ML System Design Template

Objective: Digitalize hard copy invoices and extract due dates to improve work efficiency.

1. Clarifying Questions

1. **Is extracting only the date sufficient to solve the problem?**
    - No, as some invoices may not directly mention the due date. The system must understand the context around dates to determine the due date.
2. **Can we use the document structure to help find the date?**
    - Yes, invoice structure (headers, footers, sections) can provide clues to identify and extract the relevant dates.
3. **Is there any open-source OCR for Thai?**
    - Yes, tools like Tesseract support Thai and can be used for OCR. -> but it does not work well.
4. **Are the invoices one page?**
    - No, the system must handle multi-page documents if invoices span multiple pages.

1.2 Use Case(s) and Business Goal

Use Case: Automate identifying due dates from hard copy invoices to reduce manual effort and optimize payment schedule.

Business Goal: Increase operational efficiency, reduce late payments, and improve cash flow management.

1.3 Requirements

- **Language Support**: The system must support both Thai and English languages.
- **Input Types**: The system should handle invoices in various formats.
- **Output**: Extracted dates and identified due dates from the invoices.

1.4 Constraints

- **Languages**: English and Thai.
- **Input Size**: Each document size and potential multi-page handling.

1.5 Data Sources:

- **Internal invoices:**
    - **Quantity:** Determine the number of invoices to be processed.
    - **Patterns:** Identify and categorize different invoice patterns.

- **External Data Sources**: Explore any external sources that can help validate due dates or provide additional context.

1.6 Assumptions

- The invoices are mostly consistent in format within categories.
- Dates are clearly legible for OCR tools.
- There is some contextual text around dates that can help identify the due date.

1.7 ML Formulation

1. **Text Extraction (OCR)**:
    - Use tools like Tesseract for OCR to extract text from scanned images of invoices.
2. **Date Identification**:
    - Implement Named Entity Recognition (NER) to locate all dates in the extracted text.
    - Use regex patterns to identify date formats.
3. **Due Date Classification**:
    - Train a classifier to identify context around dates to determine if they indicate a due date.
    - Features might include keywords (e.g., "due", "pay within", "net"), relative position in the document, and invoice layout.

2. Metrics

2.1 Offline Metrics

These metrics evaluate the model's performance during development and before deployment.

1. **Data Extraction Performance**:
    - **Precision**: The proportion of correctly identified dates out of all identified dates.
    - **Recall**: The proportion of correctly identified dates from all actual dates in the documents.
    - **F1 Score**: The harmonic mean of precision and recall.
2. **Due Date Classification Performance**:
    - **Accuracy**: The proportion of correctly classified due dates out of all instances.
    - **Precision**: The proportion of true positive due dates out of all dates classified as due dates.
    - **Recall**: The proportion of true positive due dates out of all actual due dates.
    - **F1 Score**: The harmonic mean of precision and recall.

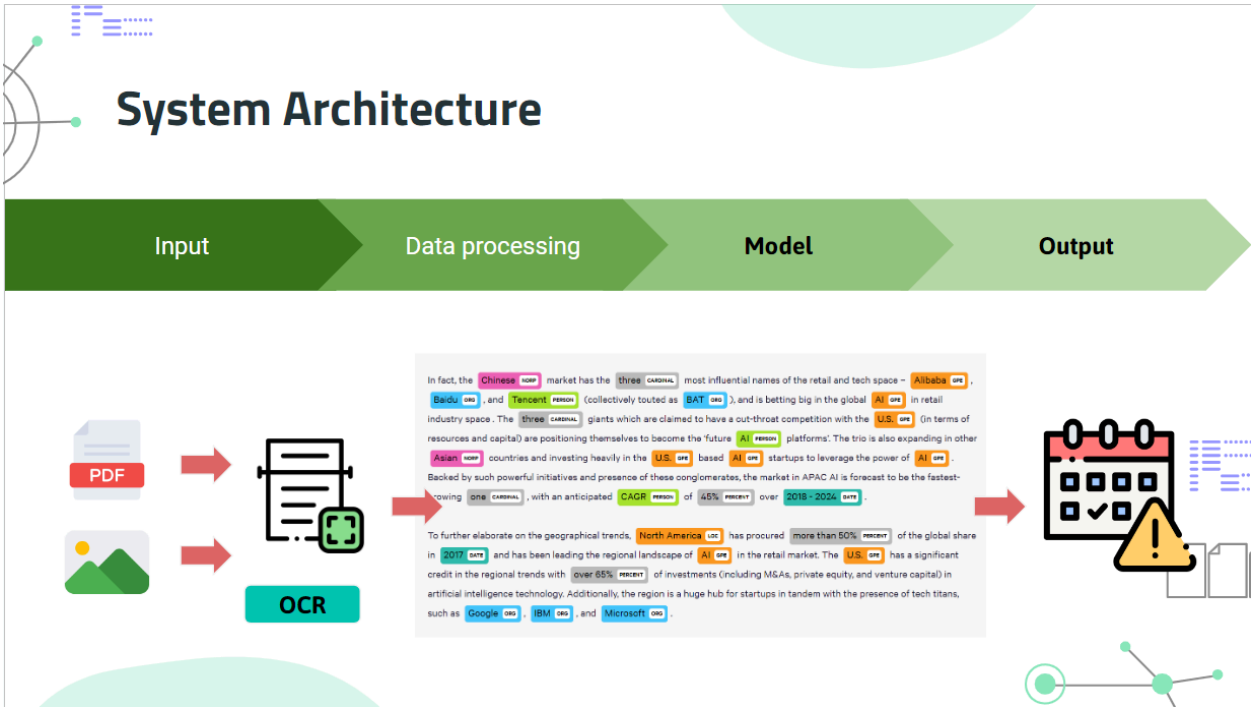
2.2 Online Metrics

These metrics are used to evaluate the performance of your system after deployment in a real-world environment.

1. **Processing Time**:
    - **Average Processing Time**: The average time to process and extract due dates from an invoice.
    - **Latency**: The time delay between submitting an invoice and the availability of the extracted due date.
2. **Error Rates**:
    - **False Positive Rate**: The proportion of non-due dates incorrectly identified as due dates.
    - **False Negative Rate**: The proportion of due dates missed by the system.
3. **User Feedback and Satisfaction**:
    - **User Satisfaction Score**: Collected through surveys or feedback forms from users interacting with the system.
    - **Error Reports**: The number and type of errors reported by users.
4. **System Utilization**:
    - **Usage Frequency**: The number of invoices the system processes over a period.
    - **Adoption Rate**: The proportion of potential users actively using the system.
5. **Financial Impact**:
    - **Reduction in Late Payments**: The number of late payments decreased due to the timely identification of due dates.
    - **Cost Savings**: The amount of money saved by automating the due date extraction process compared to manual processing.
6. **Scalability**:

- **Throughput**: The number of invoices processed per unit time.
- **System Performance**: The system's performance under varying loads ensures it can handle peak volumes efficiently.

3. Architectural Components (High-level architecture)



4. Data Collection and Preparation

Data Needs:

- Collect invoices with various patterns (formats, languages, and structures).
- Ensure the dataset includes various invoice types to improve model generalizability.

Data Sources:

- ~~Internal company invoices~~. -> I am not using this because it needs a manual label.
- Various formats on the internet.

Data Storage:

- Store the collected data locally on a secure server.
- Use a structured directory to organize invoices by type, format, and language.

ML Data Types:

- **Text**: Extracted text from invoices.

Labeling: (for future work)

- Annotate the collected invoices with the correct due dates.
- Use LabelImg or custom annotation tools to mark dates and context.

5. Feature Engineering

Feature Selection:

- **Text Features**: Extracted text around date field keywords indicating due dates (e.g., "due," "pay within").
- **Positional Features**: Position of the date within the document (e.g., header, footer, body).
- **Structural Features**: Sections of the invoice (e.g., invoice header, itemized list).

Feature Representation:

- **Text**: Use techniques like TF-IDF, and word embeddings (e.g., Word2Vec, BERT).
- **Positional**: Encode the position of text as numerical features.
- **Structural**: Use one-hot encoding for categorical sections of the document.

Feature Preprocessing:

- Normalize text data (e.g., lowercasing, removing punctuation).
- Standardize numerical features (e.g., positional information).
- Handle missing values appropriately.

6. Model Development and Offline Evaluation

Model Selection:

- Choose appropriate models for OCR (e.g., Tesseract).
- For date extraction and classification, consider models such as:
    - **NER**: SpaCy, BERT-based models.
    - **Classification**: Random Forest, SVM, Neural Networks.

Dataset Construction:

- Split the data into training, validation, and test sets.
- Ensure a balanced representation of different invoice patterns in each set.

Model Training:

- Train the OCR model to recognize and extract text from invoices.
- Train the NER model to identify date entities.
- Train the classification model to determine due dates from extracted dates.

Model Evaluation and Hyperparameter Tuning:

- Evaluate model performance using metrics like precision, recall, and F1 score.
- Use grid search or random search for hyperparameter tuning.
- Perform cross-validation to ensure model generalization.

Iterations:

- Iterate on model training and evaluation by refining features and tuning hyperparameters.
- Incorporate feedback from domain experts to improve labeling and feature engineering.

7. Prediction Service

Service Design:

- Develop an API for the prediction service.
- The service should accept invoices as input, process them, and return the due date.

Implementation:

- Use frameworks like FastAPI or Flask to build the API.
- Integrate the OCR, NER, and classification models into the service pipeline.

Scalability:

- Ensure the service can handle multiple requests simultaneously.
- Use containerization (Docker) and orchestration (Kubernetes) for scalability.

8. Online Evaluation and Deployment

Online Evaluation:

- Monitor the prediction service's performance in a live environment.
- Collect feedback from users to identify any issues or areas for improvement.
- Track online metrics like processing time, error rates, and user satisfaction.

Deployment:

- Deploy the prediction service to a production environment.
- Use CI/CD pipelines to automate deployment and updates.
- Implement logging and monitoring to track system performance and detect issues.

Continuous Improvement:

- Regularly update the model with new data to maintain accuracy.
- Continuously monitor and refine the service based on user feedback and performance metrics.