Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Institute of Computer Science

Bachelor's diploma thesis

in the field of study Computer Science
and specialisation Computer Systems and Networks

Invoice Field Extraction with Machine Learning and Transformers

Ulad Shuhayeu
student record book number 323903

thesis supervisor
mgr inż. Rajmund Kożuszek

WARSAW 2025

Invoice Field Extraction with Machine Learning and Transformers

Abstract. Financial workflows face a major obstacle in automated invoice key field extraction because of the wide range of invoice layouts and the presence of OCR errors. The thesis investigates how traditional feature-engineered systems compare to contemporary multimodal transformer models when extracting invoice fields from the FATURA benchmark (10,000 invoices, 50 templates). The evaluation process uses a single OCR-based system for all tests: EasyOCR generates tokens and bounding boxes, and a lightweight layout detector produces optional region cues which are used to classify tokens through either (i) traditional models that combine TF–IDF text features with normalized spatial coordinates (Linear/Kernel SVM, Logistic Regression, Random Forest, XGBoost) with class-imbalance mitigation (inverse-frequency weights, SMOTE), or (ii) layout-aware transformers such as LayoutLMv2.

Two evaluation protocols quantify generalization: *intra-template* (seen layouts) and *inter-template* (unseen layouts). The traditional models demonstrate exceptional performance on familiar layout types because Linear SVM with text+bbox features achieves 100.0% accuracy. The addition of spatial features to text-only models leads to significant performance improvements when dealing with new layouts, with Linear SVM reaching 92.3% accuracy and XGBoost achieving 96.15%. The two multimodal transformer models demonstrate near-perfect generalization performance: LayoutLMv2 achieves 99.98% accuracy (macro-F1 $\approx$ 1.00) and LiLT reaches 98.7%. Ablations show that spatial information and class distribution management play vital roles in traditional models, yet transformers achieve better results through their ability to process visual data and their pretraining process. The research evaluates system performance, deployment requirements, and dataset biases, concluding that template-specific solutions work best for established formats, yet multimodal transformers are essential for robust cross-template generalization in production settings.

Keywords: Invoice field extraction, Document AI, Machine learning, Transformers, FATURA dataset

Identyfikacja pól faktur z wykorzystaniem uczenia maszynowego i transformerów

Streszczenie. W potokach przetwarzania dokumentów finansowych występuje poważna przeszkoda w automatycznej identyfikacji kluczowych pól faktur wynikająca z dużej różnorodności układów faktur i występowania błędów OCR.

W pracy bada się, jak tradycyjne systemy uczeni maszynowego wypadają w porównaniu ze współczesnymi multimodalnymi modelami transformerów przy identyfikacji pól faktur ze zbioru FATURA (10 000 faktur, 50 szablonów).

Proces ewaluacji wykorzystuje jeden zestaw danych z OCR dla wszystkich testów: EasyOCR generuje tokeny i pola ograniczające, a lekki detektor układu generuje opcjonalne wskazówki dotyczące regionów.

Te cech są wykorzystywane do klasyfikacji tokenów poprzez (i) tradycyjne modele łączące cechy tekstowe TF–IDF ze znormalizowanymi współrzędnymi przestrzennymi (maszyna wektorów nośnych liniowa lub z jądrem nieliniowym, regresja logistyczna, las losowy, XGBoost) z łagodzeniem nierównowagi klas (wagi odwrotnej częstotliwości, SMOTE) lub (ii) transformery uwzględniające układ, takie jak LayoutLMv2.

Przeprowadzono badanie ilościowe dla znanych szablonów faktut (protokół intra-template) oraz dla nieznanych szablonów faktur (protokół inter-template). Tradycyjne modele wykazują wyjątkową wydajność w przypadku znanych typów układów faktur: liniowy SVM z cechami tekstowymi i polami osiąga 100,0% dokładności. Dodanie cech przestrzennych do modeli tekstowych prowadzi do znacznej poprawy wydajności w przypadku nieznanych układów, przy czym liniowy SVM osiąga 92,3% dokładności, a XGBoost 96,15%. Multimodalny transformer demonstrują niemal idealną wydajność generalizacji: LayoutLMv2 osiąga 99,98% dokładności (makro-F1 $\approx$ 1,00).

Szczegółowa analiza pokazuje, że informacje przestrzenne i uwzględnienie rozkładu klas odgrywają kluczową rolę w modelach tradycyjnych. Transformery osiągają lepsze wyniki dzięki zdolności przetwarzania danych wizualnych i uczeniu wstępnemu na bardzo dużym zbiorze danych.

W eksperymentach oceniono jakość identyfikacji pól, wymagania wdrożeniowe i błędy w zbiorach danych, stwierdzając, że rozwiązania tradycyjne najlepiej sprawdzają się w przypadku ustalonych formatów faktur, natomiast rozwiązania korzystające z multimodalnych transformerów są lepsze dla nowych, nieznanych układów faktur.

Słowa kluczowe: identyfikacja pól faktur, analiza dokumentów, uczenie maszynowe, transformery, zbiór danych FATURA

Contents

1. Introduction

1.1. Background and Context

Business organizations use invoices as key financial records to document their transactions which include crucial details about invoice numbers and dates and vendor and client information and total payment amounts. The process of automatic field extraction from invoices becomes vital because it helps organizations avoid manual data entry work and decreases human mistakes while enhancing operational efficiency. The extraction of invoice fields through automation faces challenges because invoices exist as semi-structured documents which use different templates and contain various elements including tables and logos and multiple formatting styles. The implementation of template-matching solutions and Optical Character Recognition (OCR) systems demands manual setup for each layout design but this approach fails to handle numerous different invoice formats effectively. The wide range of invoice presentation styles together with scanning artifacts and handwritten content and stamp marks makes it impossible to develop universal extraction rules for field identification. The task has gained substantial research interest in document analysis because it represents a specific case of extracting important information from visually complex documents [1].

Document AI research has made recent progress through machine learning techniques which enhance document information extraction capabilities. The initial methods for document information extraction relied on rule-based heuristics together with classical Natural Language Processing techniques after performing OCR text extraction [2]. Multiple studies have demonstrated that machine learning classifiers operating on invoice and receipt data through textual and layout features can identify field values [1]. The extraction of invoice fields through classical methods involves converting OCR output words into features that include Term Frequency Inverse Document Frequency (TF–IDF) scores and page positions before using Support Vector Machines (SVM) or Random Forests for classification [2]. The approaches demonstrate excellent precision rates when working with established domains but they fail to adapt well to new document structures or datasets. Modern deep learning models achieve better generalization through their ability to learn from extensive datasets of labeled documents. The LayoutLM series [1], [3] and the Language-Independent Layout Transformer (LiLT) [4] represent two examples of multi-modal Transformers which achieve top results in document understanding tasks across multiple languages and document layouts. The transformer models learn document features from text and images and spatial information which allows them to understand complex invoice documents better than traditional feature-engineered approaches [5]. The success of deep models depends heavily on large-scale annotated datasets and the FATURA dataset introduced 10,000 multi-layout invoice images with field annotations which enables strong model training and evaluation [6].

1.2. Research Motivation

The research basis emerges from industrial requirements together with recent technological breakthroughs. Organizations continue to use manual data entry and inflexible template-based software for invoice processing which produces both slow operations and frequent mistakes. The current market requires automated solutions which can process invoices from various sources while requiring minimal per-layout configuration. The document analysis field now offers transformer-based models trained on document corpora which show great potential to enhance extraction precision [3], [4]. The advanced models require significant computational resources and complex fine-tuning processes which create uncertainty about their practical value compared to basic methods. The research investigates how modern multi-modal transformers perform relative to traditional machine learning methods for invoice field extraction through a systematic evaluation of their performance under different conditions. The research evaluates both traditional and state-of-the-art methods through extensive testing on a wide range of data to identify their respective advantages and disadvantages. The research findings help practitioners determine whether they should implement basic document processing solutions or deploy sophisticated deep learning systems for their document handling needs.

1.3. Problem Statement and Objectives

This thesis creates an automated system which extracts essential fields from invoices that present in different unorganized formats. The system ingests scanned images and converts them into structured data. Because invoice formats are not standardized, it must handle diverse text placements, and multiple sections.

Objectives:

1. Design an end-to-end machine-learning pipeline for automatic invoice field extraction with minimal human input.
2. Compare a traditional approach—OCR, feature engineering, and conventional classifiers—with transformer-based deep-learning models, and assess their performance.

The FATURA dataset serves as the foundation for training and evaluation for both approaches. experiments span a range of invoice layouts and capture conditions [6].

Performance is assessed using quantitative metrics—precision, recall, and robustness to layout variation—to characterize each approach.

Finally, this work provides deployment guidance for invoice-extraction systems and identifies use cases where one approach is preferable to the other.

1.4. Structure of the Thesis

The remaining sections are organized as follows. The thesis begins by evaluating previous research about template-based and classical methods alongside layout-aware

transformers and OCR-free approaches to establish the position of this research within Document Analysis. The paper describes the datasets used in the study before explaining why FATURA was chosen as the main dataset and outlining the selection process and potential threats to validity. The system design and implementation are then presented, outlining the end-to-end pipeline and its components for both the classical and transformer branches. The experimental setup section follows the description of the experimental design by detailing intra- and inter-template evaluation protocols and data splits and input preparation and training configurations and evaluation metrics. This is followed by the results, which report quantitative comparisons under identical inputs at token and field levels. The research findings appear in the discussion section which also addresses operational barriers and methodological constraints of the study. The thesis concludes with its final section which shows research findings and suggests future directions for automated invoice processing system development.

2. Related Works

This section evaluates different methods for extracting essential information from visually complex business documents with a focus on invoices and receipts. The research literature can be categorized into three distinct categories: (i) template and classical machine learning systems processing OCR tokens with spatial information, (ii) layout-aware transformer models that unite text data with layout information and image content, and (iii) OCR-free end-to-end models. The assessment of these methods depends on public datasets which determine both the obtained results and the extent of generalization.

2.1. Templates and Classical Machine Learning

The first industrial systems used vendor-specific templates, which described field positions through rule-based and zone-based methods. These solutions provide high precision for established layouts but their ability to handle different suppliers is limited because each new or updated template needs manual engineering and validation. The research moved toward template-light processing systems which used OCR text segmentation to create tokens that received lexical and spatial feature representations before undergoing classification through SVM, logistic regression, random forest or boosted trees (Section 2.1.1 explains the choice of these models). The combination of TF–IDF or bag-of-words vectors with normalized bounding-box coordinates and basic layout indicators such as reading order, column index, and token type makes up representative feature sets [2]. The models operate efficiently during inference while maintaining data efficiency and transparency, but their performance weakens when encountering unfamiliar page structures because their decision boundaries depend on training data patterns [2]. The comparison between classical models and layout-aware transformers happens under both conditions where the layout is familiar and when it remains unknown.

2.1.1. Traditional Baseline Classifiers

We include SVM, Logistic Regression, Random Forest, and XGBoost as classical baselines because they are well-established for OCR-token classification with TF–IDF plus spatial features, are data-efficient to train, and provide transparent decision boundaries (linear models) or robust non-linear partitioning (tree ensembles). These models are widely used and maintained (e.g., scikit-learn and XGBoost libraries) and often deliver strong baseline accuracy on structured document tasks with modest training data and compute. For example, Sánchez-Perez et al. (2024) [7] report that a Support Vector Machine achieved the highest precision $\approx 91.86\%$ for key field extraction from invoices, closely followed by a Random Forest $\approx 91.58\%$.

Logistic regression is the simplest discriminative baseline in this setting: it is a convex, linear classifier with a single dominant hyperparameter (regularization strength), trains

very quickly on high-dimensional sparse TF–IDF features, and produces probabilistic outputs that are easy to interpret and calibrate [8], [9]. Its simplicity makes it a strong reference point for assessing the added value of more complex models.

XGBoost is widely regarded as one of the best off-the-shelf classifiers for tabular prediction tasks: as a regularized gradient-boosted tree ensemble, it captures non-linear feature interactions, handles heterogeneous feature types (sparse text vectors and numeric layout coordinates), is robust to feature scaling and missing values, and includes shrinkage and L1/L2 regularization for strong generalization [10]. In our setting, XGBoost can learn interactions between lexical cues (e.g., "total") and spatial context (e.g., bottom-right region) that linear decision boundaries may miss.

Random Forest provides a complementary tree-ensemble baseline based on bagging: it is stable, relatively insensitive to hyperparameters, and offers strong performance with low risk of overfitting on noisy OCR tokens, while yielding feature-importance diagnostics useful for analysis [8]. Together with **SVM** (a strong margin-based linear classifier for high-dimensional sparse data), these baselines offer a principled, lightweight counterpoint to layout-aware transformers, enabling a fair comparison under identical OCR inputs.

2.2. Layout-Aware Transformers

The LayoutLM family brought together text content analysis with two-dimensional layout processing to achieve top performance in form and receipt benchmark tests. The LayoutLM model extends BERT through learned bounding box representations of tokens, which enables the system to analyze text relationships with spatial positions [1]. The addition of a visual backbone and multi-modal pretraining objectives in LayoutLMv2 enables better extraction of key information from FUNSD and SROIE datasets [3]. The document understanding performance of LayoutLMv3 improves through its combination of masked text and masked image pretraining methods [11].

The LayoutLM family has inspired multiple researchers to develop new multimodal models. DocFormer uses multi-modal self-attention to merge text information with spatial data and visual content within a single transformer framework, achieving high performance on forms and receipts [5]. The layout encoding component of LiLT separates from language modeling so the layout encoder trained in one language becomes transferable to different languages when paired with a multilingual text encoder, which enhances cross-lingual performance [4].

Layout-aware transformers achieve better results than feature-engineered models on complex documents through their ability to process global information and two-dimensional layout, but they require more computational resources and training data. The transformers need to process the same OCR tokens and bounding boxes as classical models while using additional image features when available to achieve fair evaluation [3].

2.3. OCR-Free Models

The approach eliminates the requirement for explicit Optical Character Recognition (OCR) processing. The document analysis system Donut (Document Understanding Transformer) operates without OCR functionality despite its fun-sounding name. The system transforms page images into structured sequences (such as JSON field data) through direct vision-language transformer processing [12]. The system benefits from reduced error propagation because it omits text recognition steps while remaining language-independent and showing improved stability for poor-quality scanned documents. The models need extensive synthetic training data and large computational resources to learn character recognition automatically but their debugging process becomes more complicated because perception and extraction functions operate as a single unit. Most production systems maintain OCR-based pipelines because they match token-level supervision requirements even though OCR-free models show growing performance equality.

2.4. Synthesis and Relevance to This Study

Research demonstrates that token classifiers with text and spatial features perform well on familiar layouts but fail when encountering new layouts, whereas transformers that process text, layout, and appearance together achieve better robustness [2], [3]. The research uses an OCR-based pipeline to evaluate a TF–IDF+spatial classical model and LayoutLMv2 on FATURA data, while reporting results for both known and unknown page layouts.

3. Datasets

This section defines the criteria for dataset selection, reviews candidate datasets (including the one ultimately chosen), compares their properties, and justifies the final choice used in our experiments.

3.1. Selection Criteria

The pipeline in this work requires OCR tokens with bounding boxes and field/category labels to train both traditional token classifiers (TF–IDF + spatial features) and a layout-aware Transformer. Beyond annotation type, two experimental protocols are central: *intra-template* (seen layouts) and *inter-template* (unseen layouts) — see Section 5.1. A dataset with sufficient layout diversity and size is therefore essential to evaluate cross-template generalization. Specifically, we require: (i) invoice-domain data (not generic forms or receipts), (ii) token- and/or region-level annotations for key fields, (iii) layout diversity and scale to enable intra- and inter-template splits, and (iv) compatibility with both classical token classifiers and layout-aware transformers.

3.2. Datasets Considered

3.2.1. FUNSD (Form Understanding in Noisy Scanned Documents)

FUNSD provides 199 scanned, noisy forms with entity and relation annotations [13]. It is valuable for form understanding in degraded scans and supports entity linking, but it is not invoice-specific, is relatively small in size, and emphasizes form fields rather than invoice key values. For this study, FUNSD is not ideal because (i) the scale (199 pages) is insufficient to support robust inter-template evaluation, and (ii) the domain (generic forms across marketing, administrative, and scientific documents) does not match invoice structure and vocabulary [13]; see Figure 3.1a.

3.2.2. RVL–CDIP (Ryerson Vision Lab CDIP Subset)

RVL–CDIP offers 400,000 grayscale images across 16 document classes (e.g., letter, memo, form) with document-level labels for classification [14]. Although large and diverse, it lacks token- or field-level annotations and is designed for document *classification* rather than key information extraction. Using RVL–CDIP would require creating additional annotations or synthetic labels; moreover, the broad categories are weakly aligned with invoice-specific field extraction goals. Consequently, it does not meet the granularity requirements of this study; see Figure 3.1b.

3.2.3. SROIE (ICDAR 2019 Scanned Receipt OCR and IE)

SROIE contains 1,000 scanned *receipts* and poses three tasks: text localization (T1), OCR (T2), and key information extraction (T3) [15]. While widely used, SROIE focuses

on receipts (short, narrow layouts, limited field set) rather than invoices (richer layouts, item tables, multi-section structure). For token-level extraction with layout-aware models, SROIE offers a smaller field taxonomy and less layout complexity; it is therefore less suitable for evaluating cross-template generalization in invoices; see Figure 3.1c.

3.2.4. FATURA (Invoices, Multi-Layout IE)

FATURA contains 10,000 invoice images generated from 50 distinct templates, with annotations provided in multiple formats (including a Hugging Face/LayoutLM-compatible schema) and 24 classes relevant to invoices (e.g., *DATE*, *TOTAL*, *TABLE*) [6]. This dataset directly supports:

- **Invoice domain alignment:** annotations target invoice-specific fields, including line-item regions and totals [6].
- **Layout diversity at scale:** 50 templates enable explicit intra- vs. inter-template protocols to test cross-layout generalization [6].
- **Token/region compatibility:** JSON annotations and bounding boxes map cleanly to both classical token classifiers and LayoutLM-family inputs [6].

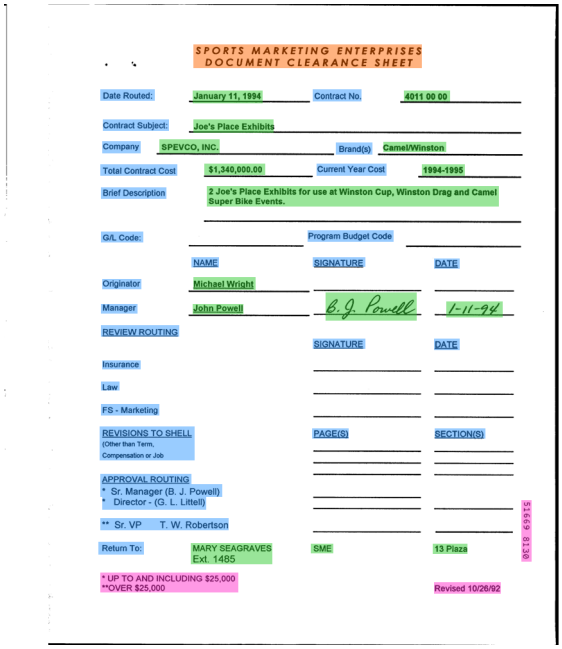
See Figure 3.1d for a representative page.

3.3. Comparison and Choice

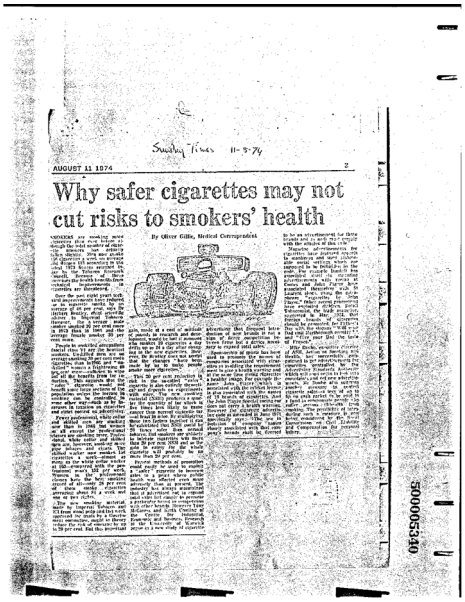
Table 3.1 summarizes the key differences relevant to this work. Based on the criteria in Section 3.1 and the properties in Section 3.2, FATURA best satisfies the requirements for invoice field extraction with both classical and transformer models and supports fair intra- and inter-template evaluation. We therefore select FATURA for all experiments.

3.4. Threats to Validity in Dataset Choices

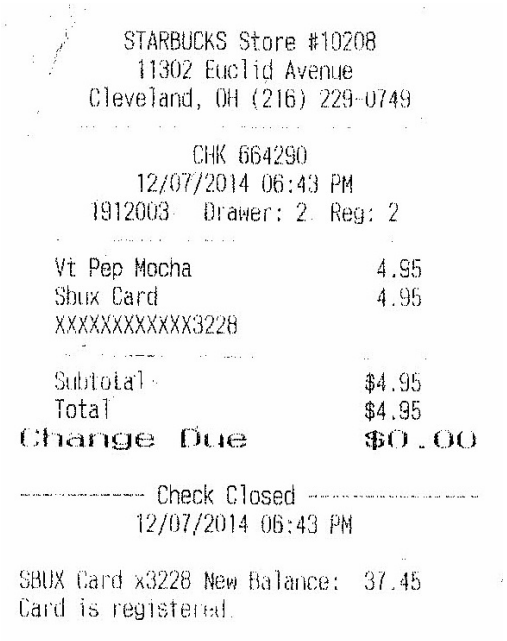
FATURA fulfills the research requirements but two limitations apply. First, the invoices stem from pre-defined templates and may not capture all artifacts of real-world scans (e.g., stamps, handwriting) [6]. Second, the unequal distribution of the 24 categories can bias learning; we mitigate this with inverse-frequency class weights and report macro-averaged metrics to represent minority classes.



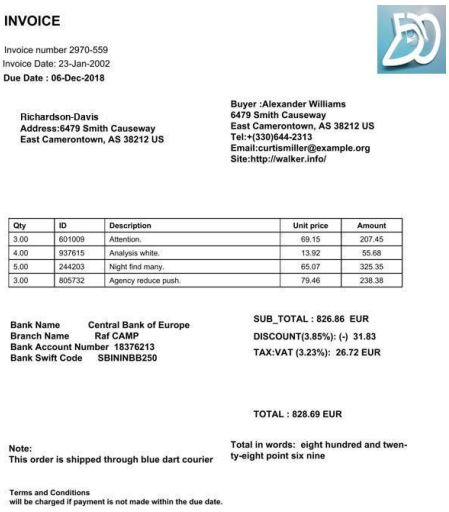
(a) FUNSD



(b) RVL–CDIP



(c) SROIE



(d) FATURA

Figure 3.1. Representative pages from each dataset (replace with your samples and add credits).

Table 3.1. Datasets considered for invoice field extraction.

Dataset	**Primary domain/-task**	**Annotations (granularity)**	**Scale**	**Fit for this study**
FUNSD [13]	Forms in noisy scans; entity linking	Entity/relationship on 199 forms	199 pages	Limited: small; not invoice-specific; focuses on forms
RVL–CDIP [14]	Document *classification* (16 classes)	Document-level labels only	400k images	Not suitable: lacks token/field labels for IE
SROIE [15]	Receipts OCR and IE	Tasks T1–T3; limited KIE fields	1k receipts	Partial: receipt-centric; smaller field set; less layout complexity for invoices
FATURA [6]	Invoices; multi-layout IE	Token/region boxes; 24 invoice classes; HF/COCO formats	10k images; 50 templates	Strong: invoice fields; intra/inter-template evaluation supported

4. System Design and Implementation

4.1. Overview

The system compares two extraction paradigms using identical OCR outputs (Figure 4.1). Pages are first processed by OCR to obtain token strings and bounding boxes. From this shared input, (i) a *classical ML branch* classifies tokens using TF–IDF and spatial features, and (ii) a *LayoutLMv3 branch* fine-tunes a pretrained multimodal transformer to predict token labels [11]. Both branches then aggregate labeled tokens into final field values. In addition, a QR detector attempts to decode machinereadable payloads that can assist postprocessing.

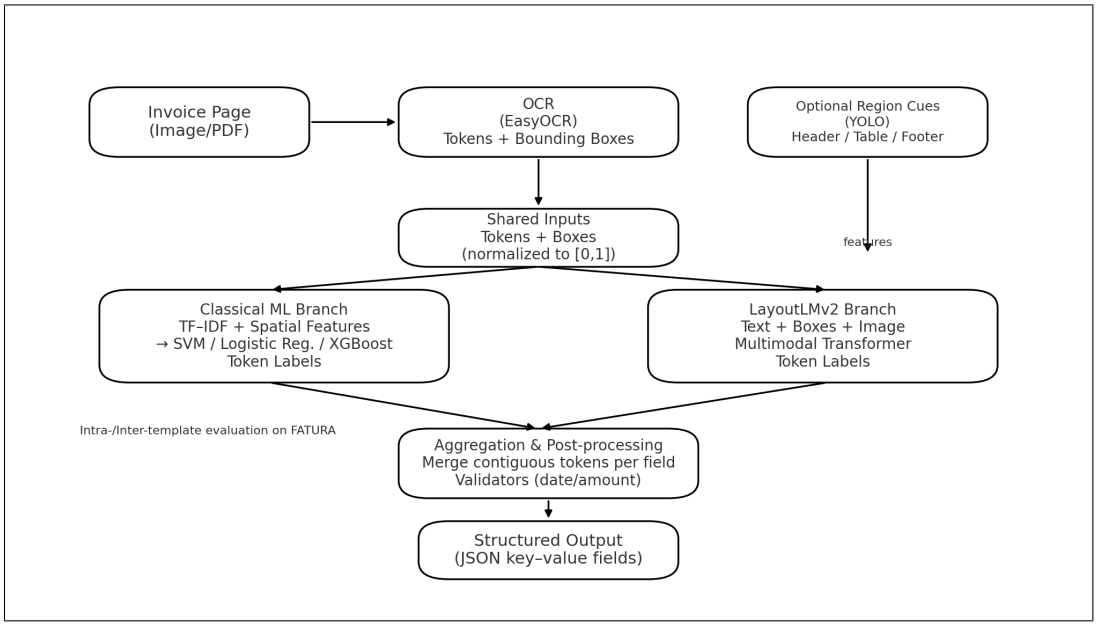


Figure 4.1. System overview. OCR provides tokens and bounding boxes that feed both branches; the transformer (LayoutLMv3) also consumes the page image. A QR decoding step provides an optional auxiliary signal for postprocessing.

4.2. Programming Language and Libraries

Language choice. I implement the system in Python (3.10+) because it offers a mature scientific stack, fast prototyping, and first-class support for both classical ML and deep learning.

Core libraries (with roles).

- **NumPy** (numpy.org) and **SciPy** (scipy.org) for vector/matrix operations and sparse algebra used by text features and classical models.
- **pandas** (pandas.pydata.org) for data wrangling (splits, labels, joins between OCR tokens and boxes).

- **scikit-learn** (scikit-learn.org) for TF–IDF vectorization and classical classifiers (multinomial Logistic Regression, linear SVM, Random Forest), plus metrics and calibration.
- **XGBoost** (xgboost.readthedocs.io) for gradient-boosted tree models on sparse/tabular features.
- **PyTorch** (pytorch.org) and **Hugging Face Transformers** (huggingface.co/docs/transformers) to fine-tune *LayoutLMv3* for token classification.
- **OpenCV** (opencv.org) for image I/O, geometry utilities, and QR decoding (see `QRCodeDetector`: docs.opencv.org).
- **EasyOCR** (github.com/JaidedAI/EasyOCR) as the OCR front-end to obtain token text and bounding polygons.
- **matplotlib** (matplotlib.org) for plots; **tqdm** (tqdm.github.io) for progress bars; **joblib** (joblib.readthedocs.io) for simple model caching.

Reasons to choose this stack

- *Python* unifies the pipeline: OCR pre-processing, feature engineering, model training, and evaluation live in one language with a consistent tooling story.
- *PyTorch* + *Transformers* provide a straightforward, dynamic-graph API and ready, well-tested implementations of LayoutLMv3. I fine-tune a pretrained checkpoint rather than training a transformer from scratch due to compute and data requirements.
- *scikit-learn* + *XGBoost* are the standard for strong, transparent baselines on token-level TF–IDF + spatial features: fast to train, easy to tune, and widely reproducible.
- *OpenCV* + *EasyOCR* keep the vision/OCR parts lightweight and scriptable, and OpenCV's built-in QR detector integrates cleanly with the post-processing rules.

Vectorization. I build TF–IDF with word unigrams, lowercasing, English stop-words, a capped vocabulary, and L2 normalization (fitted on the training split only). Spatial coordinates are normalized to $[0,1]$; continuous features are standardized where appropriate.

Reproducibility. I fix random seeds, save fitted vectorizers/scalers and model weights, and pin library versions in the repository GitHub link.

4.3. Classical Models: Mechanics, Training, and Code Sketches

This section describes the models trained on the token features from Section 4.5 with labels from Section 4.6, along with the training protocol. For each model, a concise description is followed immediately by a minimal code sketch.

4.3.1. Notation and Conventions

I use N for the number of training tokens, C for the number of classes, and d for the feature dimension. Each token has a feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and a label $y_i \in \{1,\dots,C\}$. Linear models use a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times C}$ whose cth column is $\mathbf{w}_c$, and a bias vector $\mathbf{b} \in \mathbb{R}^C$ with component b_c. I define the *logit* $z_{i,c} = \mathbf{w}_c^\top \mathbf{x}_i + b_c$ and the *softmax* $p_{i,c} = \exp(z_{i,c}) / \sum_{k=1}^{C} \exp(z_{i,k})$. For one-vs-rest SVM, the binary target for class c is $y_{i,c} \in \{-1,+1\}$.

In a decision-tree node with n samples, p_c denotes the class proportion n_c/n. For boosting I write $F^{(t-1)}(\mathbf{x})$ for the ensemble score before tree t, and define sample-wise derivatives $g_i = \partial\ell/\partial F$ and $h_i = \partial^2\ell/\partial F^2$ evaluated at $F^{(t-1)}(\mathbf{x}_i)$.

Table 4.1. Notation used in model definitions (quick legend).

Symbol	Meaning
N, C, d	#tokens, #classes, feature dimension
$\mathbf{x}_i$, y_i	features and label of token i
$\mathbf{W}$, $\mathbf{w}_c$, $\mathbf{b}$, b_c	linear weights and biases
$z_{i,c}$, $p_{i,c}$	logit and softmax prob. for class c
$y_{i,c}$	one-vs-rest target for class c (SVM)
p_c	class proportion in a tree node
$F^{(t-1)}$, g_i, h_i	ensemble score; first/second derivatives (boosting)

4.3.2. Logistic Regression (multinomial)

Intuition. A simple, interpretable baseline: each coefficient shows how a feature (e.g., a TF–IDF term or a spatial cue) pushes probability toward or away from a class. It trains fast and outputs calibrated probabilities.

Model. With logits $z_{i,c} = \mathbf{w}_c^\top \mathbf{x}_i + b_c$ and softmax $p_{i,c}$ as in Section 4.3.1, the regularized negative log-likelihood is

$$\mathscr{L}_{\mathrm{LR}}(\mathbf{W},\mathbf{b}) = -\sum_{i=1}^{N} \log p_{i,y_i} + \frac{\lambda}{2}\|\mathbf{W}\|_F^2,$$

where $\lambda \geq 0$ controls L_2 regularization. The objective is convex; we train multinomial logistic regression with the L-BFGS optimizer, which (under standard line-search conditions) converges to the global optimum. L-BFGS is a fast, low-memory quasi-Newton method that builds curvature from a short history of gradients (Newton-like steps from gradient-based curvature estimates; no Hessian needed.[1])

By-hand sketch (softmax LR).

```
import numpy as np

def softmax(z):                # z: (batch, C)
    z = z - z.max(axis=1, keepdims=True)  # stability
    e = np.exp(z)
    return e / e.sum(axis=1, keepdims=True)
```

[1] Overview: https://en.wikipedia.org/wiki/Quasi-Newton_method. For L-BFGS specifically, see Liu & Nocedal (1989): https://link.springer.com/article/10.1007/BF01589116.

```
def train_softmax_lr(X, y, C, lr=0.1, reg=1e-4, epochs=50, batch=256):
    # X: (N, d); y: (N,) in {0..C-1}
    N, d = X.shape
    W = np.zeros((d, C)); b = np.zeros((C,))
    for _ in range(epochs):
        for s in range(0, N, batch):
            e = min(s+batch, N)
            Xb, yb = X[s:e], y[s:e]
            Z  = Xb @ W + b
            P  = softmax(Z)
            Yh = np.zeros_like(P); Yh[np.arange(len(yb)), yb] = 1.0
            dZ = (P - Yh)
            gW = Xb.T @ dZ / len(yb) + reg * W
            gb = dZ.mean(axis=0)
            W -= lr * gW; b -= lr * gb
    return W, b
```

Library example (balanced multinomial LR).

```
from scipy.sparse import hstack, csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression

# Assume texts_train/val, boxes_train/val, y_train_raw/val_raw are prepared.
vec = TfidfVectorizer(lowercase=True, stop_words="english",
                      ngram_range=(1,1), max_features=50000)
X_text_tr = vec.fit_transform(texts_train)
X_text_va = vec.transform(texts_val)

scaler = MinMaxScaler()
X_sp_tr = scaler.fit_transform(np.asarray(boxes_train))
X_sp_va = scaler.transform(np.asarray(boxes_val))

X_tr = hstack([X_text_tr, csr_matrix(X_sp_tr)], format="csr")
X_va = hstack([X_text_va, csr_matrix(X_sp_va)], format="csr")

lr = LogisticRegression(multi_class="multinomial", solver="lbfgs",
                        max_iter=300, n_jobs=-1, C=1.0,
                        class_weight="balanced", random_state=42)
lr.fit(X_tr, y_train_raw)
```

4.3.3. Linear SVM (one-vs-rest)

Intuition. Maximizes the margin: it finds separating hyperplanes that keep the closest points (support vectors) as far as possible, which is effective for high-dimensional sparse TF–IDF features.

Model. For class c with binary targets $y_{i,c} \in \{-1, +1\}$, the hinge-loss primal is

$$\min_{\mathbf{w}_c, b_c} \tfrac{1}{2}\|\mathbf{w}_c\|_2^2 + C_{SVM} \sum_{i=1}^{N} \max\bigl(0, 1 - y_{i,c}(\mathbf{w}_c^\top \mathbf{x}_i + b_c)\bigr).$$

At test time, I predict the class with the largest signed score $\mathbf{w}_c^\top \mathbf{x} + b_c$.

By-hand sketch (one-vs-rest SGD).

```
def sgd_linear_svm_onevsrest(X, y, c, C_svm=1.0, lr=0.1, epochs=10):
    # y in {0..C-1}; train binary model for class c
    N, d = X.shape
    w = np.zeros(d); b = 0.0
    yc = np.where(y == c, 1.0, -1.0)
    for _ in range(epochs):
        idx = np.random.permutation(N)
        for i in idx:
            margin = yc[i] * (X[i] @ w + b)
            if margin < 1.0:
                w = w - lr * (w - C_svm * yc[i] * X[i])
                b = b + lr * (C_svm * yc[i])
            else:
                w = w - lr * w
    return w, b
```

4.3.4. Random Forest

Intuition. Robust, low-tuning non-linear baseline: many de-correlated trees vote, so variance drops and overfitting is controlled. Works well out-of-the-box on mixed lexical/spatial features.

Model. Each node chooses the split that maximizes impurity decrease; using Gini $G = 1 - \sum_c p_c^2$ (with p_c the class proportion in the node). Bagging (bootstrap sampling) and feature subsampling de-correlate trees; out-of-bag estimates provide internal validation and feature importance.

By-hand sketch (Gini + tiny forest skeleton).

```
def gini(counts):                       # counts: (C,)
    p = counts / counts.sum()
    return 1.0 - np.sum(p * p)

def best_split_feature_threshold(X, y, feature_index):
    x = X[:, feature_index]
    order = np.argsort(x)
    x, y = x[order], y[order]
    unique_thr = np.unique(x)
    best_gain, best_thr = -1.0, None
    total_counts = np.bincount(y, minlength=y.max()+1)
    G_parent = gini(total_counts)
    for t in unique_thr[:-1]:
        left = y[x <= t]; right = y[x > t]
        if len(left)==0 or len(right)==0: continue
        G_left   = gini(np.bincount(left,  minlength=total_counts.size))
        G_right  = gini(np.bincount(right, minlength=total_counts.size))
        gain = G_parent - (len(left)/len(y))*G_left - (len(right)/len(y))*G_right
        if gain > best_gain: best_gain, best_thr = gain, t
    return best_gain, best_thr

class Stump:
    def __init__(self, j=None, thr=None, left=None, right=None):
        self.j=j; self.thr=thr; self.left=left; self.right=right
    def predict(self, X):
        return np.where(X[:, self.j] <= self.thr, self.left, self.right)

def fit_decision_stump(X, y):
    d = X.shape[1]; best = (-1, None, None)
    for j in range(d):
        gain, thr = best_split_feature_threshold(X, y, j)
        if gain > best[0]: best = (gain, j, thr)
    left  = np.bincount(y[X[:, best[1]] <= best[2]]).argmax()
    right = np.bincount(y[X[:, best[1]] >  best[2]]).argmax()
    return Stump(j=best[1], thr=best[2], left=left, right=right)
```

4.3.5. XGBoost (gradient-boosted trees)

Intuition. A strong tabular/structured-data learner: trees are added sequentially to fix residual mistakes, with shrinkage and regularization to prevent overfitting. It captures interactions between lexical tokens and spatial cues that linear models miss.

Model. Let $F^{(t-1)}$ be the current ensemble score. Using a second-order (Newton) approximation of the loss ℓ around $F^{(t-1)}$, the t-th tree f_t minimizes

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} \left[g_i \, f_t(\mathbf{x}_i) + \tfrac{1}{2} h_i \, f_t(\mathbf{x}_i)^2 \right] + \Omega(f_t),$$

where $g_i = \partial \ell / \partial F$ and $h_i = \partial^2 \ell / \partial F^2$ (both evaluated at $F^{(t-1)}(\mathbf{x}_i)$), and $\Omega(f_t)$ penalizes leaf count/weights. Learning-rate shrinkage and column/row subsampling improve generalization on sparse TF–IDF + spatial features.

By-hand code sketch (boosting step).

```
import numpy as np

def _best_split_one_feature(x, g, h, lam=1.0, gamma=0.0):
    # Sort by feature; use cumulative sums for O(n) scan.
    order = np.argsort(x)
    xs = x[order]
    gs = g[order]
    hs = h[order]

    G_total = gs.sum()
    H_total = hs.sum()
    G_cum = np.cumsum(gs)
    H_cum = np.cumsum(hs)

    # Candidates are boundaries where the value changes.
    boundaries = np.nonzero(xs[:-1] != xs[1:])[0]
    if boundaries.size == 0:
        return None, -np.inf, 0.0, 0.0  # no valid split

    # Parent term is constant across thresholds.
    parent_term = (G_total**2) / (H_total + lam)

    best_gain = -np.inf
    best_thr = None
    best_GL = best_HL = 0.0

    for i in boundaries:
        GL = G_cum[i]
        HL = H_cum[i]
        GR = G_total - GL
        HR = H_total - HL

        gain = 0.5 * ((GL**2) / (HL + lam) + (GR**2) / (HR + lam) - parent_term) - gamma
```

```
        if gain > best_gain:
            best_gain = gain
            best_thr = 0.5 * (xs[i] + xs[i+1])  # midpoint threshold
            best_GL, best_HL = GL, HL

    return best_thr, best_gain, best_GL, best_HL

def one_boost_step_binary(X, y, F_prev, lam=1.0, gamma=0.0, eta=0.1):
    """
    One Newton-style boosting step for logistic loss.
    X: (N, d) features; y: (N,) in {0,1}; F_prev: (N,) current scores (log-odds).
    """
    # Gradients/Hessians for log loss.
    p = 1.0 / (1.0 + np.exp(-F_prev))
    g = p - y
    h = p * (1.0 - p)

    best_gain = -np.inf
    best_j = None
    best_thr = None
    best_GL = best_HL = 0.0

    # Find best stump across features.
    for j in range(X.shape[1]):
        thr, gain, GL, HL = _best_split_one_feature(X[:, j], g, h, lam, gamma)
        if gain > best_gain:
            best_gain = gain
            best_j = j
            best_thr = thr
            best_GL, best_HL = GL, HL

    G_total = g.sum()
    H_total = h.sum()

    F_new = F_prev.copy()

    if best_j is None:  # no split possible; single leaf update
        w = - G_total / (H_total + lam)
        F_new += eta * w
        return F_new

    # Compute leaf weights using the chosen split.
    left_mask = X[:, best_j] <= best_thr
    GL, HL = best_GL, best_HL
```

```
    GR, HR = G_total - GL, H_total - HL

    wL = - GL / (HL + lam)
    wR = - GR / (HR + lam)

    F_new[left_mask]  += eta * wL
    F_new[~left_mask] += eta * wR
    return F_new
```

4.3.6. Training protocol

TF–IDF vectorizers and scalers are fit on the training split only; continuous features are standardized. I choose hyperparameters by validation macro-F1: C for LR/SVM; number of trees and depth for Random Forest; number of boosting rounds, depth, learning rate, and subsampling for XGBoost. To address class imbalance I use inverse-frequency weights or per-sample weights in the loss/split criteria (see Section 4.7).

Artifacts and reproducibility. I persist the fitted TF–IDF vectorizer, scalers, label maps, and model weights; random seeds are fixed. These artifacts are re-used at inference for deterministic behavior.

4.4. OCR Front-End and QR Code Detection

OCR (EasyOCR) produces a sequence of tokens t_i with polygonal bounding boxes b_i. Coordinates are normalized to $[0,1]$ for scale invariance. In addition to text, I detect QR codes with OpenCV's `QRCodeDetector`, which returns the decoded payload (if any) and a polygonal bounding box. When present, the payload is parsed into candidate field values (e.g., invoice ID, issuer metadata) and its region is used as a highconfidence anchor during postprocessing. The QR signal is optional and does not alter training labels.

4.5. Feature Definitions (TF–IDF and Spatial)

Text features (TF–IDF). Let V be the vocabulary (after preprocessing) and N the number of training documents (pages or pagechunks). For term $t \in V$ with document frequency df_t, I compute

$$\mathrm{tf}(t,d) = \mathrm{count}(t \text{ in } d), \qquad \mathrm{idf}(t) = \log\Big(\frac{1+N}{1+df_t}\Big) + 1,$$

and the weight

$$w_{t,d} = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t).$$

The final vector $\mathbf{x}_d^{\text{text}} \in \mathbb{R}^{|V|}$ is L2-normalized: $\mathbf{x} \leftarrow \mathbf{x}/\|\mathbf{x}\|_2$. I use word unigrams with lowercase, English stop-words, whitespace normalization, and a capped vocabulary.

Spatial features. Given a page of width W and height H, a token's axisaligned box (x, y, w, h) (from its polygon extrema) is mapped to normalized coordinates

$$\hat{x} = \frac{x}{W}, \quad \hat{y} = \frac{y}{H}, \quad \hat{w} = \frac{w}{W}, \quad \hat{h} = \frac{h}{H}.$$

I add simple layout indicators (e.g., vertical rank, line index, page quadrant onehot). The final token feature used by classical models is

$$\mathbf{x}_i = \left[\mathbf{x}_{t_i}^{\text{text}} \,;\, \hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i, \text{ layout indicators}\right].$$

4.6. Labels and Supervision

Each token receives one label from the FATURA taxonomy (24 invoice classes such as *DATE*, *NUMBER*, *TOTAL*, *SELLER_ADDRESS*, etc.) plus a background class *OTHER*. Training uses these token labels with crossentropy loss. Field values are formed later by merging consecutive tokens with identical predicted labels.

4.7. Handling Class Imbalance (Inverse-Frequency Weights)

Invoice tokens are not uniformly distributed across classes (e.g., *OTHER* and address fields are common, while *TAX* may be rare). To prevent the model from biasing toward frequent classes, I weight each training example by a factor inversely proportional to its class frequency.

Definition. Let K be the number of classes, N the number of training tokens, and n_c the number of tokens of class c. I use the normalized inverse-frequency weight

$$w_c = \frac{N}{K \cdot n_c},$$

so that the average weight over classes is 1. For extremely small n_c, I optionally smooth the counts with a small constant $\alpha > 0$,

$$w_c = \frac{N}{K \cdot (n_c + \alpha)} \quad \text{and then rescale so that } \frac{1}{K}\sum_c w_c = 1.$$

Effect on the loss. For a training sample $(\mathbf{x}_i, y_i)$ with per-sample loss $\ell(\cdot)$ (e.g., cross-entropy for logistic regression or the transformer, hinge loss for SVM), the weighted objective is

$$\mathscr{L}_{\text{weighted}} = \sum_{i=1}^{N} w_{y_i}\, \ell\big(f(\mathbf{x}_i),\, y_i\big).$$

Misclassifying a minority-class token incurs a larger penalty, encouraging the classifier to learn decision boundaries that better cover rare classes. In tree models, the same

idea is applied via *sample weights* that influence split criteria (e.g., weighted Gini) and bootstrapping.

Practical notes. The points below operationalize the inverse-frequency scheme from Section 4.7: how to set weights in common libraries and how to keep training stable and probabilities well calibrated.

- **scikit-learn (LR, linear SVM, Random Forest):** use `class_weight='balanced'` or provide a dict `{class: `w_c`}`. This implements the formula above internally.
- **XGBoost (multiclass):** pass per-example `sample_weight` equal to w_{y_i} during `fit`. (For binary tasks, `scale_pos_weight` is a shorthand; for multiclass, use `sample_weight`.)
- **Transformer (optional):** if using a cross-entropy loss, provide a class-weight vector to the loss (e.g., PyTorch `CrossEntropyLoss(weight=...)`). In my setup I keep weighting to the classical branch for simplicity unless stated otherwise.
- **Stability:** clip extreme weights to $[w_{\min}, w_{\max}]$ if any class is very rare; mild oversampling of minority classes can be combined with weights. Because weighting can affect probability calibration, I calibrate probabilities on the validation split when needed.

Intuition. Without weighting, a classifier that always predicts the majority class can achieve deceptively high accuracy. Inverse-frequency weights make errors on rare but important fields (e.g., *TOTAL, TAX, DATE*) count more during training, which typically improves macro-averaged metrics and real extraction quality.

4.8. Classical ML Branch (Training)

I train multinomial Logistic Regression, linear SVM, Random Forest, and XGBoost on the token features from Section 4.5 using the labels in Section 4.6. This subsection summarizes the design-level training mechanics; concrete split details and hyperparameter ranges live in Section 5.4.

Inputs and targets. Each token is represented by TF–IDF text features concatenated with normalized spatial/layout features (Section 4.5); the target is a single FATURA class per token (Section 4.6). Continuous features are standardized using statistics from the training split only.

Objectives and optimization. I use the standard objectives from Sections 4.3.2–4.3.5: softmax cross-entropy for Logistic Regression, hinge loss (one-vs-rest) for linear SVM, impurity-driven splits for Random Forest, and second-order boosted trees for XGBoost. Library solvers are used for reliability (LBFGS for LR; linear SVM solver; CART-style tree growth; Newton boosting).

Class imbalance. To avoid majority-class bias, I apply inverse-frequency class weights or per-sample weights as defined in Section 4.7. In tree models, the same weights influence split criteria and bootstrapping.

Artifacts and reproducibility. I persist the fitted TF–IDF vectorizer, scalers, label maps, and model weights; random seeds are fixed. These artifacts are re-used at inference for deterministic behavior.

Pointer to experimental specifics. For split construction, search spaces, and exact hyperparameters (e.g., *C* for LR/SVM; tree counts/depth for RF; rounds, depth, learning rate, and subsampling for XGBoost), see Section 5.4.

Example: Logistic Regression training (built-in class balancing)

```
import numpy as np
from scipy.sparse import hstack, csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import MinMaxScaler          # or StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
# (optional) from sklearn.calibration import CalibratedClassifierCV

# --- Inputs expected (already split) ---
# texts_train : list[str]  OCR tokens for training
# boxes_train : array-like shape (N_train, 4) normalized (x, y, w, h) in [0,1]
# y_train_raw : list[str]  string labels (FATURA classes) for training
# texts_val   : list[str]
# boxes_val   : array-like shape (N_val, 4)
# y_val_raw   : list[str]

# --- Vectorize text on train; transform val ---
vec = TfidfVectorizer(lowercase=True, stop_words="english",
                      ngram_range=(1,1), max_features=50000)
X_text_tr = vec.fit_transform(texts_train)   # (N_tr, |V|)
X_text_va = vec.transform(texts_val)         # (N_va, |V|)

# --- Scale spatial on train; transform val ---
scaler = MinMaxScaler()                      # keeps [0,1]; or StandardScaler
X_sp_tr = scaler.fit_transform(np.asarray(boxes_train))   # (N_tr,4) dense
X_sp_va = scaler.transform(np.asarray(boxes_val))         # (N_va,4) dense

# --- Concatenate text + spatial (CSR) ---
X_tr = hstack([X_text_tr, csr_matrix(X_sp_tr)], format="csr")
X_va = hstack([X_text_va, csr_matrix(X_sp_va)], format="csr")
```

```
# --- Train multinomial LR with built-in inverse-frequency balancing ---
lr = LogisticRegression(
    multi_class="multinomial",
    solver="lbfgs",
    max_iter=300,
    n_jobs=-1,
    C=1.0,
    class_weight="balanced",      # <- balances classes internally
    random_state=42
).fit(X_tr, y_train_raw)          # y can be strings; sklearn handles encoding

# --- Predict on validation and report ---
y_pred = lr.predict(X_va)         # same dtype as y_train_raw (strings)
acc    = accuracy_score(y_val_raw, y_pred)
print(f"Validation accuracy: {acc:.4f}")
print(classification_report(y_val_raw, y_pred)
```

4.9. LayoutLMv3 Branch (Multimodal Transformer)

I fine-tune a pretrained LayoutLM-family model to predict a label for each OCR token. The model jointly uses (i) the subword-tokenized text, (ii) the 2D bounding boxes, and (iii) the page image, which allows it to leverage both linguistic content and document layout [3], [11].

Inputs. For each page I prepare the list of OCR words, their axis-aligned bounding boxes, the page image, and one flat class label per token (the taxonomy from Section 4.6). Bounding boxes are mapped to the integer coordinate space $[0, 1000]$ expected by LayoutLM-family models, using (x_0, y_0, x_1, y_1) coordinates derived from OCR polygons.

Preprocessing and alignment. Subword tokenization is applied to the OCR words; the per-token labels are aligned to subwords (the first subword inherits the token label and continuation subwords are ignored in the loss with the standard -100 mask). Images are converted to RGB and resized/normalized by the model's processor so that text, boxes, and pixels remain aligned after padding and truncation.

Model and objective. I use `LayoutLMv3ForTokenClassification` with a linear classification head over token embeddings. Training minimizes cross-entropy over visible (non-masked) subword positions. Optionally, a class-weight vector can be supplied to the loss to mitigate imbalance (analogous in spirit to the inverse-frequency scheme in Section 4.7); in the main experiments I keep weighting in the classical branch unless stated otherwise.

Training setup. I fine-tune a publicly available pretrained checkpoint rather than pretraining from scratch, as pretraining a vision–language document model requires substantial compute and data. Hyperparameters (learning rate, batch size, epochs, weight decay) are selected on the validation split using macro-F1 over token labels. Checkpoints are saved per epoch and the best model by validation macro-F1 is used for reporting.

Inference and aggregation. At test time the model outputs per-token class probabilities. I take the argmax label per token and then merge consecutive tokens with identical labels into field values (e.g., multi-token dates or addresses).

5. Experiments

This section describes the experimental protocol which tested a standard feature-engineered pipeline against a layout-aware transformer model using identical OCR output data. The design achieves fairness through equal data input and post-processing methods, and it establishes exact dataset divisions which support reproducibility.

5.1. Intra- vs. Inter-Template Evaluation

The primary problem with invoice extraction systems exists in their ability to process different document arrangements.

Intra-template (seen layouts). The training and test pages in intra-template evaluation use templates from the same template set. In this thesis, I train on invoices from all 50 templates and then evaluate on held-out pages drawn from those same 50 templates. This setting assesses performance when documents share the same layout characteristics as the training data—including the same suppliers or closely similar formatting—so scores are typically higher because the model can leverage consistent template-specific patterns.

Inter-template (unseen layouts). The evaluation process of inter-template requires that training and test pages derive from different template sets. The training set contains all templates which are absent from both validation and test sets:

$$\mathcal{T} = \mathcal{T}_{\text{train}} \dot{\cup} \mathcal{T}_{\text{val}} \dot{\cup} \mathcal{T}_{\text{test}}, \qquad \mathcal{D}_{\text{split}} = \bigcup_{T_k \in \mathcal{T}_{\text{split}}} \mathcal{D}_k.$$

This method evaluates how well the model generalizes between different templates when it encounters suppliers or layouts it has not seen before, which better represents real-world deployment scenarios in diverse environments.

Evaluating both regimes is necessary because intra-template assessment demonstrates stable vendor performance, but inter-template evaluation demonstrates how well the system performs when vendors change.

5.2. Split Construction

Splits are created at the page level with awareness of templates to avoid leakage. Tokenization, TF–IDF fitting, feature standardization, and model selection use only the training portion. Random seeds are fixed for reproducibility.

Intra-template (seen layouts). For intra-template evaluation, an 80/20 page-wise random split with a fixed seed is applied. Shuffling is stratified by label distribution when applicable. Hyperparameters for classical models are selected by cross-validation on

training data, but the transformer model depends on its internal development subset for early stopping.

Inter-template (unseen layouts). The training, validation, and test sets of templates must be mutually exclusive. The partition balances template counts across splits, with all pages from a given template assigned to only one split. No template appears in more than one split.

Preprocessing boundaries and leakage control. Statistics such as TF–IDF vocabulary, feature scalers, and class priors are fit on the training portion before being used for validation and test. The transformer model uses its pretrained tokenizer to perform subword tokenization, and bounding boxes are normalized based on page dimensions and, where required, scaled to $[0, 1000]$.

5.3. Input Preparation

Pages are processed with Tesseract OCR (`pytesseract`) to produce word tokens t_i and bounding boxes $b_i = (x_i, y_i, w_i, h_i)$ in pixels. Boxes are normalized by page width and height into a 0–1 range:

$$\hat{x}_i = \frac{x_i}{W},\ \hat{y}_i = \frac{y_i}{H},\ \hat{w}_i = \frac{w_i}{W},\ \hat{h}_i = \frac{h_i}{H},$$

with (W, H) the page width/height. Tokens are ordered in stable reading order (top-to-bottom, left-to-right). For LayoutLMv3, word boxes are clamped to integers in $[0, 1000]$; if a word is split into multiple subwords, its bounding box is copied to all subwords [11].

Implementation hooks (practical). In the current pipeline, preprocessing mainly consists of converting extracted text into a normalized representation suitable for vectorization. Tokens are lowercased and redundant whitespace is reduced through the configuration of the TfidfVectorizer, ensuring consistent input to the model. Geometric features are also computed for each token, including bounding box coordinates, width, height, area, and aspect ratio, which are later standardized for training.

This ensures all tokens are lowercased and extra whitespace does not affect tokenization:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(
    lowercase=True,        # capitalisation unification
    token_pattern=r"\b\w+\b",  # splits on whitespace and normalises tokens
)
```

5.4. Training Configuration

Classical branch configuration. The classical branch represents each token through TF–IDF vectors of word unigrams with vocabulary limits and combines these with normalized geometric features and layout indicators such as line index and page quadrant. Features are standardized on training splits only. This provides a strong yet efficient baseline [8].

Table 5.1. Classical branch: vectorization and feature settings.

Component	Setting
TF–IDF analyzer	words (unigrams)
Stop-words	English
Vocabulary cap	5,000 terms
Normalization	L2 (unit length)
Use IDF / smooth IDF	yes / yes
Text preprocessing	lowercase; whitespace normalization
Spatial features	normalized $(\hat{x}, \hat{y}, \hat{w}, \hat{h}) \in [0,1]^4$
Extra geometry	line index (z-scored), page quadrant one-hot
Feature scaling	`StandardScaler` fitted on train only
Label imbalance	inverse-frequency class weights (when supported)

Hyperparameter selection. Hyperparameters are selected by validation macro-F1. Linear SVM and Logistic Regression vary margin strength ($C \in \{0.1, 1, 10\}$). Random Forest varies the number of trees, and XGBoost varies tree count, depth, learning rate, and subsampling [10].

Table 5.2. Classical branch: hyperparameter search space and fixed options.

Model	Parameter	Values / Setting
Linear SVM	C	{0.1, 1, 10}
	class_weight	`'balanced'`
Logistic Regression	C	{0.1, 1, 10}
	solver / max_iter	`lbfgs` / 1000
	class_weight	`'balanced'`
Random Forest	n_estimators	{100, 300}
	random_state	42
XGBoost	n_estimators (trees)	{200, 400}
	max_depth	{6, 8}
	learning_rate	{0.05, 0.1}
	subsample	0.8
Selection	metric	validation macro-F1 [8], [10]
General	seeds	`random_state` = 42 for splits and models

The main experiments utilize the optimal model parameters according to validation results. The ablation studies evaluate components such as spatial features, class weighting, and model complexity by toggling them on and off.

LayoutLMv3. The LayoutLMv3 model requires a pretrained base model to undergo fine-tuning with AdamW (learning rate 5×10^{-5}, weight decay 0.01), a linear decay schedule, and mixed precision training on GPU. Sequence length is capped at 384 tokens. Logging stores both loss values and token-level F1 scores per epoch [11].

Reproducibility controls. Results are made reproducible with fixed random seeds, recorded software versions, and storage of fitted artifacts including TF–IDF models, scalers, and train/validation/test indices.

5.5. Evaluation Metrics

Two kinds of metrics are reported: token-level classification and field-level exact match.

Why these metrics. We measure how well the classifier predicts token labels given fixed outputs. Token-level precision, recall, and F1 provide per-class diagnostics under imbalance, and macro-F1 is used as the main summary instead of accuracy.

Confusion terms for a class c.

	$\hat{y}_i = c$	$\hat{y}_i \neq c$
$y_i = c$	$\mathbf{TP}_c$	$\mathbf{FN}_c$
$y_i \neq c$	$\mathbf{FP}_c$	$\mathbf{TN}_c$

Per-class precision, recall, and F1.

$$\text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}, \quad \text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}, \quad \text{F1}_c = \frac{2\,\text{Precision}_c\,\text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}.$$

Averaging schemes. Macro-F1 averages all classes equally:

$$\text{MacroF1} = \frac{1}{|\mathscr{C}|} \sum_{c\in\mathscr{C}} \text{F1}_c.$$

Weighted-F1 weights classes by support:

$$\text{WeightedF1} = \sum_{c\in\mathscr{C}} \frac{\text{support}_c}{\sum_{k\in\mathscr{C}} \text{support}_k} \text{F1}_c.$$

Token-level accuracy.

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{\hat{y}_i = y_i\},$$

with N the number of evaluated tokens. Accuracy can be optimistic under imbalance, so macro-F1 is the main summary.

Reporting convention. Unless noted otherwise: the background label (Other) is included in token-level metrics; metrics are aggregated across pages; and detailed per-class results are provided in an appendix for diagnostics.

6. Results

The section presents numerical and descriptive findings about the classical pipeline and the layout-aware transformer models which process identical OCR tokens and bounding boxes. The headline token-level metric is macro-F1; weighted-F1 and accuracy are provided for context. The report includes field-level exact match results after span aggregation when such data exists.

6.1. Main Comparison under Identical Inputs

Model panel rationale. We use a two-stage panel. First, on the easier intra-template split (Tables 6.1 and 6.3) we screen a broader set of classical learners to compare families: linear SVM, polynomial and RBF SVM, logistic regression, random forest, and with spatial features also XGBoost. Second, on the harder inter-template split (Tables 6.2 and 6.4) we retain only the strongest classical baselines from the screening (linear SVM and logistic regression for text-only; linear SVM and XGBoost for text+spatial) and add a layout-aware transformer as a reference upper bound. Lower-performing or redundant models from the screening (such as polynomial and RBF SVM in the text-only case) are omitted on inter-template to reduce redundancy, variance in results, and computation.

We evaluate two regimes: intra-template (seen layouts) and inter-template (unseen layouts) (Section 5.1). Tables 6.1–6.4 summarize the token-level results.

Table 6.1. FATURA, intra-template (random hold-out): text-only models on Fatura JSON tokens. Reported: Weighted-F1, Macro-F1, and Accuracy.

Model (features)	Weighted-F1	Macro-F1	Accuracy
Linear SVM (text only)	0.87	0.87	0.88
Linear SVM (text only, class-weighted)	0.87	0.87	0.88
Poly SVM, deg=3 (text only)	0.86	0.86	0.87
RBF SVM, $\gamma = 5$, $C = 5$ (text only)	0.83	0.75	0.85
Random Forest, 100 trees (text only)	0.87	0.87	0.87
Logistic Regression, lbfgs (text only)	0.87	0.86	0.88

Table 6.2. FATURA, inter-template (templates disjoint): text-only models on Fatura JSON tokens. Reported: Weighted-F1, Macro-F1, and Accuracy.

Model (features)	Weighted-F1	Macro-F1	Accuracy
Logistic Regression, lbfgs (text only)	0.72	0.71	0.77
Linear SVM (text only)	0.72	0.71	0.76

Table 6.3. FATURA, intra-template (random hold-out): text+spatial models (TF–IDF unigrams + normalized (x, y, w, h, line_idx)). Reported: Weighted-F1, Macro-F1, Accuracy.

Model (features)	Weighted-F1	Macro-F1	Accuracy
Linear SVM (text + spatial)	1.0000	0.9800	1.0000
Random Forest (text + spatial)	1.0000	1.0000	1.0000
Logistic Regression (text + spatial)	1.0000	0.9400	1.0000
XGBoost (text + spatial)	1.0000	1.0000	1.0000

Table 6.4. FATURA, inter-template (templates disjoint): text+spatial models. We report the two strongest classical baselines (linear SVM, XGBoost) and a layout-aware transformer as a reference upper bound. Reported: Weighted-F1, Macro-F1, Accuracy.

Model (features)	Weighted-F1	Macro-F1	Accuracy
Linear SVM (text + spatial, class-weighted)	0.9100	0.9000	0.9230
XGBoost (text + spatial, none)	0.9600	0.9700	0.9615
LayoutLMv2/3 (text + layout + image)	0.9998	0.9998	0.9998

Summary of trends. Text-only models on the intra-template split (Table 6.1) achieve macro-F1 scores between 0.86 and 0.87 and accuracy scores between 0.87 and 0.88. On the inter-template split (Table 6.2), text-only models achieve macro-F1 scores of about 0.71 and accuracy between 0.76 and 0.77. When spatial features are added, classical models on the intra-template split (Table 6.3) reach nearly perfect results, with macro-F1 and accuracy scores approaching 1.00. On the inter-template split (Table 6.4), combining text with spatial features produces stronger results than text alone, for example linear SVM achieves 0.90 macro-F1 and 0.923 accuracy while XGBoost achieves 0.97 macro-F1 and 0.9615 accuracy. The layout-aware transformer (Table 6.4) reaches 0.9998 accuracy, showing excellent generalization to unseen layouts.

6.2. Per-Class Performance (Token Level)

The performance metrics of precision, recall, and F1 for each class show which GST rate classes contain errors and how class distribution affects macro vs. weighted scores. Table 6.5 shows the intra-template results for the Linear SVM model using text and spatial features with class weighting. The Other class has the highest support, so macro-F1 is the most suitable evaluation metric.

Table 6.5. Per-class token metrics on FATURA, intra-template, for Linear SVM (text+spatial, class-weighted). Values from the notebook's classification$_report$.

Class	Precision	Recall	F1	Support
AMOUNT_DUE	0.99	1.00	1.00	158
BILL_TO	1.00	1.00	1.00	335
BUYER	1.00	1.00	1.00	1299
CONDITIONS	1.00	1.00	1.00	436
DATE	1.00	1.00	1.00	1993
DISCOUNT	1.00	1.00	1.00	495
DUE_DATE	1.00	1.00	1.00	1188
GST(1%)	0.98	0.93	0.95	45
GST(12%)	0.85	1.00	0.92	33
GST(18%)	0.95	1.00	0.98	121
GST(20%)	0.94	0.98	0.96	48
GST(5%)	1.00	0.74	0.85	43
GST(7%)	0.82	0.82	0.82	34
GST(9%)	0.85	0.84	0.84	68
GSTIN	1.00	1.00	1.00	253
GSTIN_BUYER	1.00	1.00	1.00	111
GSTIN_SELLER	1.00	1.00	1.00	199
NOTE	1.00	1.00	1.00	1058
NUMBER	1.00	1.00	1.00	1769
OTHER	1.00	1.00	1.00	1994
PAYMENT_DETAILS	1.00	1.00	1.00	526
PO_NUMBER	1.00	1.00	1.00	288
SELLER_ADDRESS	1.00	1.00	1.00	1631
SELLER_EMAIL	1.00	1.00	1.00	875
SELLER_NAME	1.00	1.00	1.00	1333
SELLER_SITE	1.00	1.00	1.00	261
SEND_TO	1.00	1.00	1.00	366
SUB_TOTAL	1.00	1.00	1.00	1335
TAX	1.00	1.00	1.00	770
TITLE	1.00	1.00	1.00	1521
TOTAL	1.00	1.00	1.00	1556
TOTAL_WORDS	1.00	1.00	1.00	777
accuracy			1.00	22,919
macro avg	0.98	0.98	0.98	22,919
weighted avg	1.00	1.00	1.00	22,919

Per-class observations. Table 6.5 reports the intra-template per-class results for a linear SVM baseline, while Table 6.6 reports the per-class F1 scores obtained with LayoutLM in the inter-template setting. The transformer reaches perfect performance across all classes, including minority GST rate categories that were more difficult for classical baselines. This confirms that LayoutLM leverages contextual, spatial, and visual cues to disambiguate classes such as GST(5%), GST(7%), and GST(9%), which remain challenging in the classical pipeline. In contrast, the SVM baseline (Table 6.5) shows lower recall and F1 for these classes due to small support and high lexical similarity. The comparison highlights the robustness of transformer-based models in generalizing to unseen layouts.

Table 6.6. Per-class F1 scores for LayoutLM in the inter-template evaluation.

Class	F1
GST(5%)	1.00
GST(7%)	1.00
GST(9%)	1.00
GST(12%)	1.00
GST(18%)	1.00
GST(20%)	1.00
AMOUNT_DUE	1.00
BILL_TO	1.00
DATE	1.00
DISCOUNT	1.00
BUYER	1.00
CONDITIONS	1.00
DUE_DATE	1.00

6.3. Reporting Notes and Reproducibility

The main evaluation metric is macro-F1 unless otherwise specified, while weighted-F1 and accuracy serve as additional evaluation metrics. Transformer accuracy on inter-template comes from the same experimental run used for the main comparison, while per-class F1 for that row was not logged in the current run. Seeds, split indices, fitted artifacts (TF–IDF models, scalers, label maps), and environment versions are stored to enable exact re-runs.

7. Discussion and Summary

7.1. Summary of Findings

The experiments evaluate two methods which employ identical tokenization and bounding box processing: (i) feature-engineered token classifiers that use TF–IDF and normalized 2D coordinates and (ii) the layout-aware transformer models LayoutLMv2/3 that combine text data with layout and image information [3], [11]. Three main points emerge from the results:

(a) Seen layouts (intra-template). Classical models with spatial features achieve near-perfect token accuracy on held-out pages of the same templates. The results indicate that most predictive information stems from geometric patterns including stable header and footer positions and section order which linear and tree models can easily detect.

(b) Unseen layouts (inter-template). The integration of spatial features in classical models leads to better performance than text-based models but the layout-aware transformer outperforms all models for detecting new templates. The results confirm previous research which demonstrates that using text alongside layout and image information leads to better cross-template generalization [3].

(c) Per-class behavior. The system achieves high performance in structured fields including NUMBER, DATE and TOTAL but struggles with fine-grained tax fields like GST(7%) and GST(9%) and long free-text spans like SELLER_ADDRESS. The gap between macro-F1 and weighted-F1 reflects the class imbalance in the FATURA dataset [6].

7.2. Classical Models vs. Layout-Aware Transformers

The models that use feature engineering work best when the page structure remains constant between different pages (intra-template) because they provide fast results and human-readable explanations. However, their performance declines when layouts change, since their decision boundaries are tied to the training feature patterns. LayoutLMv2/3 uses learned 2D positional embeddings and visual features to identify visually similar zones (for example, Subtotal vs. Total) and to retrieve context information that exists after OCR splits. The method provides better resistance to layout changes at the expense of increased computational requirements [3], [11].

7.3. Practical implications.

Overall, the experiments reveal a clear division of strengths. Classical models augmented with spatial features are an efficient choice for stable intra-template deployments—fast to train, inexpensive to run, and near-perfect on token-level F1—whereas the transformer excels when layouts vary across templates. The production environment

requires transformer-based models to achieve robustness and generalization because it needs to handle documents from unexpected templates. Their performance under distribution shift conditions is enhanced because they process both global context and layout information which makes them suitable for deployment at scale.

7.4. Threats to Validity

The FATURA dataset includes 10,000 invoices which are distributed across 50 different templates with labels that correspond to each invoice type [6]. Although large and diverse, the synthetic or generated layouts may not capture all artifacts of real-world scans such as stamps or handwriting, so absolute performance may differ on live data.

Template leakage. The inter-template experiments use separate template sets for each experiment where TF–IDF, scalers and other statistics are trained only on the training data. Any leakage would bias results upward.

OCR dependence. Both approaches rely on OCR tokens. The ongoing presence of systematic OCR errors prevents subsequent operations because transformers need image data yet their performance remains restricted by the current high rates of OCR failure. The final results will depend on the OCR engine chosen and the particular settings applied.

Hyperparameters and randomness. The learning process of transformers depends on three hyperparameters: learning rate, batch size and sequence length, whereas classical models depend on regularization and feature scaling. Fixed seeds and protocols reduce variance but cannot eliminate it.

Metric choice. The selection of metrics should not focus only on token-level accuracy, since this metric becomes deceptive when there are imbalances in the data. The main evaluation metric is macro-F1 but field-level exact match provides additional information about real-world performance.

Reproducibility. The study records software versions, trained artifacts (TF–IDF, scalers, label maps) and split indices to ensure deterministic re-runs. Hardware differences (CPU/GPU) affect throughput but not the protocol itself.

7.5. Possible Improvements

Invocie Classification One possible extension is to introduce a document-level classifier that first predicts whether an input invoice belongs to a known (seen) template or to an unseen template. In the case of a known template, the system could route the document to a lightweight traditional model such as an SVM with spatial features, which is fast

and already achieves near-perfect performance in this regime. For unseen templates, the document could instead be routed to a transformer-based model such as LayoutLM, which has demonstrated superior robustness and generalization in inter-template evaluation (Table 6.4 and Table 6.6). Such a hybrid architecture could balance efficiency and accuracy by applying the appropriate model depending on template familiarity.

Multilingual and cross-domain transfer. The evaluation of language-agnostic layout encoders like LiLT for processing multilingual invoices and achieving cross-lingual transfer represents a promising research path. The system needs to combine multilingual OCR processing with normalization functions to achieve this goal [4].

Line-item extraction. Future systems will include table structure detection and recognition capabilities to extract line items which will transform token-level labels into complete item-level JSON data while performing total amount validation through row-sum consistency checks.

OCR-free or hybrid paths. The evaluation process should include testing OCR-free models like Donut against low-quality and multi-script scanned documents. The system should direct documents through a page quality assessment system to determine their processing route [12].

Robustness assessment. The research needs to perform robustness tests using synthetic distortions which include image blurring and JPEG compression and random document cropping. The evaluation of performance under different scan conditions requires robustness curve analysis which plots macro-F1 scores against perturbation intensity levels.

Bibliography

[1] Y. Xu, M. Li, L. Cui, S. Huang, F. Wei, and M. Zhou, "Layoutlm: Pre-training of text and layout for document image understanding", *arXiv preprint arXiv:1912.13318*, 2020. [Online]. Available: https://arxiv.org/abs/1912.13318.

[2] A. R. Katti, C. Reisswig, C. Guder, *et al.*, "Chargrid: Towards understanding 2d documents", *arXiv preprint arXiv:1809.08799*, 2018. [Online]. Available: https://arxiv.org/abs/1809.08799.

[3] Y. Xu, T. Lv, L. Cui, *et al.*, "Layoutlmv2: Multi-modal pre-training for visually-rich document understanding", *arXiv preprint arXiv:2012.14740*, 2020. [Online]. Available: https://arxiv.org/abs/2012.14740.

[4] J. Wang, Y. Xu, L. Cui, W. Che, and F. Wei, "Lilt: A simple yet effective language-independent layout transformer for structured document understanding", *arXiv preprint arXiv:2202.13669*, 2022. [Online]. Available: https://arxiv.org/abs/2202.13669.

[5] S. Appalaraju, B. Jasani, D. Uthus, S. Xie, R. Krishnamoorthi, and P. Natarajan, "Docformer: End-to-end transformer for document understanding", *arXiv preprint arXiv:2106.11539*, 2021. [Online]. Available: https://arxiv.org/abs/2106.11539.

[6] M. Limam, M. Dhiaf, and Y. Kessentini, "Fatura: A multi-layout invoice image dataset for document analysis and understanding", *arXiv preprint arXiv:2311.11856*, 2023. [Online]. Available: https://arxiv.org/abs/2311.11856.

[7] J. Sánchez and G. A. Cuervo-Londoño, "A bag-of-words approach for information extraction from electricity invoices", *AI*, vol. 5, no. 4, pp. 1837–1857, 2024. DOI: 10.3390/ai5040091.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://jmlr.org/papers/v12/pedregosa11a.html.

[9] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd. O'Reilly Media, 2019, ISBN: 9781492032649. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", *arXiv preprint arXiv:1603.02754*, 2016. [Online]. Available: https://arxiv.org/abs/1603.02754.

[11] S. Huang, T. Lv, L. Cui, *et al.*, "Layoutlmv3: Pre-training for document ai with unified text and image masking", *arXiv preprint arXiv:2204.08387*, 2022. [Online]. Available: https://arxiv.org/abs/2204.08387.

[12] G. Kim, T. Hong, S. Yun, W. Kim, S. Yoon, and A. Oh, "Donut: Ocr-free document understanding transformer", *arXiv preprint arXiv:2111.15664*, 2021. [Online]. Available: https://arxiv.org/abs/2111.15664.

[13] G. Jaume, H. K. Ekenel, and J.-P. Thiran, “Funsd: A dataset for form understanding in noisy scanned documents”, *arXiv preprint arXiv:1905.13538*, 2019. [Online]. Available: https://arxiv.org/abs/1905.13538.

[14] A. W. Harley, A. Ufkes, and K. G. Derpanis, “Evaluation of deep convolutional nets for document image classification and retrieval”, *arXiv preprint arXiv:1502.07058*, 2015. [Online]. Available: https://arxiv.org/abs/1502.07058.

[15] Z. Huang, K. Chen, J. He, *et al.*, “Icdar2019 competition on scanned receipt ocr and information extraction”, *arXiv preprint arXiv:2103.10213*, 2021. [Online]. Available: https://arxiv.org/abs/2103.10213.

List of Figures

List of Tables