Cairo University
Faculty of Engineering
Department of Computer Engineering

# Aigile

The Intelligent Automation of the Scrum Workflow

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

**Presented by**

Mostafa Hani

**Supervised by**

Prof. Ahmed Darwish

# Abstract

This report outlines my key contributions to the Aigile project, which aims to improve Scrum processes in agile software development using artificial intelligence. I developed two main components: a system for automatic task assignment and a tool for summarizing sprint review meetings. The task assignment system uses an AdaBoost model with features to detect changes in data patterns and prioritize recent information, achieving a top accuracy of 79.74%. This system adjusts to shifting team needs, ensuring tasks are assigned effectively. The summarization tool employs TF-IDF and TextRank methods, enhanced to focus on key speakers and agile-specific terms, to create clear and useful summaries of meeting notes. These summaries were rated highly by agile experts for their relevance. Both components are built into Jira as plugins and a simple desktop application, making task assignment and meeting documentation faster and more efficient. This work shows how AI can improve agile project management by providing practical, adaptable solutions.

# Table Of Contents

# 1. Introduction

This report details my individual contributions to the Aigile project, a system designed to automate key aspects of the Scrum framework in agile software development. My work focuses on two modules: an online learning system for automated task assignment and an extractive summarization system for sprint review meetings. These modules address the challenges of manual task allocation and verbose meeting documentation, leveraging machine learning and natural language processing to enhance efficiency and accuracy.

# 2. Online Learning for Task Assignment

This section describes the development, implementation, and evaluation of the online learning system for task assignment. The system automates developer recommendations for Jira issues, adapting dynamically to changing team dynamics using online machine learning techniques.

## 2.1. Technical Background

### 2.1.1. Paradigms of Machine Learning for Task Assignment

With text data represented as numerical vectors, various machine learning models can be employed. These models can be categorized into two distinct learning paradigms based on how they are trained.

- **Batch Learning**: This is the traditional paradigm where the model is trained on a large, static dataset (a "batch") all at once. The model learns the patterns from the entire historical dataset before it is deployed to make predictions. The **Topic Modeling (LDA)** and **Deep Learning (LSTM)** approaches implemented in this project are examples of batch learning. They are powerful for finding global patterns in historical data but must be periodically retrained from scratch to incorporate new information.
- **Online Learning (Stream Learning)**: This paradigm is designed for dynamic environments where data arrives sequentially. Instead of being trained on a static dataset, an online model updates itself incrementally with each new data point (e.g., each new Jira issue). The core process is a "test-then-train" cycle: the model first makes a prediction for a new issue, and once the correct assignment is known, it immediately learns from that single instance. This allows the model to adapt continuously to changes in the project, such as developers gaining new skills or the nature of tasks evolving over time. A key component of online learning is **concept drift detection**, using algorithms like **ADWIN (ADaptive WINdowing)**,

to identify when the underlying patterns in the data have changed, signaling that the model may need to adjust more rapidly.

## 2.2. Comparative Study of Previous Work

Recent research in automated issue assignment has moved beyond simple classification towards more sophisticated and practical frameworks.

- **Foundational Classification and Ranking**: Much of the research has focused on selecting the best machine learning model for the job. A comprehensive survey by **Al-Fraihat et al.** compared the performance of several classical batch-learning algorithms, including Decision Trees and SVM [1]. Acknowledging that assigning to a specific person can be rigid, **Shafiq et al.** developed the TaskAllocator system, which recommends a suitable *role* instead [2]. A significant evolution is framing the problem as "learning to rank," which is more aligned with real-world needs. The work by **Alkhazi et al.** is a prime example, using historical data to provide a ranked list of suitable developers rather than a single prediction [3].

- **Quantitative and Interpretable Frameworks**: To build more trustworthy systems, researchers have developed structured frameworks. The work by **Aslam and Ijaz** proposes a quantitative framework that calculates a "Task-Member Suitability Index" based on weighted factors like developer skills and workload [4]. Addressing the "black box" nature of many models, **Samir et al.** focused on interpretability, using techniques like SHAP (SHapley Additive exPlanations) with a Random Forest model to explain *why* it recommended a particular developer, a crucial feature for gaining the trust of project managers [5] .

## 2.3. Design and Architecture

### 2.3.1. Functional Description

The primary function of this module is to automate task assignment in agile software development by predicting and recommending the most suitable developer for a given Jira issue, using online machine learning to adapt dynamically to evolving team dynamics and assignment patterns. Unlike offline learning approaches that rely on static datasets and periodic retraining, this module processes Jira issues in a streaming fashion, enabling continuous learning and adaptation to concept drift without requiring full model retraining. This functionality addresses the need for flexibility in agile environments, where task

assignments are influenced by technical suitability, learning objectives, workload balancing, and team evolution.

This module operates in two distinct modes:

- **Training Mode**: Fetches historical Jira issues for a specified project, preprocesses the data, trains an online learning model, and saves the trained model to AWS S3 for deployment.
- **Testing Mode**: Processes new Jira issues in real-time, predicts the top N (typically 3) recommended developers for assignment, and updates the model incrementally upon receiving the true assignment from a user or manager.

The input to the module is a Jira issue represented as a JSON object, containing attributes such as:

- **Summary**: A concise title describing the issue (e.g., “Implement user authentication endpoint”).
- **Description**: Detailed text outlining the issue, often in Atlassian Document Format (ADF).
- **Labels**: Tags associated with the issue (e.g., “PULL-req”, “bug”).
- **Components**: System components affected by the issue.
- **Priority**: The issue’s priority level (e.g., “Medium”, “High”).
- **Issue Type**: The category of the issue (e.g., “Task”, “Bug”).

The output is a ranked list of the top 3 recommended developers, along with an updated model state after learning from the true assignment.

### 2.3.2. Modular Decomposition

This module, responsible for automating task assignment in agile software development using online machine learning, is decomposed into five fine-grained components to ensure modularity, maintainability, and scalability. Each component handles a specific aspect of the task assignment pipeline, from data acquisition to model deployment and user interaction. This modular design facilitates independent development, testing, and integration with other project modules, while enabling seamless operation within the agile workflow. The components are designed to be interoperable, communicating through standardized data formats such as JSON, CSV, and pickle files. Figure 1 illustrates the pipeline architecture, showing the flow of data through the components.

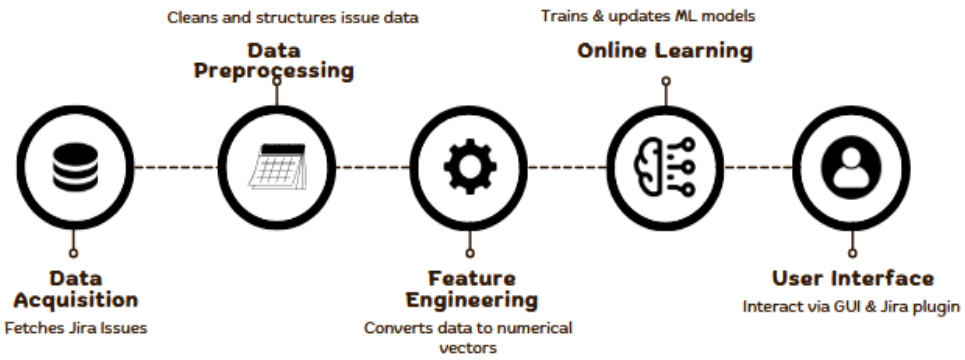


***Figure 1:*** *Online Task Assignment Pipeline*

The five components of this module are:

1. **Data Acquisition Component**:
    - **Function**: Retrieves Jira issues from the Jira REST API, supporting both training and Training modes. In training mode, it fetches historical issues for a specified project to build a baseline model. In Training mode, it retrieves individual new issues for real-time processing.
    - **Implementation**: Utilizes the Python requests library to make API calls, authenticating with an email and API token. The get_text_from_description function parses Atlassian Document Format (ADF) descriptions into plain text, extracting attributes such as summary, description, labels, components, priority, and issue type. Issues are saved as JSON files in the data/ directory for training or deploy and test/jsons/ for deployment.
    - **Output**: JSON files containing raw Jira issue data, structured for downstream processing.
    - **Rationale**: This component ensures robust integration with Jira, handling API rate limits and errors, and provides a flexible interface for fetching data in both batch and streaming scenarios.

2. **Data Preprocessing Component**:
    - **Function**: Transforms raw JSON issue data into a structured format suitable for machine learning, addressing inconsistencies and preparing data for feature engineering.
    - **Implementation**: Employs the pandas library to clean and normalize data, handling missing values (e.g., replacing null fields with empty strings) and converting complex attributes (e.g., lists of labels) into strings. Assignee names are mapped to numerical IDs, with mappings saved for later use in deployment. The processed data is stored as CSV files in data/ for training or deploy and test/csvs/ for deployment.

- **Output**: CSV files with columns such as summary, description, labels, components, priority, issue_type, and assignee_num, ready for feature extraction.
- **Rationale**: This component ensures data consistency and compatibility with machine learning models, enabling efficient processing of diverse Jira issue attributes.

3. **Feature Engineering Component**:
    - **Function**: Converts structured issue data into numerical feature vectors for input to online learning models, capturing the semantic and categorical properties of issues.
    - **Implementation**: Utilizes a custom feature engineering pipeline defined in the MyOnlineModel class, which orchestrates feature selection and transformation. Text attributes (e.g., summary, description) are processed using Term Frequency-Inverse Document Frequency (TF-IDF), defined as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \cdot \log\left(\frac{N}{\text{DF}(t)}\right)$$

      where $TF(t, d)$ is the term frequency of term t in document d, $DF(t)$ is the number of documents containing t, and N is the total number of documents. Categorical attributes (e.g., priority, issue type) are encoded numerically. The pipeline ensures sparse, high-dimensional data is handled efficiently.
    - **Output**: Numerical feature vectors representing each Jira issue, suitable for model training and prediction.
    - **Rationale**: This component bridges the gap between raw data and machine learning, enabling models to learn from both textual and categorical issue attributes effectively.

4. **Online Learning Component** :
    - **Function**: Trains and evaluates online learning models, adapting to new issues and concept drift in a streaming fashion. It supports eight models, from simple classifiers (e.g., Naive Bayes) to advanced ensembles (e.g., AdaBoost ensemble with 10 Naive Bayes models).
    - **Implementation**: Uses the river library to implement models in a test-then-train loop, where each issue is first used for prediction, then for updating the model. Models are trained incrementally, with performance metrics (e.g., accuracy, drift detection counts) recorded. Trained models are serialized as pickle files and uploaded to AWS S3 for persistence and deployment. The component also supports experiment mode, running models across multiple projects in parallel using multiprocessing.

- **Output**: Trained model files (.pkl) stored in AWS S3, along with JSON files containing performance metrics (e.g., accuracy, drift counts) in results_new_1/.
- **Rationale**: This component is the core of this module, enabling dynamic learning and adaptation to evolving assignment patterns, with robust storage and evaluation mechanisms.

5. **User Interface Component**:
   - **Function**: Provides a graphical interface for users to interact with the task assignment system, enabling retrieval of Jira issues for training, real-time model training, deployment to AWS S3, and prediction of developer assignments via a Jira plugin, supporting both operational use and experimentation.
   - **Implementation**: The Training tab is built using Python's tkinter library, enabling users to retrieve Jira issues via the Jira API (e.g., jira-python), input issue details (e.g., summary, description, priority), and train online learning models (e.g., Naive Bayes, AdaBoost with 10 Naive Bayes models) incrementally using the MyOnlineModel framework. Trained models are uploaded to AWS S3 using the AWS SDK (boto3) for persistent storage and retrieval. The Testing tab is implemented as a Jira plugin, integrated into Jira's interface, which loads the AWS-hosted model to predict top-3 developer assignments with confidence scores for new or existing issues. Both tabs authenticate securely with Jira and AWS via user-provided credentials or configuration files.
   - **Output**: The Training tab displays retrieved Jira issues, training progress, and model performance metrics (e.g., accuracy from RollingAccuracyCalc), with trained models saved as .pkl files in AWS S3. The Testing tab, via the Jira plugin, displays top-3 developer predictions with confidence scores within Jira's interface.
   - **Rationale**: This component enhances usability, making the system accessible to non-technical users (e.g., project managers) and facilitating real-world deployment and experimentation.

# 2.4. Results

To select the best model, eight online learning models were trained and evaluated on Apache Jira issues from five projects. Naive Bayes served as the baseline due to its simplicity and efficiency. Each model was trained incrementally using the learn_one method and evaluated using predict_one on a held-out test set per project. Accuracy was computed via RollingAccuracyCalc, with concept drift monitored using ADWIN.

***Table 5.5:*** *Naïve_Bayes Model Results*

| ***Project*** | ***Naïve Bayes*** |
|---|---|
| AMBARI | 59.59% |
| ARROW | 34.25% |
| CASSANDRA | 51.37% |
| CB | 74.28% |
| FLINK | 26.91% |
| GEODE | 41.76% |
| HDDS | 44.07% |
| HIVE | 54.04% |
| HUDI | 45.37% |
| IGNITE | 42.23% |
| IMPALA | 41.52% |
| IOTDB | 47.10% |
| MESOS | 40.95% |
| OAK | 68.01% |
| SPARK | 74.17% |

***Table 5.6:*** *Naïve_Bayes with ADWIN Results*

| ***Project*** | ***Naïve_Bayes with ADWIN*** |
|---|---|
| AMBARI | 67.73% |
| ARROW | 39.58% |
| CASSANDRA | 57.41% |
| CB | 73.49% |
| FLINK | 37.96% |
| GEODE | 43.48% |
| HDDS | 48.75% |
| HIVE | 55.51% |
| HUDI | 47.19% |
| IGNITE | 45.16% |
| IMPALA | 43.90% |
| IOTDB | 45.76% |
| MESOS | 42.45% |
| OAK | 68.01% |
| SPARK | 71.20% |

***Table 5.7:*** *HoeffdingAdaptiveTreeModel Results.*

| ***Project*** | ***HoeffdingAdaptiveTreeModel*** |
|---|---|
| AMBARI | 34.84% |
| ARROW | 15.81% |
| CASSANDRA | 25.91% |
| CB | 31.98% |
| FLINK | 12.61% |
| GEODE | 17.76% |
| HDDS | 26.89% |
| HIVE | 20.40% |
| HUDI | 26.20% |
| IGNITE | 14.94% |
| IMPALA | 26.35% |
| IOTDB | 20.49% |
| MESOS | 14.84% |
| OAK | 43.91% |
| SPARK | 36.82% |

***Table 5.8:*** *PassiveAggressiveModel Results.*

| ***Project*** | ***PassiveAggressiveModel*** |
|---|---|
| AMBARI | 62.60% |
| ARROW | 35.23% |
| CASSANDRA | 45.77% |
| CB | 68.72% |
| FLINK | 33.67% |
| GEODE | 37.43% |
| HDDS | 40.17% |
| HIVE | 43.20% |
| HUDI | 43.97% |
| IGNITE | 38.38% |
| IMPALA | 37.25% |
| IOTDB | 34.62% |

| | |
|---|---|
| MESOS | 37.57% |
| OAK | 51.14% |
| SPARK | 53.93% |

***Table 5.9:*** *LeveragingBaggingModel Results.*

| ***Project*** | ***LeveragingBaggingModel*** |
|---|---|
| AMBARI | 40.93% |
| ARROW | 25.03% |
| CASSANDRA | 27.80% |
| CB | 18.14% |
| FLINK | 19.14% |
| GEODE | 22.47% |
| HDDS | 20.00% |
| HIVE | 21.14% |
| HUDI | 21.17% |
| IGNITE | 24.73% |
| IMPALA | 22.68% |
| IOTDB | 25.05% |
| MESOS | 21.03% |
| OAK | 39.20% |
| SPARK | 44.11% |

***Table 5.10:*** *SoftMax Model Results.*

| ***Project*** | ***SoftMax Model*** |
|---|---|
| AMBARI | 66.49% |
| ARROW | 44.53% |
| CASSANDRA | 52.47% |
| CB | 74.68% |
| FLINK | 40.88% |
| GEODE | 39.91% |
| HDDS | 51.95% |
| HIVE | 54.56% |
| HUDI | 50.00% |

| | |
|---|---|
| IGNITE | 47.29% |
| IMPALA | 42.15% |
| IOTDB | 42.39% |
| MESOS | 42.83% |
| OAK | 63.05% |
| SPARK | 59.51% |

***Table 5.11:*** *Adaboost (Enhanced Model) Results.*

| ***Project*** | ***Adaboost*** |
|---|---|
| AMBARI | 68.7% |
| ARROW | 47.09% |
| CASSANDRA | 60.37% |
| CB | 76.76% |
| FLINK | 47.55% |
| GEODE | 51.75% |
| HDDS | 59.78% |
| HIVE | 65.32% |
| HUDI | 53.39% |
| IGNITE | 54.41% |
| IMPALA | 48.20% |
| IOTDB | 51.65% |
| MESOS | 52.91% |
| OAK | 67.87% |
| SPARK | 74.69% |

***Table 5.12:*** *Enhanced Adaboost (Super EnhancedModel) Results*

| ***Project*** | ***Enhanced Adaboost*** |
|---|---|
| AMBARI | 71.53% |
| ARROW | 48.83% |
| CASSANDRA | 63.45% |
| CB | 79.74% |
| FLINK | 46.56% |
| GEODE | 53.34% |
| HDDS | 60.66% |

| | |
|---|---|
| HIVE | 68.67% |
| HUDI | 53.64% |
| IGNITE | 56.58% |
| IMPALA | 49.78% |
| IOTDB | 56.28% |
| MESOS | 52.60% |
| OAK | 70.55% |
| SPARK | 76.79% |

# 3. Extractive Summarization

This section describes the development, implementation, and evaluation of the extractive summarization system for sprint review meetings. The system generates concise, speaker-aware summaries of meeting transcripts, prioritizing agile-specific information.

## 3.1. Technical Background

This subsection provides the theoretical foundation for extractive summarization, detailing techniques and algorithms used in the system.

### 3.1.1. Extractive Text Summarization Techniques

Text summarization is a core task in natural language processing (NLP) aimed at condensing a document into a concise representation while retaining its key information. Summarization is broadly divided into extractive and abstractive approaches. Extractive summarization, central to this project, selects the most significant sentences directly from the source text without altering their content. Abstractive summarization generates new sentences through paraphrasing and synthesis.

Extractive summarization relies on scoring mechanisms to rank sentences based on their importance. Common techniques include:

- **Statistical Methods**: Term Frequency -Inverse Document Frequency (TF-IDF) measures word importance by combining term frequency $\left(\mathrm{TF}(w,d)\right)$, the count of word (w) in document (d)) with inverse document frequency $\mathrm{IDF}(w,D) = \log\left(\frac{|D|}{|\{d \in D : w \in d\}|+1}\right) + 1$, smoothed to prevent zero divisions. The TF-IDF score is:

$$\mathrm{TFIDF}(w,d,D) = \mathrm{TF}(w,d) \cdot \mathrm{IDF}(w,D)$$

- **Graph-Based Methods**: Algorithms like TextRank and LexRank model sentences as nodes in a graph, with edges weighted by similarity metrics (e.g., Jaccard or cosine similarity). TextRank applies a PageRank-inspired formula to rank sentences:

$$\text{Score}(v_i) = (1-d) + d \sum_{v_j \in \text{In}(v_i)} \frac{\text{sim}(v_i, v_j)}{\sum_{v_k \in \text{Out}(v_j)} \text{sim}(v_j, v_k)} \cdot \text{Score}(v_j)$$

where (d) (typically 0.85) is the damping factor, and $\left(\text{sim}(\text{v}_\text{i}, \text{v}_\text{j})\right)$ denotes similarity between sentences (v_i) and (v_j).

- **Machine Learning Methods**: Supervised models (e.g., classifiers) or unsupervised techniques (e.g., clustering) predict sentence salience, often requiring labeled data.

# 3.2. Comparative Study of Previous Work

Extractive text summarization is a well-established field with decades of research, where foundational methods like TF-IDF and TextRank remain robust due to their simplicity and effectiveness. Recent publications focus on incremental improvements, such as feature-based scoring, graph-based enhancements, and clustering, but innovations are limited. Below, we compare four recent studies from IEEE and other peer-reviewed sources, highlighting their contributions, strengths, and limitations.

**Rani and Lobiyal (2021)**: Published in *Expert Systems with Applications*, this study proposes a weighted word embedding-based approach for extractive summarization [6]. It uses word embeddings to score sentences, enabling user-requested summaries tailored to specific needs.

- **Strength**: Effective for generating on-demand summaries with high relevance to user queries.
- **Limitation**: Struggles to manage redundancy, potentially including repetitive sentences in summaries, which reduces coherence.

**Naik and Gaonkar (2017, revisited in surveys)**: Published in the *2017 IEEE RTEICT Conference*, this work presents a feature-based sentence extraction method using rule-based concepts [7]. It selects sentences that maximize information coverage by combining multiple features (e.g., TF-IDF, sentence position).

- **Strength**: Produces summaries with broad content coverage by selecting informative sentences.
- **Limitation**: Developing effective rules requires significant time and domain expertise, limiting scalability across diverse datasets.

**Belwal, Rai, and Gupta (2021)**: Published in *Journal of Ambient Intelligence and Humanized Computing*, this study introduces a graph-based extractive summarization

method using keyword and topic modeling [8]. It enhances coherence by identifying and reducing redundant sentences.

- **Strength**: Improves summary coherence and effectively handles redundant content through graph-based analysis.
- **Limitation**: Fails to capture sentences with semantic equivalents, potentially missing nuanced content variations.

**Sharaff, Jain, and Modugula (2022)**: Published in *International Journal of Information Technology*, this work proposes a feature-based cluster ranking approach for single-document summarization [9]. It minimizes the impact of sentence order on summary quality by clustering sentences based on features like TF-IDF and semantic similarity.

- **Strength**: Addresses coherence issues by reducing sensitivity to sentence order.
- **Limitation**: Requires pre-specification of the number of clusters, which can be challenging to optimize without prior knowledge.

# 3.3. Design and Architecture

## 3.3.1. Functional Description

This module is designed to automate the summarization of sprint review meeting transcripts in agile software development, producing concise, speaker-aware summaries that capture critical information such as task progress, blockers, and assignments. This module addresses the need for efficient documentation in agile environments, where sprint reviews generate verbose discussions that must be distilled into actionable insights for stakeholders, such as project managers and developers. Unlike abstractive summarization, which generates paraphrased summaries and is handled by a team member, This module employs an extractive summarization approach, selecting and presenting the most relevant utterances verbatim to preserve the original context and intent.

This module processes a raw meeting transcript and produceses a summary. It supports two extractive summarization methodologies developed as part of this project: **TF-IDF Summarizer** and **TextRank Summarizer**. These methodologies prioritize utterances based on their content relevance, speaker roles, and domain-specific keywords, ensuring that summaries highlight key points such as completed tasks, blockers requiring intervention, and planned actions.

The input to This module is a text transcript of a sprint review meeting, formatted as a sequence of speaker-utterance pairs, typically following the pattern “Speaker: Utterance” (e.g., “Team Lead: Let’s start the sprint review”). The transcript may include:

- **Speaker Information**: Names or roles (e.g., “Team Lead”, “John”).

- **Utterance Content**: Statements about task progress (e.g., “I completed the login page redesign”), blockers (e.g., “I’m blocked on the API integration”), or assignments (e.g., “We’ll assign Mike to help”).
- **Contextual Details**: Numeric data (e.g., “5 story points”) or agile-specific terms (e.g., “velocity”).

The output is a structured summary in two formats:

- **Clean Summary**: A list of selected speaker-utterance pairs, preserving the original wording and order (e.g., “John: I completed the login page redesign, 5 story points”).
- **Debug Summary**: An annotated version including weights or scores for each utterance, useful for analysis and tuning (e.g., “John: I completed the login page redesign [Weight: 0.85]”).

### 3.3.2. Modular Decomposition

This module, responsible for automating the summarization of sprint review meeting transcripts in agile software development, is decomposed into four fine-grained components to ensure modularity, maintainability, and scalability. These components form a pipeline that processes raw transcripts into concise, speaker-aware summaries using two extractive summarization methodologies: TF-IDF Summarizer and TextRank Summarizer.

The four components of This module are:

1. **Preprocessing Component**:
    - **Function**: Extracts speaker-utterance pairs from raw meeting transcripts and cleans the text for downstream processing, ensuring compatibility with both TF-IDF and TextRank summarizers.
    - **Implementation**: The MeetingPreprocessor class processes transcripts by splitting them into lines and using regular expressions to identify speaker-utterance pairs (e.g., “Team Lead: Let’s start the sprint review”). The extract_dialogues method handles continuation lines by appending them to the previous speaker’s utterance. The preprocess_utterance method tokenizes utterances, converts them to lowercase, removes punctuation, and filters out stopwords (e.g., “a”, “the”) and short words (< 3 characters). Custom stopwords, such as agile-specific terms (“sprint”, “review”), can be provided to tailor preprocessing to the domain.
    - **Output**: A list of tuples (List[Tuple[str, str]]) containing speaker-utterance pairs and a list of tokenized utterances (List[List[str]]) for feature extraction.
    - **Rationale**: This component ensures that raw, unstructured transcripts are standardized into a format suitable for summarization, handling variations

in transcript style (e.g., multi-line utterances) and removing noise to improve feature quality.

2. **TF-IDF Calculation Component**:
    - The TF-IDF Calculator class computes TF-IDF scores for the tokens in each utterance. This process assigns a weight to each word, indicating its importance to an utterance in the context of the entire meeting transcript. These scores provide the core numerical features used by the TF-IDF Summarizer to rank sentences

3. **Sentence Scoring and Selection Component**:
    - **Function**: Scores utterances based on their relevance and selects the most important ones for the summary, implementing distinct strategies for TF-IDF and TextRank summarizers.
    - **Implementation**:
        - **TF-IDF Summarizer**: The Summarizer class's score_sentences method assigns scores to utterances by averaging TF-IDF scores of their tokens. It applies weighting strategies:
            - **Non-Team Leader Boost**: Utterances from non-lead speakers (e.g., excluding "Team Lead", "Manager") receive a 1.3x weight to prioritize developer contributions, as team leader utterances often include non-informative prompts (e.g., "Great", "What did you work on?").
            - **Problem and Action Keywords**: Utterances containing problem keywords (e.g., "blocked", "issue", WordNet synonyms like "obstacle") are boosted by 2.0x, while action keywords (e.g., "completed", "assigned", synonyms like "finished") receive a 1.5x boost. These keywords, defined in summarizer.py, are expanded using WordNet to include synonyms, ensuring comprehensive coverage of agile-relevant terms.
            - **Numeric Data Boost**: Utterances with numbers (e.g., "5 story points") are boosted by 2.0x to capture quantitative updates.
            - **Dynamic Sentence Selection**: To address the challenge of determining summary length (a critical issue in extractive summarization), the method selects all utterances containing problem or action keywords and identifies the minimum score among them $\left((\text{min\_keyword\_score})\right)$ Additional utterances with scores ($\geq$ min_keyword_score) are included, ensuring a

non-hardcoded, adaptive summary length that balances coverage and conciseness.

- **TextRank Summarizer**: The TextRank class builds a similarity graph using Jaccard similarity:

$$\text{Similarity}(s_1, s_2) = \frac{|\text{set}(s_1) \cap \text{set}(s_2)|}{|\text{set}(s_1) \cup \text{set}(s_2)| + 10^{-7}}$$

where $(s_1, s_2)$ are tokenized utterances. The run_pagerank method applies the PageRank algorithm with a damping factor $\big((\text{damping} = 0.85)\big)$ to rank utterances, selecting the top (N) based on SummaryConfig.top_n.

- **Output**: For TF-IDF, a list of (index, score) tuples and a set of keyword-containing utterance indices; for TextRank, a list of utterance scores and selected indices.
- **Rationale**: This component encapsulates the core ranking logic, with TF-IDF emphasizing keyword-driven relevance and TextRank capturing semantic coherence, ensuring summaries are both informative and contextually relevant.

4. **Summary Generation Component**:
   - **Function**: Formats the selected utterances into clean and debug summaries, preserving speaker information and original wording for extractive output.
   - **Implementation**:
     - **TF-IDF Summarizer**: The generate_summary method formats selected utterances into two outputs: a clean summary (e.g., “John: I completed the login page redesign”) and a debug summary with weights (e.g., “John: I completed the login page redesign [Weight: 0.85]”). Utterances are capitalized for readability.
     - **TextRank Summarizer**: The summarize method returns a dictionary with a list of speaker-utterance pairs and debug information (scores, selected indices), maintaining original order.
   - **Output**: A dictionary with clean (string of formatted utterances) and debug (string or dictionary with weights/scores) keys.
   - **Rationale**: This component ensures user-friendly, actionable summaries for agile teams, with debug outputs facilitating analysis and tuning. The preservation of original wording addresses the need for accurate documentation in sprint reviews.

# 4. Conclusion

My contributions to the Aigile project demonstrate the application of AI to enhance Scrum processes. The online learning task assignment system achieved a peak accuracy of 79.74% using an AdaBoost ensemble with drift detection and recency heuristics, effectively adapting to dynamic agile environments. The extractive summarization system produced relevant, speaker-aware summaries of sprint review transcripts using TF-IDF and TextRank methodologies, addressing the need for efficient documentation. Integrated into Jira as plugins and a local GUI, these modules streamline task allocation and meeting documentation, showcasing the potential of AI to optimize agile workflows.

# 5. References


***[1]*** *D. Al-Fraihat, B. Al-Shboul, and M. Al-Sayyed, "A comprehensive survey on the automatic bug triage," Journal of Systems and Software, vol. 175, pp. 110915, 2021.*

***[2]*** *U. Shafiq, T. Ali, and A. Ahmad, "TaskAllocator: A Recommendation System for Task Assignment in Software Development," in Proceedings of the 2021 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, Oct. 2021, pp. 1-11.*

***[3]*** *A. Alkhazi, M. W. Mkaouer, and A. Ouni, "Learning to Rank for Bug Assignment," in Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, Nov. 2020, pp. 678-689.*

***[4]*** *F. Aslam and M. Ijaz, "A quantitative framework for task assignment in agile software development," Information and Software Technology, vol. 147, pp. 106889, 2022.*

***[5]*** *A. Samir, M. A. Ghoneim, and H. El-Deeb, "An interpretable approach for bug assignment using explainable AI," Expert Systems with Applications, vol. 213, pp. 118934, 2023.*

***[6]*** *R. Rani and D. K. Lobiyal, "Extractive text summarization using a novel weighted word embedding based approach," Expert Systems with Applications, vol. 183, pp. 115383, 2021.*

***[7]*** *N. R. Naik and M. N. Gaonkar, "Extractive Text Summarization Using Feature Based Sentence Extraction," in 2017 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, May 2017, pp. 896-900.*

***[8]*** *R. Belwal, S. Rai, and S. Gupta, "A graph-based extractive text summarization method using keywords and topic modeling," Journal of Ambient Intelligence and Humanized Computing, vol. 12, pp. 10457–10470, 2021.*

***[9]*** *A. Sharaff, S. Jain, and S. Modugula, "A feature-based cluster ranking approach for single-document text summarization," International Journal of Information Technology, vol. 14, pp. 2481–2489, 2022.*