Roster Email Parser and Extractor
A Technical Overview

Nishant Padmanabhan

Spetember 2025

1 Overview

This document provides a comprehensive explanation of `extractor.py`, a Python script designed to automate the extraction of healthcare provider information from EML (`.eml`) email files and populate it into a standardized Excel template.

The script leverages a powerful, locally-run Large Language Model (LLM) from the Hugging Face `transformers` library to intelligently parse unstructured text from emails and extract structured data. This approach is highly flexible and robust, capable of handling variations in email formatting that would break traditional rule-based parsers.

The core goal of this tool is to drastically reduce manual data entry, improve accuracy, and accelerate the administrative workflows associated with managing healthcare provider rosters.

2 How It Works: The Technical Workflow

The script operates through a sequential pipeline, moving from data ingestion to processing and finally to output generation.

2.1 Step 1: Loading and Parsing EML Files

The script begins by reading an `.eml` file. It processes both plain text and HTML parts of the email.

- `load_eml_text(path)`: This function opens an `.eml` file, parses its contents, and extracts all readable text. If the email contains HTML, it uses the `BeautifulSoup` library to strip out the HTML tags, ensuring only the clean, human-readable text is kept.

2.2 Step 2: Preparing the AI Model

To understand the text, the script uses a state-of-the-art language model.

- `MODEL_NAME = "microsoft/Phi-3-mini-4k-instruct"`: This configuration variable specifies a compact but powerful model from Microsoft, chosen for its balance of performance and relatively low computational requirements.
- `get_llm_pipe(model_name)`: This function initializes the `transformers` pipeline. It first attempts to load the model onto a GPU for faster processing (`device_map="auto"`). If a GPU is not available, it gracefully falls back to using the CPU.

2.3 Step 3: Dynamic Prompting

This is the core of the extraction logic. We instruct the AI on what we need in plain English and provide a structured format for its response.

- `make_prompt(email_text)`: This function constructs a detailed set of instructions for the LLM. It defines a JSON `schema` listing every piece of information we want to extract (e.g., "provider_name", "Provider NPI", "Phone number", "Address"). This schema tells the model the exact key names, expected value formats, and the fallback value (`"Information not found"`). The full email text is then embedded into this prompt.

2.4 Step 4: LLM Extraction

The prompt is sent to the LLM, which reads the email and attempts to fill in the JSON schema.

- `extract_with_llm(pipe, email_text)`: This function executes the AI model and performs cleanup on the generated text to ensure it can be parsed as valid JSON, even attempting to fix common errors.

2.5 Step 5: Populating the Excel Template

Once the data is extracted into a JSON object, it is placed into the correct columns of the Excel template.

- `TEMPLATE_TO_KEY`: This dictionary maps the Excel column headers (e.g., "Provider Name") to the JSON keys (e.g., "provider_name").
- `process_eml_files(...)`: This main loop reads headers from the template, processes each EML file, and builds a new row in the exact order specified by the template.
- `append_row_to_template(...)`: This function uses the `openpyxl` library to append the newly extracted row to the Excel file and save the result.

2.6 Step 6: Command-Line Interface (CLI)

The script is packaged with a user-friendly CLI using Python's `argparse` library, allowing a user to run the process from the terminal.

3 Real-World Impact and Benefits

3.1 Increased Productivity of Medical Practitioners

- **Reduced Administrative Burden:** By automating roster updates, the tool frees up valuable time for clinical staff, allowing them to focus on patient care.
- **Faster Onboarding:** Automating this process ensures that new practitioners can be integrated into the system swiftly and efficiently.

3.2 Convenience

- **Automated Workflow:** Transforms a tedious manual process into a single command, capable of processing hundreds of emails in batches.
- **Error Reduction:** The AI model, guided by a strict schema, provides consistent and accurate data extraction, improving data quality and reliability over manual entry.

3.3 Affordability

- **Open-Source Technology:** The solution is built on free, open-source libraries and AI models, eliminating licensing fees.
- **Reduced Labor Costs:** Automation significantly reduces the operational expenses associated with manual data entry.
- **Accessible Hardware:** The script is designed to run on standard computer hardware, with a fallback to CPU if a GPU is not present.

4 How to Use the Script

4.1 Installation

1. Ensure you have Python 3.8+ installed.
2. Install the necessary libraries via pip:

```
pip install transformers torch accelerate openpyxl beautifulsoup4 lxml
```

4.2 Running from the Command Line

Execute the script from your terminal, providing the required paths.

```
python extractor.py /path/to/eml_files /path/to/template.xlsx /path/to/output.xlsx
```

Arguments:

`eml_input` Path to a single `.eml` file or a folder of `.eml` files.

`template` Path to the standardized Excel template file.

`output` Path where the final, populated Excel file will be saved.

`--batch [N]` (Optional) Process `N` emails at a time.

`--verbose` (Optional) Enable detailed logging for debugging.