**Suppl 3.** The programming information

This document provides a description of the key Python scripts used in the development and execution of the LLM-based medication extraction and DILI risk linkage pipeline evaluated in this study. The code is organized into configuration, utility functions, core service modules (extraction, DILI connection), and processing scripts.

**1. Configuration File (config.py)**

* **Purpose:** Centralizes all configuration settings for the application, promoting modularity and ease of modification.
* **Key Components / Functionality:**
	+ Defines a Config class.
	+ Loads sensitive information (like API keys) and file paths from environment variables or a .env file using python-dotenv.
	+ Stores constants such as API keys (GROQ\_API\_KEY), the target LLM model name (MODEL), supported LLM models (SUPPORTED\_MODELS), file paths for input/output (COMBINED\_FILE, MIMIC\_INPUT\_FILE, MIMIC\_OUTPUT\_FILE, LOG\_FILE), and processing parameters (fuzzy matching threshold, API retry logic, delays).
	+ Initializes basic logging configuration for the application (level, format, file/stream handlers).
* **Inputs:**
	+ Environment variables (primarily GROQ\_API\_KEY).
	+ .env file located in the project base directory (optional fallback).
* **Outputs:** Provides configuration variables imported and used by other modules. Sets up application-wide logging.
* **Key Dependencies:** os, dotenv, logging.

**2. Utility Functions (app/services/utils.py)**

* **Purpose:** Contains general-purpose helper functions used across different modules, primarily for text processing related to drug names.
* **Key Components / Functionality:**
	+ contains\_drug\_names(text): Implements a simple regular expression-based heuristic check to quickly determine if a string likely contains drug-like names (primarily capitalized words, excluding common non-drug terms). Used potentially for pre-filtering or simple validation.
	+ clean\_and\_split\_drug\_names(drug\_string, delimiters=...): Takes a string potentially containing multiple drug names (e.g., from concatenated fields), splits them based on defined delimiters (commas, semicolons, slashes), cleans whitespace, and returns a list of individual drug names.
* **Inputs:** Text strings (text, drug\_string).
* **Outputs:** Boolean (contains\_drug\_names), List of strings (clean\_and\_split\_drug\_names).
* **Key Dependencies:** re, logging.

**3. Medication Extraction Service (app/services/medication\_extractor.py)**

* **Purpose:** Handles the core logic for interacting with the Large Language Model (LLM) via the Groq API to extract medication information from unstructured text. Includes logic for handling long text inputs via chunking.
* **Key Components / Functionality:**
	+ **Groq Client Initialization:** Initializes the Groq API client using the API key from config.py.
	+ **Tokenizer Initialization:** Loads the tiktoken tokenizer (cl100k\_base) for accurately counting tokens and enabling text chunking.
	+ **System Prompt (SYSTEM\_PROMPT):** Defines the detailed instructions given to the LLM for the extraction task, specifying the desired fields (name, normalized\_name, dosage, frequency, date) and the required JSON output format.
	+ **Chunking Logic (\_split\_text\_into\_chunks):** Splits input text exceeding the model's context window (minus buffers) into smaller, overlapping chunks based on token counts.
	+ **API Interaction (\_call\_groq\_for\_chunk):** Sends individual text chunks (or the whole text if short enough) to the Groq API chat completion endpoint. Handles JSON response parsing, API errors (rate limits, connection errors, validation errors), and implements retry logic with delays (Config.MAX\_RETRIES, Config.RETRY\_DELAY, Config.API\_CALL\_DELAY).
	+ **Result Deduplication (\_deduplicate\_medications):** Merges results from multiple chunks (if used) and removes duplicate medication entries based on a composite key (normalized name, dosage, frequency). *(Note: This is internal deduplication within the extractor; the manuscript evaluation used manual deduplication)*.
	+ **Main Function (extract\_medications\_from\_API):** Orchestrates the process: checks inputs, determines if chunking is needed, calls the API (potentially multiple times for chunks), aggregates results, performs deduplication, and returns the final list.
* **Inputs:** user\_input (clinical text string), model name (optional, defaults to Config.MODEL), Config settings (API key, model, delays, retries, buffer sizes), System Prompt.
* **Outputs:** A list of dictionaries, where each dictionary represents an extracted medication (['name':..., 'normalized\_name':..., ...}]), or None if a critical error occurs (e.g., API key missing, tokenizer unavailable). Returns an empty list [] if no medications are found.
* **Key Dependencies:** groq, tiktoken, json, logging, re, time, config.

**4. DILI Connector Service (app/services/dili\_connector.py)**

* **Purpose:** Links extracted medication names (specifically normalized\_name) to potential DILI risk information stored in a local curated Excel file.
* **Key Components / Functionality:**
	+ **DILI Data Loading (\_load\_dili\_data):** Loads the DILI information from the specified Excel file (Config.COMBINED\_FILE) into a pandas DataFrame (\_dili\_df). Creates a pre-processed list of unique, lowercased drug names (\_dili\_df\_choices) for efficient fuzzy matching. Includes caching to avoid reloading the file repeatedly. Handles file-not-found and other loading errors.
	+ **Main Lookup Function (get\_dili\_risk\_from\_excel):** Takes a list of medication dictionaries (output from medication\_extractor.py). For each medication, it attempts to find the best match for its normalized\_name (or fallback name) within the \_dili\_df\_choices using fuzzy string matching (fuzzywuzzy.process.extractOne, fuzz.ratio) based on the Config.FUZZY\_MATCH\_THRESHOLD.
	+ **Data Enrichment:** If a sufficiently close match is found, it retrieves the corresponding DILIrank likelihood and LiverTox score from the cached DataFrame (\_dili\_df) and adds them (DILIrank\_Likelihood, LiverTox\_LikelihoodScore) along with match details (DILI\_Match\_Details) to the medication dictionary. Handles cases where no match is found or DILI data is unavailable.
* **Inputs:** List of medication dictionaries (must contain 'normalized\_name' or 'name'), Config settings (COMBINED\_FILE path, FUZZY\_MATCH\_THRESHOLD), the Combined.xlsx file itself.
* **Outputs:** The input list of medication dictionaries, updated with added keys for DILI information (DILIrank\_Likelihood, LiverTox\_LikelihoodScore, DILI\_Match\_Details). Values are 'Unknown' or indicate lookup failure if no match is found or data is missing.
* **Key Dependencies:** pandas, fuzzywuzzy, logging, os, time, config.

**5. MIMIC Processing Script (mimic\_processing/process\_mimic.py)**

* **Purpose:** Orchestrates the end-to-end processing of a batch of MIMIC-IV discharge summaries from an input CSV file. It reads notes, calls the medication extractor, calls the DILI connector, and saves the structured results (one row per medication or note status) to an output CSV file. This script was used to generate the data subsequently used for the manual validation reported in this manuscript.
* **Key Components / Functionality:**
	+ **Configuration & Constants:** Defines input/output CSV paths (INPUT\_CSV\_PATH, OUTPUT\_CSV\_PATH from Config) and the structure of the output CSV (OUTPUT\_COLS).
	+ **Main Processing Function (process\_mimic\_discharge\_summaries):**
		- Reads the input CSV containing MIMIC notes (requires columns like 'note\_id', 'text').
		- Iterates through each row (each discharge summary).
		- For each note, calls extract\_medications\_from\_API to get the list of extracted medications.
		- If medications are found, calls get\_dili\_risk\_from\_excel to append DILI risk information.
		- Handles different scenarios: successful extraction, no medications found by LLM, LLM extraction errors.
		- Transforms the data: Creates one output row for each extracted medication, including note identifiers, LLM-extracted details, DILI linkage results, and placeholder columns for subsequent manual validation (manual\_\* fields). Creates status rows for notes with errors or no medications found.
		- Compiles results into a pandas DataFrame.
		- Writes the final DataFrame to the specified output CSV file.
	+ **Execution Block (if \_\_name\_\_ == "\_\_main\_\_":)**: Handles script execution, sets up logging, calls the main processing function, and reports completion status.
* **Inputs:** Input CSV file containing MIMIC discharge summaries (Config.MIMIC\_INPUT\_FILE), Configuration settings (Config), outputs from medication\_extractor and dili\_connector.
* **Outputs:** Output CSV file (Config.MIMIC\_OUTPUT\_FILE) with detailed, structured results (one row per extracted medication or note status), suitable for analysis and validation. Log file entries documenting the process.
* **Key Dependencies:** pandas, logging, json, tqdm, time, os, sys, config,
app.services.medication\_extractor, app.services.dili\_connector.