Aplicación LLM para aplicar OCR de texto a JSON

Modelos y entrenamiento

Marc Pomar · CTO en Kirbic
marc@kirbic.com

Contexto y Librerías

- Hugging Face Transformers
- Modelos Pytorch en Timm

Hugging Face Transformers

Transformers is a library of pretrained natural language processing, computer vision, audio, and multimodal models for inference and training. Use Transformers to train models on your data, build inference applications, and generate text with large language models.

- HF Transformers Tasks

> ⚠️ Antes de empezar el proyecto creamos un nuevo virtual environment

```
python3 -m venv env
source env/bin/activate
```

Objetivo

Tomar la imagen de un ticket y alimentar directamente el modelo LLM para generar un objeto json correctamente formateado.

Task: Image-to-structure data conversion

Opción A)

```
[Ticket Image] >> [LLM] >> [Json object]
```

Opción B)

```
[Ticket Image] >> [OCR Engine] >> [Text Boxes] >> [LLM] >> [Json object]
```

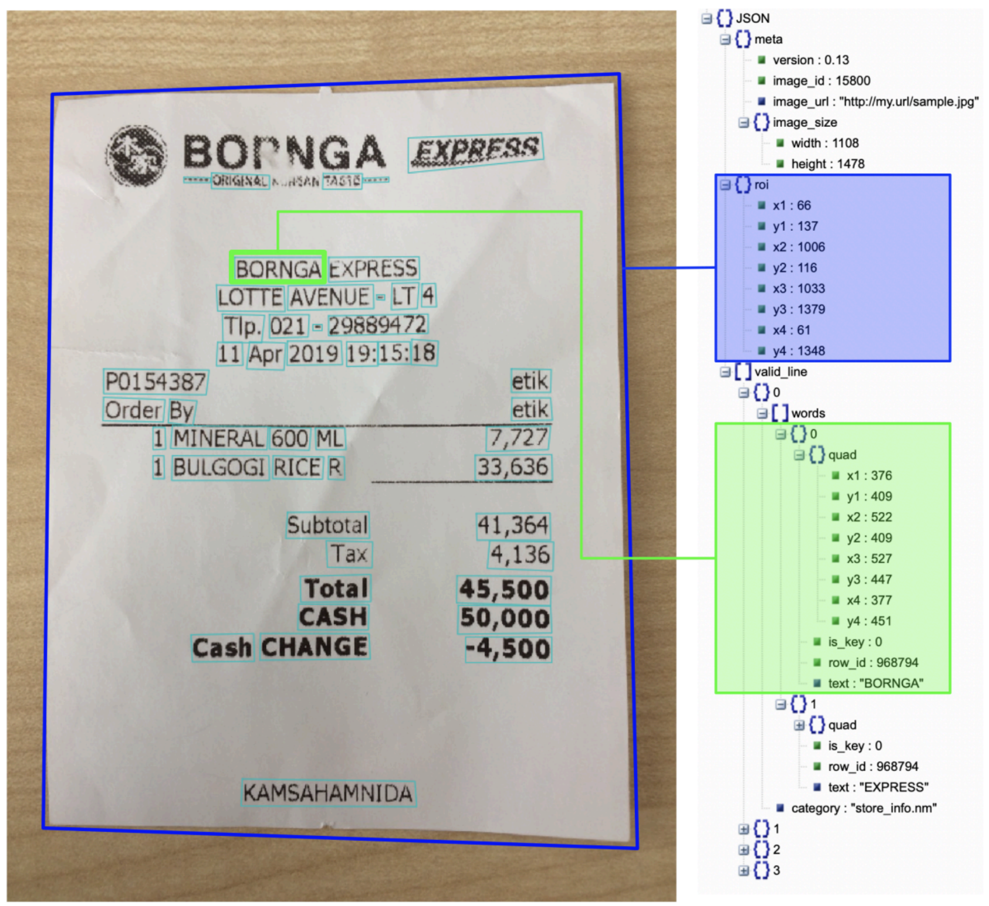
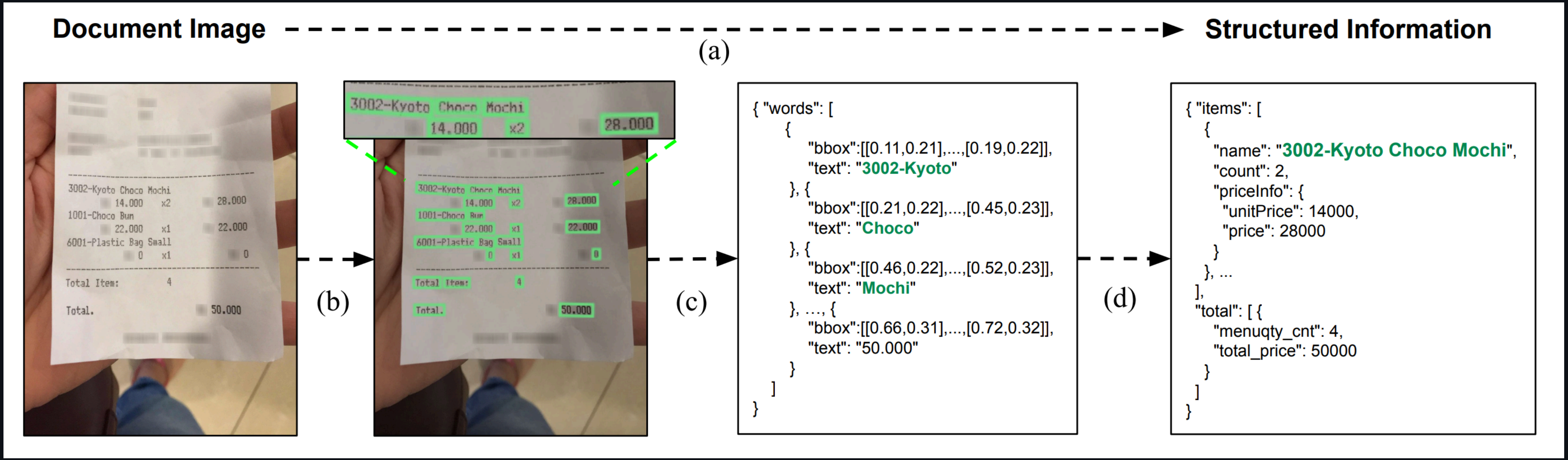


Figure 1: An example of receipt image (left) and json (right).


Document Image
(a)
Structured Information
3002-Kyoto Choco Mochi
14.000 x2 28.000
1001-Choco Bun
22.000 x1 22.000
6001-Plastic Bag Small
0 x1 0
Total Item: 4
Total. 50.000
(b)
(c)
(d)
{ "words": [
{
"bbox":[[0.11,0.21],...,[0.19,0.22]],
"text": "3002-Kyoto"
}, {
"bbox":[[0.21,0.22],...,[0.45,0.23]],
"text": "Choco"
}, {
"bbox":[[0.46,0.22],...,[0.52,0.23]],
"text": "Mochi"
}, ..., {
"bbox":[[0.66,0.31],...,[0.72,0.32]],
"text": "50.000"
}
]
}
{ "items": [
{
"name": "3002-Kyoto Choco Mochi",
"count": 2,
"priceInfo": {
"unitPrice": 14000,
"price": 28000
}
}, ...
],
"total": [{
"menuqty_cnt": 4,
"total_price": 50000
}
]
}

```json
{
  "receipt": {
    "store": "The Lone Pine",
    "address": "43 Manchester Road",
    "phone": "617-3236-6207",
    "invoice": "Invoice 08000008",
    "date": "09/04/08",
    "table": "Table",
    "items": [
      { "name": "Carlsberg Bottle", "price": "16.00", "quantity": "2" },
      { "name": "Heineken Draft Standard.", "price": "15.20", "quantity": "1" },
      {
        "name": "Heineken Draft Half Liter.",
        "price": "15.20",
        "quantity": "1"
      },
      {
        "name": "Carlsberg Bucket (5 bottles).",
        "price": "80.00",
        "quantity": "1"
      },
      { "name": "Grilled Chicken Breast.", "price": "74.00", "quantity": "1" },
      { "name": "Sirloin Steak", "price": "96.00", "quantity": "1" },
      { "name": "Coke", "price": "3.50", "quantity": "1" },
      { "name": "Ice Cream", "price": "18.00", "quantity": "5" }
    ],
    "subtotal": "327.30",
    "tax": "16.36",
    "service_charge": "32.73",
    "total": "400.00"
  }
}
```

Tareas

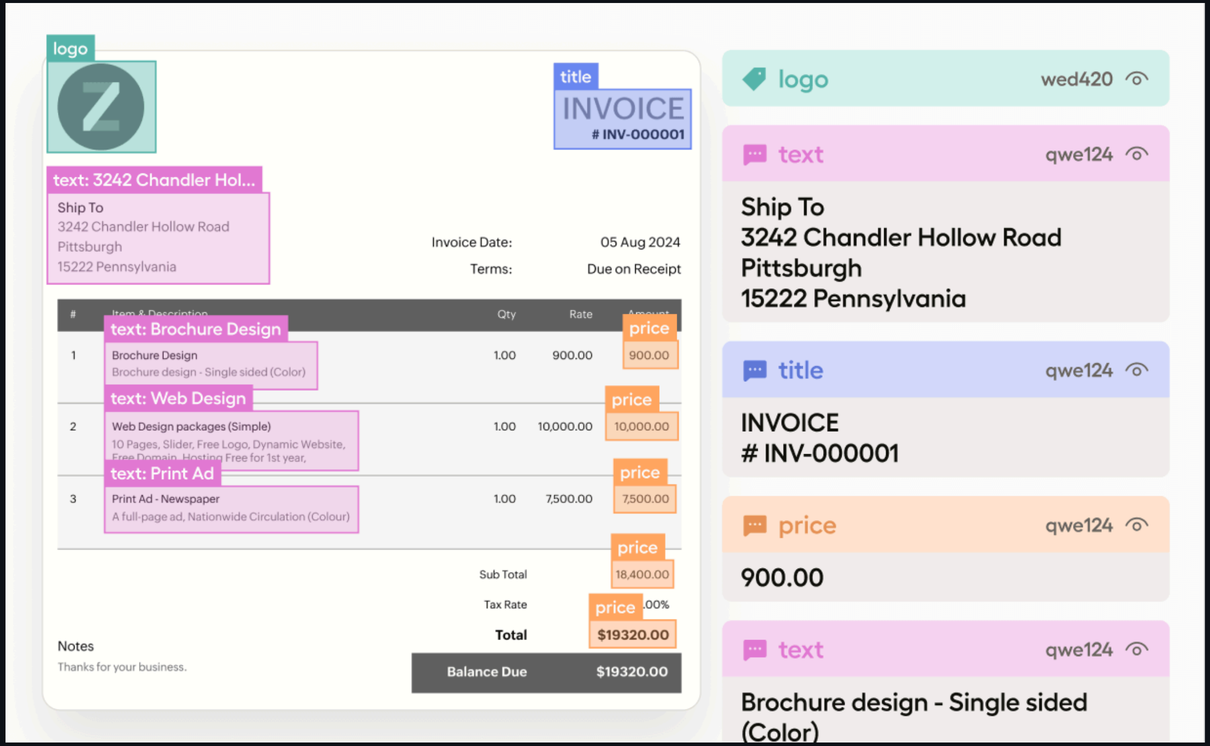
- OCR
- Layout awareness
- Structured data extraction in JSON

Datasets disponibles

- DocVQA · Datset HF
- Openpdf-MultiReceipt-1K · Dataset HF
- Wildreceipt · Dataset HF · Dataset HF/2 · Github
- CORD · Dataset HF · Github · Paper

Etiquetado de datos

Label Studio



Instalar label studio

```
python -m pip install label-studio
```

Arrancar label studio

```
label-studio start
```

👀 OCRs más usados

1. PaddlePaddle · Github · Articulo
2. Tesseract · Github

🚀 Modelos para Inferencia

Modelos multimodales:

- TrOCR (plain) – Best suited for pure OCR tasks, but lacks layout and structural understanding.
- TrOCR/Tesserract + LayoutLM – Combines OCR with layout-aware structured extraction, potentially improving key-value extraction.
- Donut (Swin + Bart) – A fully end-to-end document understanding model that might simplify the pipeline.

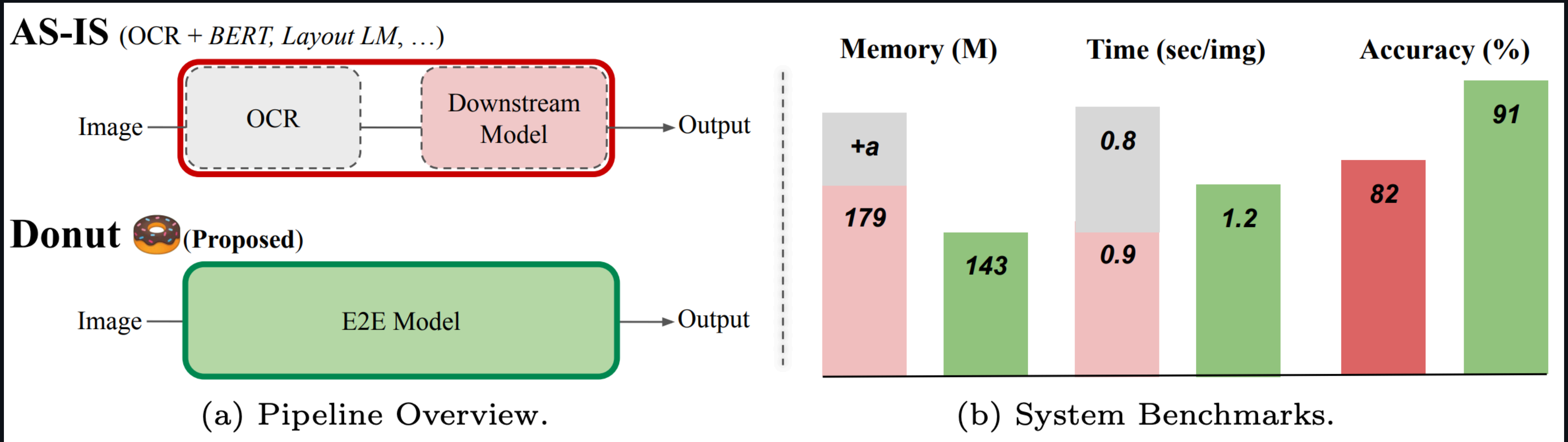
📦 Modelo LayoutML

- Ver demo Tesseract + LayoutML

> 💡 Más info

🍩 Modelo Donut

Comparativa con otros modelos:

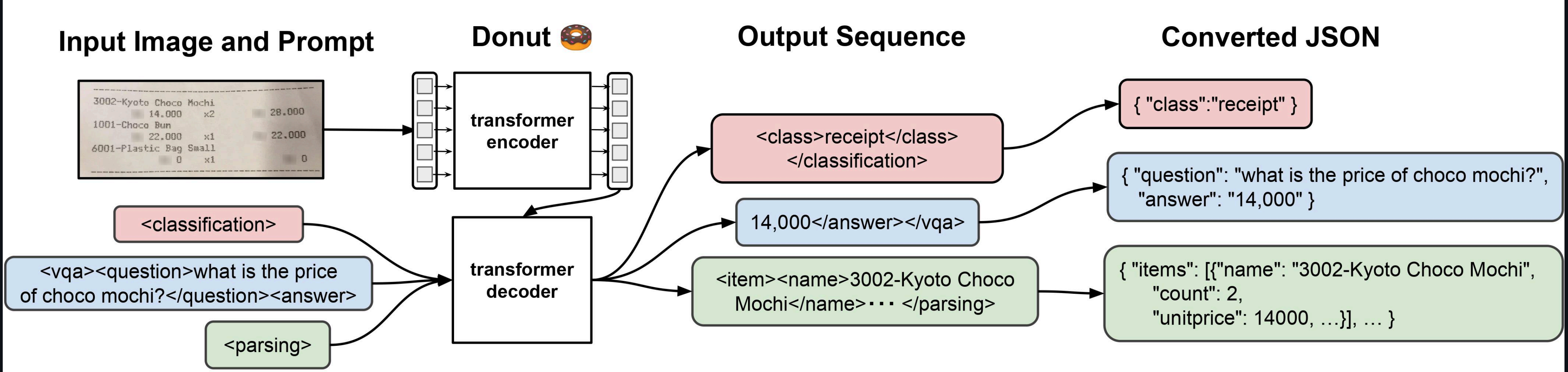


¿Como funciona el modelo Donut?

Donut cuenta con un codificador de visión (Swin) y un decodificador de texto (BART). Swin convierte las imágenes de documentos en embeddings y BART las procesa en secuencias de texto con significado.

> 💡 DonutModel
> 📝 Paper

Arquitectura del modelo:



Ventajas:

- No depende de que el OCR sea en un lenguaje específico
- Se evita el coste computacional de correr el OCR antes
- Al trabajar con imagenes directamente, se evita propagar errores

> 💡 Análisis

Swin Transformer

> 💡 Swin Transformer
> 📝 Paper

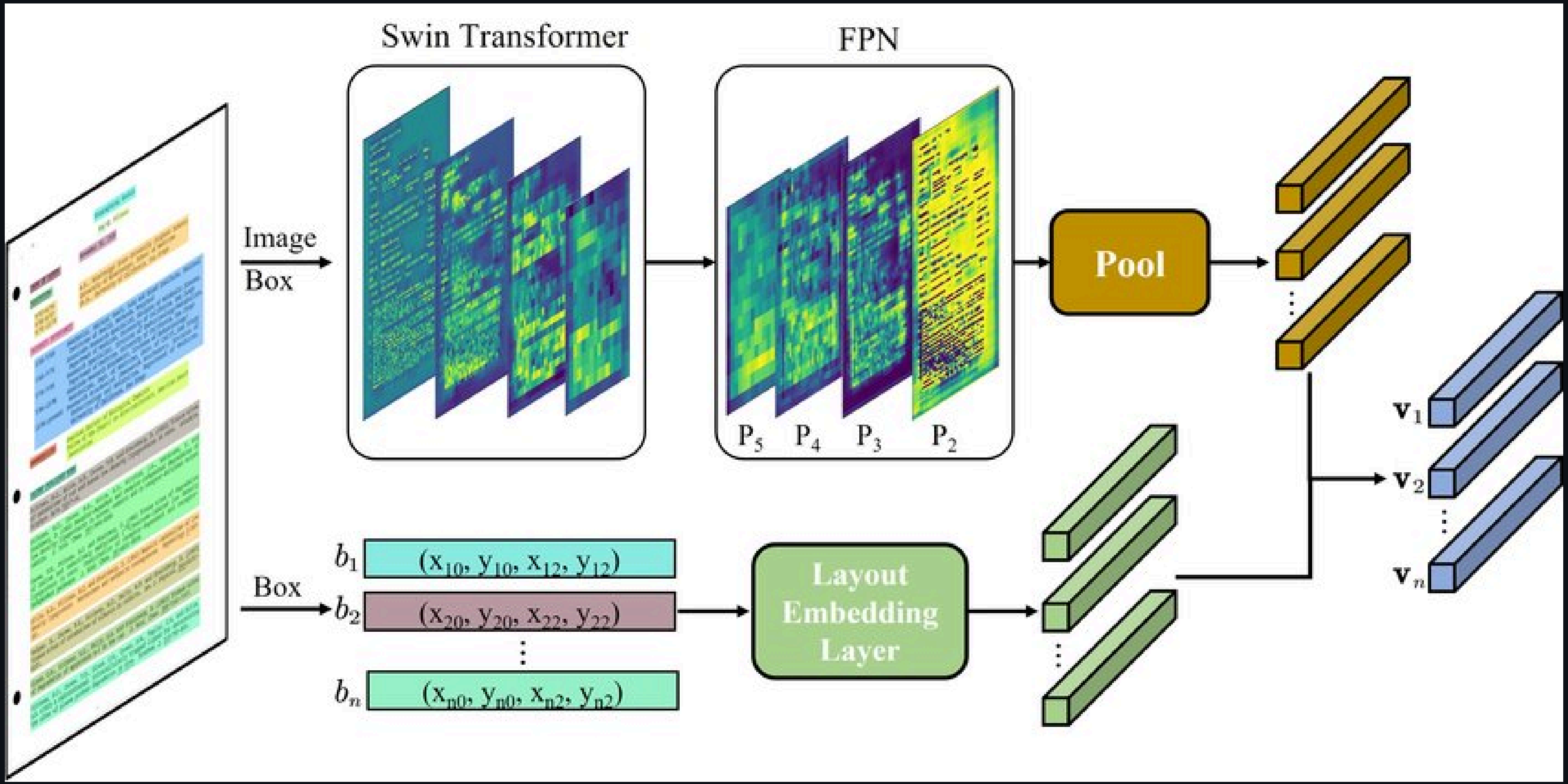
BART LLM

> 💡 BART
> 📝 Paper
> 🔄 Comparativa Bert vs GPT

🅰️ Visual Encoder (Swin):

Cualquier modelo CNN (Convolutional Neural Networks) puede ser usado como encoder. Pero en este caso el Swin Transfomer es el que mejor resultados da en el análisis previo del papar.

1. Splitting the image into non-overlapping patches.
2. Applying the Swin Transformer: This model uses shifted window-based multi-head attention combined with a two-layer MLP (multi-layer perceptron) to process each patch.
3. Merging Patches: Patch merging layers patches the tokens at each stage
4. Passing the final representation to the Textual Decoder.


Swin Transformer
FPN
Image
Box
Pool
P_5 P_4 P_3 P_2
Box
b_1 $(x_{10}, y_{10}, x_{12}, y_{12})$
b_2 $(x_{20}, y_{20}, x_{22}, y_{22})$
b_n $(x_{n0}, y_{n0}, x_{n2}, y_{n2})$
Layout Embedding Layer
$\mathbf{v}_1$
$\mathbf{v}_2$
$\mathbf{v}_n$

🅱️ Textual Decoder (BART):

1. Receives encoded features from the Visual Encoder.
2. Generates a sequence of tokens (y_0, y_1,..y_i,..y_m, where ‘m’ is a hyperparameter).
3. Uses BART (Bidirectional and Auto-Regressive Transformer) as its decoding architecture.

> 💡 Análisis

Fine-tune modelo Donut

- https://www.philschmid.de/fine-tuning-donut

Frontend con Gradio

> Ejemplo DocVQA

- Notebook: gradio + donut

5. Despliegue en producción

Demo usando KServe y Kubeflow