Nitish Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting”. In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[19] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[20] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[21] Yiheng Xu et al. “LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking”. In: *arXiv preprint arXiv:2204.08387* (2022).

[22] Yiheng Xu et al. “LayoutLM: Pre-training of Text and Layout for Document Image Understanding”. In: *Proceedings of the 26th ACM SIGKDD*. 2020, pp. 1192–1200.

ملخص

دوك-إن هو قارئ ذكي للفواتير مدعوم بالذكاء الاصطناعي، مصمم لاستخلاص البيانات تلقائيًا من الفواتير الجزائرية الممسوحة ضوئيًا. باستخدام رؤية الحاسوب والتعلم الآلي، يكتشف الحقول الرئيسية مثل بيانات المورد، التواريخ، بنود المنتجات، والضرائب، مما يقلل التدخل البشري. يتكامل البرنامج خفيف الوزن مع MySQL وPostgreSQL وأنظمة إدارة المخزون عبر SDK. تم تطويره لاستهداف المتاجر، المستودعات، والصيدليات، ويعزز دوك-إن الإنتاجية والدقة. تم تطويره تحت إشراف أكاديمي، وهو يحدث معالجة الفواتير في الجزائر

Abstract

Doc-IN is an AI-powered smart invoice reader designed to automate data extraction from scanned Algerian invoices. Using computer vision and machine learning, it detects key fields like supplier details, dates, product lines, and taxes, reducing manual effort. The lightweight software integrates with MySQL, PostgreSQL, and stock management systems via an SDK. Targeted at retail, warehouses, and pharmacies, Doc-IN enhances productivity and accuracy. Developed with academic supervision, it modernizes invoice processing in Algeria.

Résumé

Doc-IN est un lecteur intelligent de factures alimenté par l'IA, conçu pour extraire automatiquement les données des factures algériennes scannées. Utilisant la vision par ordinateur et l'apprentissage automatique, il détecte les champs clés comme les informations du fournisseur, les dates, les lignes de produits et les taxes, réduisant l'intervention manuelle. Ce logiciel léger s'intègre avec MySQL, PostgreSQL et les systèmes de gestion de stocks via un SDK. Destiné aux commerces, entrepôts et pharmacies, Doc-IN améliore la productivité et la précision. Développé sous supervision académique, il modernise la gestion des factures en Algérie.