Helwan University

Faculty of Computer Science And Artificial Intelligence

Department of Artificial Intelligence

Invoice Information Extraction (IE)

Abdelhalim Ashraf	20210486
Ali Adel	20210577
Omar Tamer	20210598
Abdel Rahman Ramadan	20210505
Mohi El-Din Maher	20210887
Eid Osama Eid	20210648

Supervisor: Dr. Islam Jamal

May 11, 2025

Contents

Chapter 1

Introduction

In the modern world, businesses and individuals increasingly rely on **digital solutions** to manage their finances. One major bottleneck in this process is the **manual extraction of information** from receipts and invoices, which is time-consuming, error-prone, and expensive at scale.

To address this challenge, our project focuses on automating **Invoice Information Extraction (IIE)** by **fine-tuning a powerful multimodal Large Language Model (LLM)**—specifically, Qwen2-VL-2B-Instruct. This model is capable of **understanding and reasoning over both visual (image) and textual (OCR) content**, making it especially suited for complex documents like receipts.

Our system takes a **scanned image of a receipt** as input and outputs a **structured JSON object** containing key information such as:

- **Menu Items:**
  - Grilled Chicken Sandwich
    Quantity: 1
    Price: 75.00
  - Sub-item: Extra Cheese — Quantity: 1, Price: 10.00
  - Iced Coffee
    Quantity: 2
    Price: 20.00
    No sub-items
- **Subtotal:**
  - Subtotal Price: 125.00
  - Tax: 10.00
- **Total:**
  - Total Price: 135.00
  - Paid via E-Money: 135.00

This enables full automation of the information extraction pipeline, reducing the need for manual data entry and improving consistency.

Why Multimodal LLM?

Traditional models require separate OCR engines and rule-based parsers. In contrast, our **multimodal LLM** directly understands the **layout, content, and context** in a single step by jointly processing both:

- **The receipt image**, which captures layout, fonts, and structure.
- **OCR text**, which provides raw content.

By fine-tuning the model on labeled invoice data (CORD-V2), we teach it to recognize common fields and learn **how to organize them meaningfully**—even when layouts differ.

Real-World Applications

Our system has broad applicability in many domains:

1. **Expense Tracking Apps:**
   Apps like Expensify or Mint can automatically pull transaction data from uploaded receipts—eliminating manual entry for users.
2. **Automated Bookkeeping:**
   Accounting software can ingest hundreds of receipts per day from various sources (employees, vendors) and organize them without human supervision.
3. **Retailer Financial Analytics:**
   Retailers can analyze spending patterns, item popularity, and vendor activity by aggregating structured data from invoices and receipts.

Chapter 2

Dataset

CORD-V2 Dataset Overview

Dataset Name: CORD-V2 (Consolidated Receipt Dataset - Version 2)
Source: CORD GitHub Repository

Size & Content

- 1,000 scanned receipt images
- Each receipt is accompanied by OCR-annotated text
- Annotations include bounding boxes and labeled entities such as:
  - `store-name`, `menu-item`, `date`, `total-amount`, and more

Structure

- Each data sample consists of:
  - A receipt image
  - Corresponding OCR text
  - Labeled fields with coordinates
- Entities are categorized for structured extraction tasks, enabling the model to learn detailed invoice layouts and key-value relations.

Preprocessing Steps

To prepare the data for fine-tuning the multi-modal LLM, the following steps were applied:

- **Image Preprocessing:**
  - All receipt images were loaded and resized while maintaining their aspect ratio, ensuring a maximum side length of 600 pixels. This improves model compatibility and speeds up training/inference.

Figure 2.1: Example of scanned receipts from the CORD-V2 dataset

- **Instruction-style Prompt Construction:**
  - Annotated data was transformed into prompt-response pairs, simulating how a user might query specific information (e.g., "Extract the total price from this receipt").
  - These prompts were paired with the corresponding structured outputs using a custom Pydantic schema.
- **Text Normalization & Cleaning:**
  - Labels were normalized to ensure consistency in field names across samples.
  - Incomplete or corrupted entries were filtered out to maintain data quality.
- **OCR + Visual Fusion:**
  - For each image, OCR-extracted text was combined with its visual representation to feed into the multi-modal model.
- **Structured Output Formatting:**
  - The expected output from the model was formatted using a predefined Pydantic schema (`InvoiceDetails`) that includes:
    - Menu items and sub-items (name, quantity, price)
    - Subtotal and tax
    - Final total and payment methods (e.g., electronic money)
- **Result Extraction:**
  - After inference, relevant fields (e.g., `total_price`, `tax_price`, `cashprice`, and `changeprice`) were programmatically extracted from the model's output for evaluation and comparison.

This robust preprocessing pipeline ensured the dataset was well-structured, instruction-ready, and optimized for fine-tuning the **Qwen2-VL-2B-Instruct** model using LoRA adaptation.

Chapter 3

Model Architecture

Base Model Information

Model Name: Qwen/Qwen2-VL-2B-Instruct
Paper: Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution

Architecture Highlights

- A **Vision-Language Large Language Model (VLLM)** with **2 billion parameters**
- Combines visual and textual modalities by jointly encoding both types of input
- Instruction-tuned to follow natural language prompts for a wide range of multi-modal tasks
- Optimized for low-resource deployment compared to larger multimodal models (e.g., GPT-4V, Flamingo)

Key Capabilities

- **Multimodal Reasoning:** Understands and correlates visual elements (like tables, logos, and handwriting) with textual queries
- **Visual Question Answering (VQA):** Can accurately answer questions about image content, such as "What is the total amount on this receipt?"
- **Image Captioning:** Describes images using coherent, structured text
- **Document Layout Understanding:** Leverages spatial relationships between elements in structured documents like receipts, invoices, or forms

Relevance to Our Use Case

- The model's ability to jointly process **images and OCR text** makes it especially effective for **Invoice Information Extraction**.
- Instruction-tuning allows prompting the model to extract specific fields (e.g., total, merchant name) in a controlled and structured format.

Chapter 4

Fine-Tuning with LoRA

What is LoRA?

LoRA (Low-Rank Adaptation) is a parameter-efficient fine-tuning technique that introduces small, trainable low-rank matrices into pre-trained model layers. This approach enables efficient adaptation while keeping most of the original model weights frozen.

How We Applied LoRA

- **Scope of Adaptation:** Only the **attention layers** in both the **vision and text encoders** were adapted.
- **Library Used:** PEFT (Parameter-Efficient Fine-Tuning) with either `QLoRA` or a custom `LoRAConfig`.
- **Hardware:** Enabled fine-tuning on a single GPU (16–24 GB memory).
- **Efficiency:** Allowed full fine-tuning without needing to load or update all model parameters.

Training Hyperparameters

Hyperparameter	Value
`learning_rate`	1e-4
`train_batch_size`	1
`eval_batch_size`	1
`gradient_accumulation`	4
`total_train_batch_size`	4
`optimizer`	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
`epsilon`	1e-08
`lr_scheduler_type`	Cosine
`warmup_ratio`	0.1
`seed`	42
`num_epochs`	3.0

Table 4.1: Training hyperparameters used during fine-tuning

Training Progress

Epoch	Step	Training Loss	Validation Loss
0.5	100	0.0779	0.0685
1.0	200	0.0647	0.0511
1.5	300	0.0292	0.0500
2.0	400	0.0280	0.0449
2.5	500	0.0130	0.0488
3.0	600	0.0116	0.0481

Table 4.2: Loss progression over training epochs

Key Benefits

- **Memory Efficiency:** Adapted only a small subset of parameters.
- **Faster Training:** Reduced training time while retaining high performance.
- **Lower Risk of Overfitting:** LoRA's constrained updates minimize overfitting.

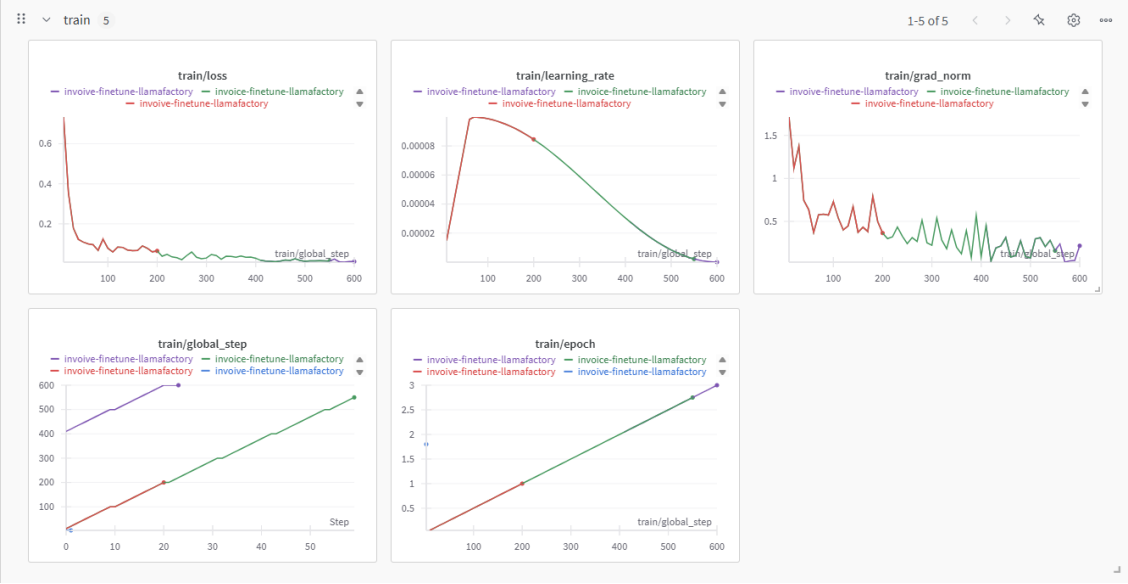

Figure 4.1: Model Training Loss

Chapter 5

Evaluation Methods

In this section, we evaluate the performance of the **LoRA fine-tuned model** against the **base model** using two primary methods: a **comparison of missed values** and an **overall performance evaluation** based on **precision**, **recall**, and **F1-score**. The following steps are taken to analyze and visualize the results.

5.1 Comparison of Missed Values

The first evaluation metric focuses on comparing the missed values for four fields: `total_price`, `tax_price`, `cashprice`, and `changeprice`. This comparison shows how the models performed on each field by plotting the **missed percentage** of predictions for both the **LoRA fine-tuned model** and the **base model**.

Methodology

The number of missed predictions for each field is calculated, and the percentage of misses is plotted for both models. This allows us to visually assess how well each model is performing on different fields.

Visualization

A side-by-side bar chart is created to compare the missed percentage for each field. The height of the bars represents the percentage of missed predictions, with a clear distinction between the LoRA model (shown in `cadetblue`) and the base model (shown in `orange`).

5.2 Overall Model Performance

The second evaluation method focuses on overall metrics: **precision**, **recall**, and **F1-score**, computed across all fields as a multi-class classification task. These metrics offer a comprehensive view of the model's ability to correctly identify and classify the different fields.

Metrics

- **Precision:** Measures the percentage of true positive predictions out of all positive predictions made by the model.
- **Recall:** Measures the percentage of true positive predictions out of all actual positives.

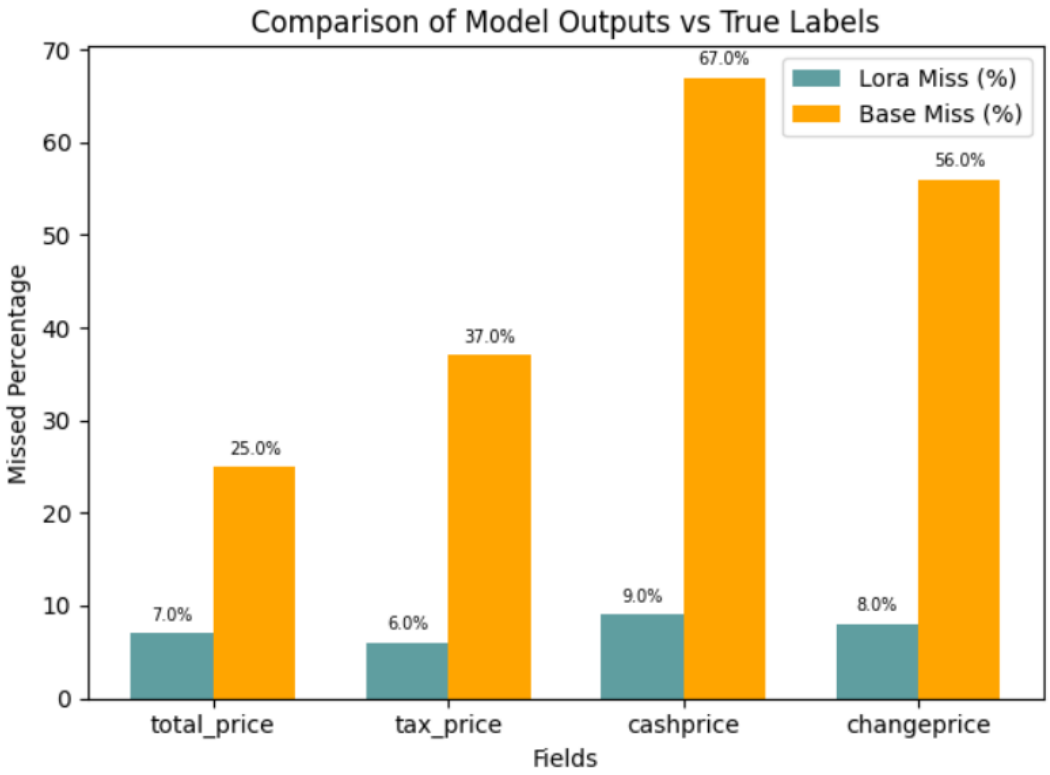

Figure 5.1: Comparison of Missed Values: LoRA fine-tuned model vs Base model for fields: total_price, tax_price, cashprice, and changeprice.

- **F1-score:** The harmonic mean of precision and recall, providing a balance between the two.

Methodology

For each model, **true labels** and **predicted values** are compared for all fields, and the **overall precision**, **recall**, and **F1-score** are computed. These metrics are then visualized on a bar chart that shows the performance for both models across individual fields, as well as the overall performance.

Visualization

A bar chart is generated for the **accuracy per field** for both models, as well as the **overall precision**, **recall**, and **F1-score**. This provides a clear comparison of how each model performs across different metrics.

These evaluations help to demonstrate the strengths and weaknesses of the LoRA fine-tuned model in comparison to the base model, providing insights into the effectiveness of the fine-tuning process.

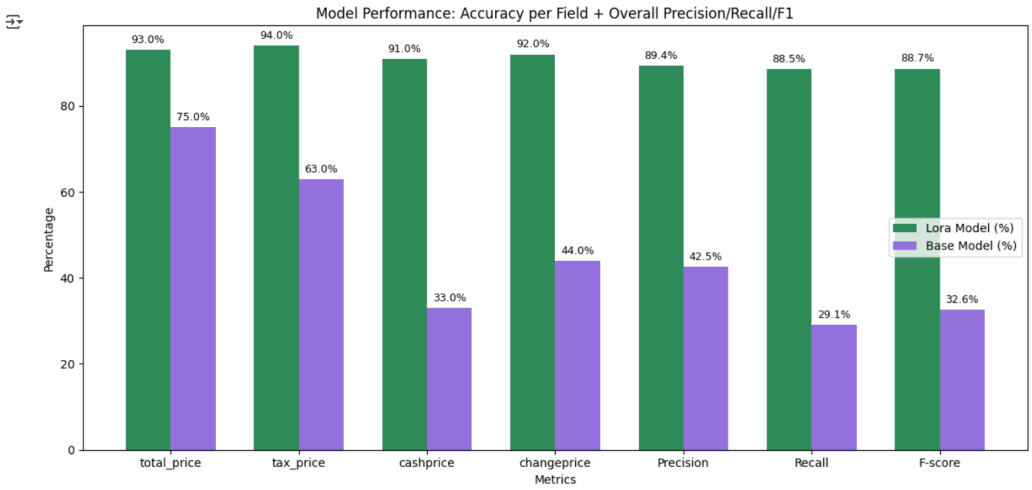

Figure 5.2: Overall Performance Evaluation: Comparison of accuracy, precision, recall, and F1-score for LoRA fine-tuned model and Base model.

Chapter 6

Model Output

The model outputs predictions in a structured **JSON format**, which helps in organizing the predicted data into clearly defined fields. This JSON structure is helpful for downstream processing, analysis, and easy integration with other systems. The format is designed to include important details of the invoice, such as menu items, prices, subtotal, and total.

Example of JSON Output

The output of the model is presented as a JSON object, which contains key information such as:

- **Menu Items**: The list of items purchased.
- **Prices**: The price for each menu item.
- **Subtotal**: The total price of the items before taxes.
- **Total**: The total price after applying taxes.

An example of the JSON structure could look like this:

```
{
  "invoice_id": "12345",
  "merchant_name": "Restaurant XYZ",
  "items": [
    {"item_name": "Pizza", "price": 12.99},
    {"item_name": "Soda", "price": 2.50}
  ],
  "subtotal": 15.49,
  "tax": 1.50,
  "total": 16.99,
  "date": "2023-10-01"
}
```

This structured output allows for easy extraction and processing of information such as item details, prices, and the overall total, enabling seamless integration with other systems such as billing and accounting software.

Example Invoice

Below is an example of an invoice that the model processes to generate the corresponding JSON output:

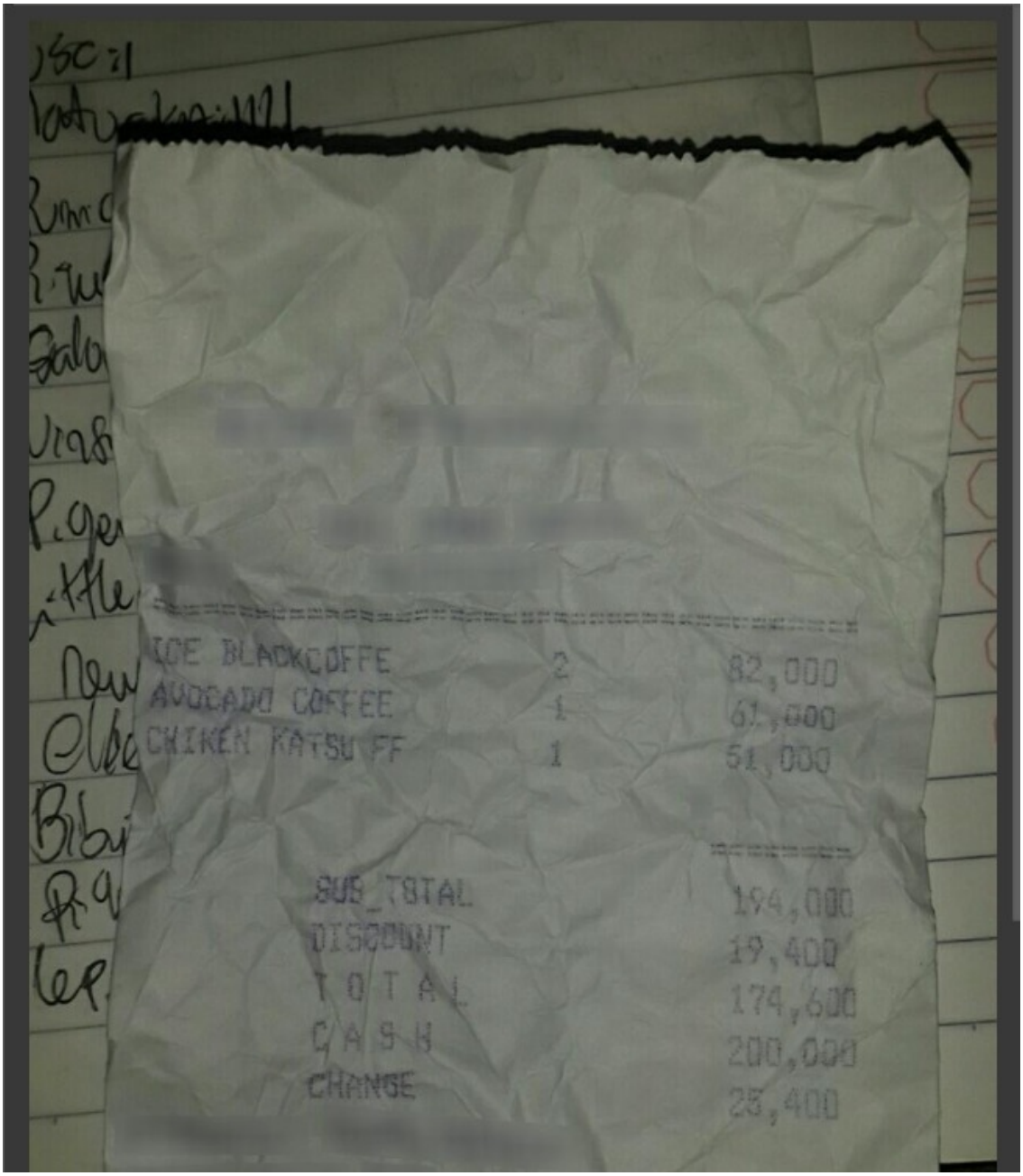

Figure 6.1: Sample Invoice: An example of an invoice that is processed to generate the JSON output.

Visualizing the JSON Output

Below is a visualization of the JSON structure based on the model's output:

{'menu": [{"nm": "ICE BLACKCOFFEE", "cnt": "2", "price": "82,000"}, {"nm":
"AVOCADO COFFEE", "cnt": "1", "price": "61,000"}, {"nm": "CHIKEN KATSU FF",
"cnt": "1", "price": "51,000"}], "sub_total": {"subtotal_price": "154,000",
"discount_price": "19,400"}, "total": {"total_price": "174,600", "cashprice"
"200,000", "changeprice": "25,400"}}': ''}

Figure 6.2: JSON Output: The structured JSON format containing the invoice details such as menu items, prices, and totals.