Agentic Text Normalization

This report details the design, implementation and evaluation of the Agentic Text Normalization System. The system is designed to clean and standardize raw text data containing music composition writers, a task complicated by noisy, unstructured and redundant information. The solution is implemented as a single Python script and employs a modular, agent-based architecture, with adaptive complexity analysis, strategy selection and specialized processing pipelines to achieve a robust and scalable approach to text normalization.

1. Overall Approach and Architecture

Core Philosophy

The core philosophy of this solution is that not all normalization tasks are equally difficult. A simple input requires less processing than a complex one, so a one-size-fits-all approach is often inaccurate. To address this, the system uses a dynamic, complexity-driven agentic workflow to overcome the inherent variability by:

- Adaptive Processing, where different complexity levels require different processing strategies,
- Modular Design, where each agent has a specific responsibility, enabling easy maintenance and extension,
- Degradation, adopting fallback mechanisms to ensure robustness when dependencies are unavailable,
- Orchestration, where a central orchestrator coordinates agent interactions based on input characteristics

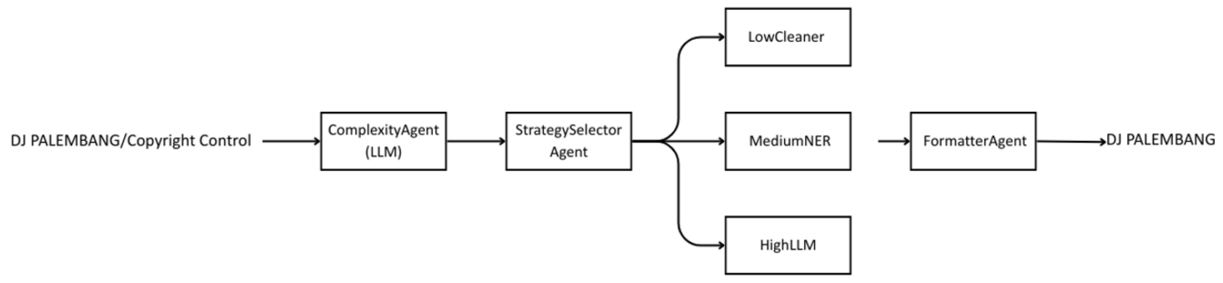
Architecture

The system implements six agentic-wised Python classes, each with a specific well-defined responsibility. This mimics an agentic framework like LangChain or CrewAI. The agents in the system are:

- **Orchestrator Agent:** The main controller that manages the entire workflow. It invokes other agents in the correct sequence and passes data between them.
- **Complexity Agent:** Its sole job is to classify the input text's difficulty. It provides a complexity label (low, medium, high), for the strategy selector.
- **Strategy Selector Agent:** A simple router that maps the complexity label to a specific extraction strategy, for example if complexity is high-> high llm agent.

- **Low Complexity Cleaner Agent:** A lightweight tool that uses a pre-compiled list of regular expressions to strip out common non-writer entities like publishers and legal terms. It is fast and precise for simple, known patterns.
- **Medium Complexity NER Agent:** This agent represents a classic NLP approach. It leverages a pre-trained spaCy model to perform Named Entity Recognition to find entities. It also includes regex-based fallbacks to identify name-like patterns and common stage name conventions if NER fails.
- **High Complexity LLM Agent:** This agent uses a Hugging Face transformers model, google/flan-t5-large, with a few-shot prompt. This allows it to handle patterns, nested information and separators that regex or standard NER would struggle with. For robustness, it falls back to the MediumComplexityNERAgent if the LLM fails.
- **Formatter Agent:** This final agent is responsible for presentation. It focuses on standardizing the output by fixing comma inversions, correcting capitalization, ensuring consistent spacing and joining the final list of names with a "/" separator.

A key feature of the architecture is that, if the transformers library or spaCy model is not available, the agents automatically fall back to their next-best heuristic or regex-based methods. This ensures the system remains functional even with missing dependencies, enhancing its robustness.



2. Techniques and Tools

The solution integrates a hybrid of traditional and modern AI techniques to create a comprehensive system.

- **Rule-Based Processing:** The *re* module is used extensively for:
  1. Splitting raw strings by multiple delimiters.
  2. Identifying and removing a curated list of known non-writer patterns.

- **Named Entity Recognition:** The *spaCy* library, specifically the *en_core_web_sm* model, is used to identify PERSON entities in the medium-complexity path.

- **Large Language Models:** The Hugging Face transformers library is used to run the *google/flan-t5-large* model. This model is leveraged for two key tasks:

    1. Complexity Analysis with a zero-shot classification prompt.
    2. High-Complexity Agent with a few-shot prompt.

3. Challenges and Limitations

While the system is robust, it has several challenges and limitations:

- **Model Dependency:** The highest accuracy is achieved when spaCy and transformers models are available. While the system degrades to heuristics, performance on medium-to-high complexity inputs may be reduced.
- **LLM Reliability:** The specific llm is a good compromise between performance and size, but it can occasionally fail to follow instructions or may hallucinate. A larger, more powerful model or a fine-tuned model would be necessary for higher reliability.
- **Ambiguity of Names:** The distinction between a unique artist name and a corporate entity can be difficult. The current system relies on NER and regex patterns, but these can fail on unseen edge cases.
- **Static Non-Writer List:** The list of non-writer entities is manually curated. It is effective for the provided dataset, but would need continuous updates to handle new entities.

4. Future Improvements

The modular design provides a clear path for scaling and adaptation. To significantly boost accuracy, the spaCy NER model could be fine-tuned on the provided dataset. This would improve its ability to recognize artist-specific name formats and distinguish them from other entities. Similarly, a smaller text-generation model could be fine-tuned specifically for the normalization task, likely outperforming the general-purpose flan-t5-large model.

Examples can be found in the *results_eval.csv,* with >70% exact match accuracy.