COLUMBIA UNIVERSITY
School of Professional Studies

Multi-Agent Text Processing System

Team 3:

Hao Jiang, Jiayi Jin, Ruicheng Deng, Yilu Shen, Zhenxiao Luo, Huiting Chen

Table of Contents

Project Motivation & Overview

❖ Motivation

- Text analysis tasks are growing in complexity
- Single-agent models can fall short for nuanced tasks
- Multi-agent systems offer modular, collaborative processing

❖ Our Objectives

- Design a **multi-agent text-processing system**
- Define a clear, specialized role per agent
- Demonstrate the system via an intuitive Streamlit app

❖ Why It Matters

- Explore agent coordination for end-to-end workflows
- Apply the system in real-world applications
- Build a reusable, extensible architecture for future use cases

Agent 1: Summarizer Agent

- **Function**
  - Generate concise summaries of input text
  - Preserve essential meaning and tone

- **Implementation**
  - Build using the OpenAI API
  - Design prompt engineering for brevity and clarity
  - Include tokenization and cost optimization

- **Workflow Role**
  - Act as the first processing step in the agent pipeline
  - Enable smoother downstream processing for critique and tone adjustment

- **Key Contribution**
  - Improve readability and reduce cognitive load for users

Agent 2: Tone Adjuster Agent

❖ Function
- Modify the tone of the summarized text
- Support styles (e.g., formal, friendly, persuasive)

❖ Implementation
- Use OpenAI with customized prompts
- Preserve meaning while shifting tone
- Handle edge cases and maintain fluency

❖ Workflow Role
- Follow with Summarizer Agent
- Enhance user control and text personalization

❖ Key Contribution
- Enable adaptable communication for different contexts and audiences

Agent 3: Sentiment Agent

- **Function**
  - Classify emotional tone as positive, negative, or neutral
  - Provide quick emotional insight into text

- **Implementation**
  - Use OpenAI with sentiment-specific prompts
  - Test on varied tone-adjusted inputs for accuracy

- **Workflow Role**
  - Help users understand emotional impact

- **Key Contribution**
  - Add emotional intelligence to the app
  - Support informed revisions and audience-aware communication

Agent 4: Translation Agent

❖ **Function**
- Translate final output into user-selected languages
- Maintain meaning, tone, and sentiment across translations

❖ **Implementation**
- Use OpenAI API for language conversion
- Support multiple target languages
- Validate input and handle translation failures

❖ **Workflow Role**
- Enable content localization and accessibility

❖ **Key Contribution**
- Expand reach to multilingual users
- Showcase adaptability of the agent-based architecture

Agent 5: Style Enhancer Agent

❖ **Function**
- Improve grammar, clarity, and sentence flow
- Enhance stylistic consistency without altering meaning

❖ **Implementation**
- Use OpenAI API to restructure and refine text
- Target redundancy and awkward phrasing

❖ **Workflow Role**
- Polish text before readability scoring
- Refine final output for fluency and elegance

❖ **Key Contribution**
- Elevate professionalism and readability
- Prepare text for external publication or presentation

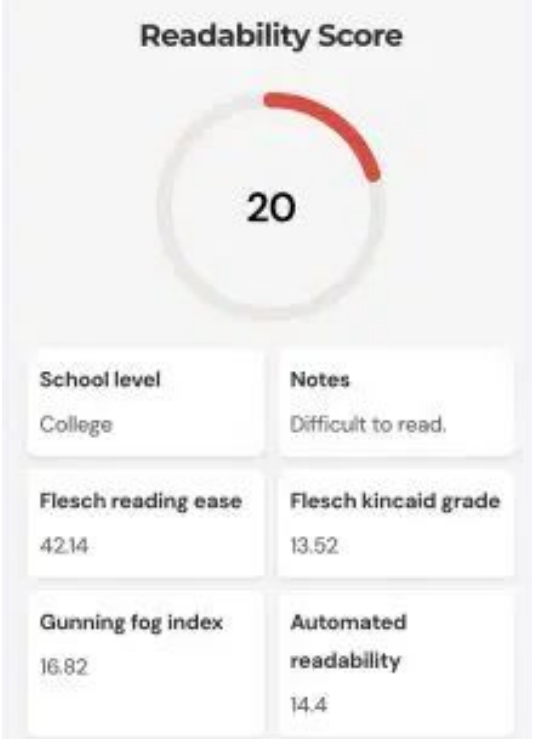
Agent 6: Readability Scorer Agent

❖ Function
- Analyze the final text for readability
- Return scores based on standard metrics

❖ Implementation
- Use OpenAI API to get readability score
- Output flesch reading ease and flesch-kincaid grade

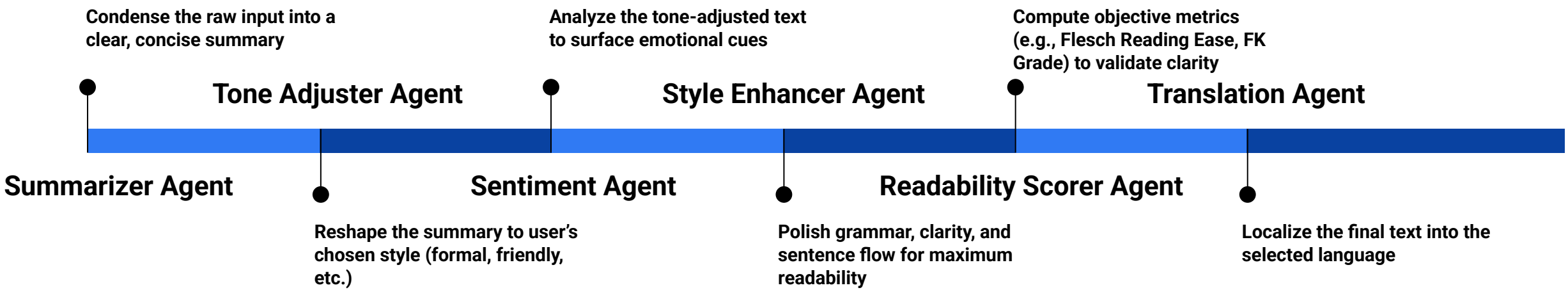
❖ Workflow Role
- Final evaluation step of the pipeline
- Validate effectiveness of prior agents

❖ Key Contribution
- Provide objective clarity feedback
- Support audience-appropriate communication



Workflow Integration

Before passing text into the pipeline, users should enter/upload text via the Streamlit UI.
Users can trigger any individual agent or click “Run All Agents” to execute the full workflow in one click!



Streamlit: Fast Prototyping for AI Applications

Streamlit

What is Streamlit?

A Python framework for building interactive web apps quickly

Designed for data science, AI, and machine learning workflows

Streamlit Used in the Project

Built a multi-agent text processing system

Used Streamlit to:

- Accept text input st.text_area()
- Trigger six different agents with buttons st.button()
- Display processed results st.write(), st.success()

Why Streamlit?

Minimal coding required, fast development

Smooth, clean UI perfect for prototyping AI systems

Easy to deploy for internal testing and demonstrations

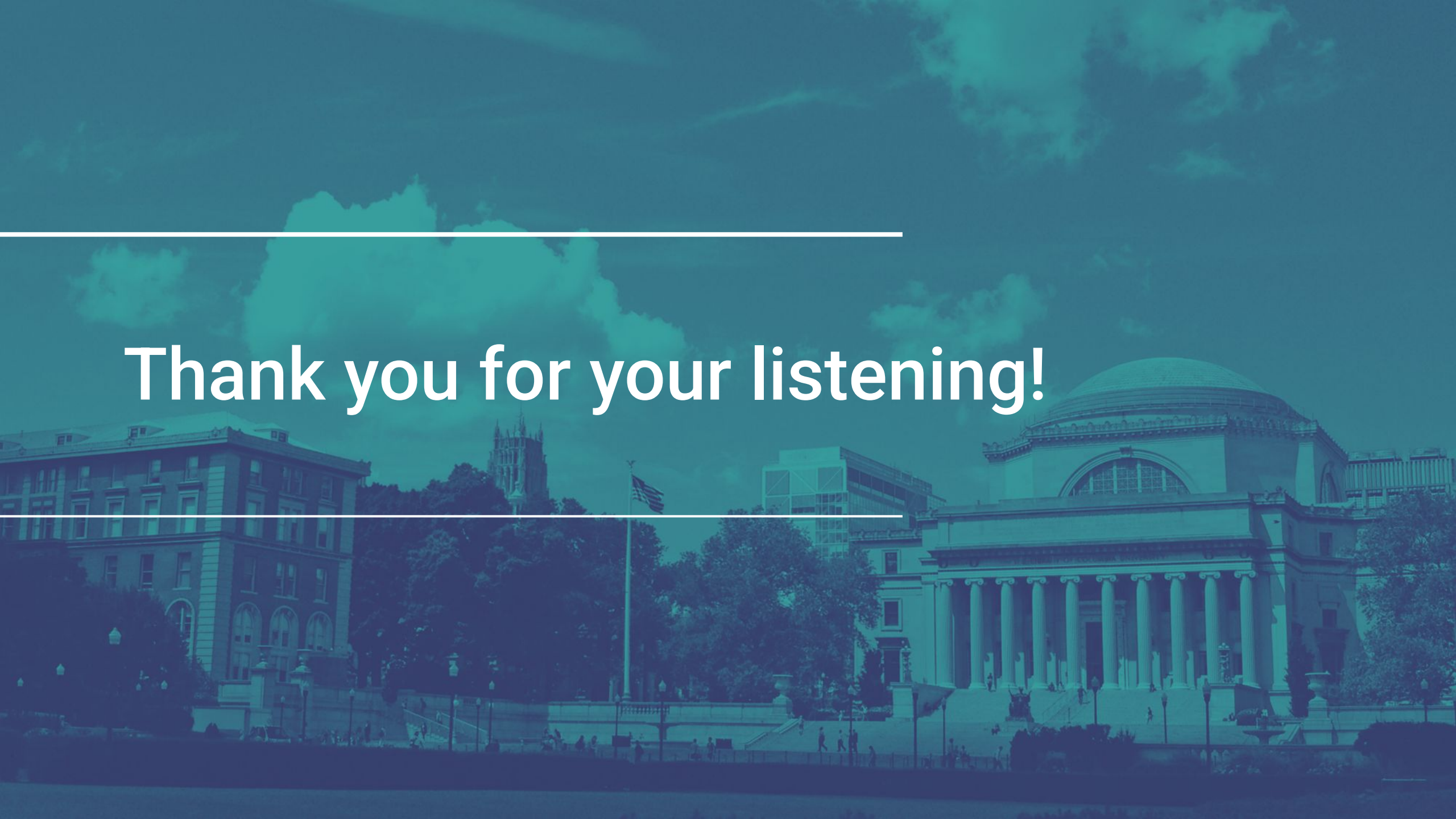
Live Demonstration

Project Challenges & Solutions

- **Standardize Prompts**: define a shared JSON schema and validator to parse outputs and chain agents reliably
- **Fallback OCR**: auto-switch to Tesseract with progress spinners in Streamlit when PyPDF2 failed
- **Throttle Management**: coach repeated inputs and batch prompts to limit API calls and avoid rate limits
- **Error Handling**: wrap each `chat.completions.create()` call in try/except to handle failures carefully
- **UI Control**: provide individual agent buttons + “Run All” option with expandable result panels
- **Document Local vs. LLM Scoring**: compare deterministic vs AI-driven results between LLM agent and agent using local formula

Insights & Lessons

- **Modular Design**: Simplify adding or swapping agents without rewriting core logic
- **Cost–Performance Trade-Off**: Balance LLM accuracy against local computation speed and cost
- **Prompt Engineering**: Learn that precise, consistent prompts dramatically improve output quality
- **User-Centric UI**: Confirm that clear controls and feedback drive better user engagement
- **Maintainability**: Note that structured error handling and logging ease debugging and future updates


Thank you for your listening!