Cairo University
Faculty of Engineering
Department of Computer Engineering

Aigile

The Intelligent Automation of the Scrum Workflow

A Graduation Project **Individual 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

Mennatallah Ahmed Moustafa

Supervised by

Dr. Ahmed Darwish

July 2025

Chapter 4: System Design and Architecture

4.1. Module 1: Backlog Generation

Module 1 is decomposed into backend and frontend sub-modules to ensure maintainability and scalability. The backend handles AI processing and data management, while the frontend provides the user interface. Below is the detailed decomposition.

I contributed to the development and deployment of Module 1, focusing on the **User Story Generation Sub-Module**, the **Prioritization Sub-Module**, and the deployment of the backend with a Flask server on PythonAnywhere. Below is a detailed description of my work:

User Story Generation Sub-Module

- **Purpose**: Generates structured user stories from refined project requirements.
- **My Contributions**:
    - Implemented the generate_user_stories function in generate_us.py, which processes refined requirements into a prompt using a template from generate_us_prompt_content.txt.
    - Developed the user_story_parser function in generate_us.py, utilizing regular expressions to extract user stories, epics, and descriptions from the LLM's output, formatting them into a list of story dictionaries.
    - Ensured the output adhered to a structured format (user story, epic, description) for downstream processing.

Prioritization Sub-Module

- **Purpose**: Assigns priority scores to generated user stories using a simulated multi-agent approach.
- **My Contributions**:
    - Designed and implemented the agents function in prioritize_us.py, which coordinates three simulated agents (Product Owner, Senior Developer, Senior QA) to prioritize user stories.

- Developed the construct_prompt function in prioritization_helpers.py to combine agent-specific prompts with the final prioritization prompt (batch_100_dollar_prompt.txt), enabling the LLM to allocate a $100 budget across user stories.
- Implemented the parse_response function to extract and process the LLM's allocation outputs, ensuring accurate priority scores for each story.

Backend Deployment and Flask Server

- **Purpose**: Deployed the backend infrastructure to ensure the reliable operation of Module 1.
- **My Contributions**:
  - Configured and deployed the backend on PythonAnywhere, ensuring seamless integration with the frontend and external services.
  - Set up and maintained a Flask server to handle API requests, including endpoints for user story generation and prioritization, hosted at https://mou3.pythonanywhere.com.
  - Optimized server configuration for performance and reliability, including managing dependencies and ensuring compatibility with PythonAnywhere's environment.

4.2. Module 2: Story Point Estimation

The Story Point Estimation module is decomposed into four sub-modules. These sub-modules work together to process data, experiment with models, incorporate historical context, and enable cross-project adaptability. Below, each sub-module is detailed to clarify its role and implementation.

In this part, we tried different methods, starting with random data splits, moving to separate project tests, breaking tasks into smaller parts, and finally using a fair test method to get the best results. Here’s the Step-by-Step Process

Starting with Random Data

We divided our data into two parts: 80% for training (teaching the AI) and 20% for testing (checking the AI). We used a machine learning model called Support Vector Machine (SVM) with BERT embeddings to convert the raw text of the user story into numerical features suitable for the machine learning model, inspired by a prior study reporting 28.8% accuracy with C=5000, gamma=1. Through hyperparameter validation, tuning parameters like C and gamma, we

improved this to **36.33% accuracy** with C=1, gamma=0.1. This was a good start because BERT understands the meaning behind words, which helped with the tricky user stories.

Why We Chose BERT at First: BERT looks at words from both directions in a sentence, making it good at understanding complex tasks, like figuring out if a story needs a lot of work or just a little.

Recognizing the limitations of the initial SVM with BERT, we refined the random data approach to enhance generalization, reduce computational overhead, and address data imbalance. We introduced two key improvements: stratified k-fold cross-validation and SMOTE (Synthetic Minority Oversampling Technique).

1. **Stratified k-Fold Cross-Validation**: Instead of a single 80/20 split, we adopted stratified k-fold cross-validation to ensure robust evaluation. Stratified sampling preserved the proportional representation of story point values in each fold, mitigating bias from imbalanced data. The model is trained on k-1 folds and tested on the last fold, repeating for all folds to compute average performance metrics.

2. **SMOTE for Data Imbalance**: We applied SMOTE to the training data to address the skewed distribution of story points, generating synthetic samples for underrepresented story points. SMOTE creates synthetic examples by interpolating between minority class samples, improving model performance on rare classes. The SMOTE is applied after preprocessing but before embedding generation.

We also explored alternative embeddings and models to optimize performance. All the experiments are detailed below.

1. Exploring Various Embeddings with the SVM Model, as it is the most widely used Method in Previous Story Point Estimation Studies

Table 4.4: *Comparison of Embedding Methods for SVM in Story Point Estimation*

Model Approach	Feature Set	Test MAE (↓)	Test Accuracy (↑)
Classification (SVM)	TF-IDF	1.5504	38.66%
Classification (SVM)	GloVe	1.8759	33.10%
Classification (SVM)	fastText	1.5059	40.00%
Classification (SVM)	BERT	1.6694	36.33%
Classification (SVM)	SBERT (all-MiniLM-L6-v2)	1.465	41.30%
Classification (SVM)	SBERT (all-mpnet-base-v2)	1.4775	41.00%
Classification (SVM)	SBERT + fastText	1.461	40.40%

- **TF-IDF (Term Frequency-Inverse Document Frequency)**:
    - **Why We Used It**: This method is a quick and easy way to count how often words appear in user stories, giving us a basic starting point.
    - **Why It Fell Short**: It doesn't understand the deeper meaning behind words, which makes it less helpful for figuring out story points.
- **GloVe + SMOTE**:
    - **Why We Used It**: GloVe looks at words in a broader context across the whole text, not just a small window, to balance speed and meaning.
    - **Why It Fell Short**: Its embeddings stay the same and don't adapt to new context, so it struggled to capture the full meaning of user stories.
- **FastText**:
    - **Why We Used It**: This method is good at handling rare or unusual words by breaking them into smaller parts, making it more flexible.
    - **Why It Was Better Than TF-IDF**: It understands word shapes (morphology), which helped it perform better than TF-IDF.
    - **Extra Note**: We tried a pre-trained FastText model, but it only got 37% accuracy. Training it on our data gave us a better 40% accuracy.
- **BERT**:
    - **Why We Used It**: BERT understands the context of words by looking at the whole sentence, which seemed perfect for our task.
    - **Why It Fell Short**: It learned too much from the specific projects we trained it on and got confused by small noises in the data, causing it to overfit.
- **SBERT (all-MiniLM-L6-v2)**:
    - **Why We Used It**: SBERT is designed to compare sentence meanings, which fits well with understanding similar user stories.
    - **Results**: When used with SVM, it achieved **41.34% accuracy** with an MAE of 1.4650 and MdAE of 1.0000.
    - **Why It Was Better**: It handles cases where stories sound similar but mean the same thing, making it the best option so far.
- **SBERT + FastText**:
    - **Why We Used It**: We combined SBERT's sentence understanding with FastText's word-part strength, using the top two methods together.
    - **Why It Worked Well**: This mix gave us an accuracy close to SBERT alone, showing it's a strong team effort.

Why SBERT Beats BERT, Even Though Both Use Context

Both BERT and SBERT work by understanding context, but they perform differently. BERT is a big, general model that needs a lot of extra tuning to fit our specific task, which we didn't fully do. SBERT, on the other hand, is a smaller, specially tuned version made for comparing

sentences, and its simple design let it work better right away. Even though BERT has a more advanced structure, SBERT's focus on sentence similarity gave it the edge for our project.

Also, SBERT's confusion matrix has the strongest diagonal (e.g., 1115 for 1), showing it predicts better, while BERT's higher off-diagonals (e.g., 438) indicate overfitting.

Table 4.5: *Confusion Matrix for SBERT with SVM in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	1115	408	147	101	60
2	649	597	161	96	90
3	321	257	328	239	83
5	175	145	215	405	151
8	116	128	75	208	251

Table 4.6: *Confusion Matrix for BERT with SVM in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	978	438	205	124	86
2	621	566	181	130	95
3	343	263	295	221	106
5	211	171	210	332	167
8	134	165	104	177	198

How This Helps Story Point Estimation

Guessing story points depends a lot on the effort and complexity hidden in user story descriptions. For example, two stories about minor UI fixes should get the same point, even if the words are a bit different. By using embeddings like SBERT that keep similar meanings close together, we give the model a clear space to work in. This reduces the confusion from small word changes and lets the AI focus on the real effort needed

2. Testing Different Models

- **SVM (Classification)**:
    - **Why We Chose It**: This model is strong at working with text data, making it a good choice for our user stories.

    - **What We Found**: It worked best with SBERT, giving us **41.34% accuracy** and an MAE of 1.465.
- **SVR (Regression)**:
    - **Why We Chose It**: This model predicts story points as continuous values, which suits our task.
    - **What We Found**: When paired with SBERT, it achieved an MAE of 1.367 and an accuracy of 33.5%.

Table 4.7: *Confusion Matrix for SVR with SBERT in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	403	962	412	54	0
2	200	841	515	37	0
3	44	401	622	161	0
5	19	191	559	320	2
8	12	119	383	263	1

    - **Why It Was Better**: It fits well with the idea that story points are ordered numbers, like 1 being less than 5.
- **XGBRegressor**:
    - **Why We Chose It**: This model can handle tricky, non-straightforward patterns in the data.
    - **What We Found**: It reached **26.73% accuracy** with an MAE of 1.5038.

Table 4.8: *Confusion Matrix for XGBRegressor with SBERT in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	*58*	*765*	*889*	*118*	*1*
2	*26*	*532*	*915*	*120*	*0*
3	*10*	*219*	*774*	*225*	*0*
5	*3*	*92*	*614*	*379*	*3*
8	*1*	*61*	*374*	*342*	*0*

- **Ordinal Regression**:

- **Why We Chose It**: This model treats story points as a sequence where the gap between 3 and 5 is different from 3 to 8, which matches how we think about effort.
- **What We Found**: It got **29.0% accuracy** with an MAE of 1.4619.

Table 4.9: *Confusion Matrix for Ordinal Regression with SBERT in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	121	1114	564	31	1
2	54	885	622	32	0
3	17	418	682	111	0
5	3	226	663	199	0
8	3	140	469	164	2

- **LSTM**:
  - **Why We Chose It**: This model looks at the order of words in a sentence, which might help with the flow of user stories.
  - **What We Found**: It achieved **37.57% accuracy** with an MAE of 1.5351.

Table 4.10: *Confusion Matrix for LSTM with BERT in Story Point Estimation*

True/Predicted	1	2	3	5	8
1	555	209	47	90	15
2	369	264	65	82	17
3	176	135	92	188	23
5	98	84	66	259	38
8	77	74	28	155	55

Adding Extra Information and Cleaning Data

We also tried to add extra details to the user stories and add more cleaning to the text.

1. **Metadata**: We included special features to give the model more clues about the tasks:
   - **Uncertainty & Risk Features**: We checked for signs of uncertainty, which means more work.

        - **Uncertainty Score**: We counted words like "investigate," "explore," or "research" showing how much exploration is needed.
        - **Question Count**: We counted question marks (?) to spot unclear requirements.
    - **Specification Quality & Scope Features**: We measured size and clarity to gauge complexity.
        - **Text Length**: We counted characters in each story; longer text might mean more work.
        - **Readability Grade**: We used the Flesch-Kincaid test to get a grade level showing text difficulty.
    - **Task Type Features**: We identified task types to predict effort.
        - We labeled tasks as "bug", "feature", "refactor", or "other" based on keywords in the text.
    - **Results**: When we added these features to SBERT, we got **37.1% accuracy** with an MAE of 1.601.

Table 4.11: Confusion Matrix for SBERT with Metadata in Story Point Estimation.

True/Predicted	1	2	3	5	8
1	925	508	204	124	70
2	575	583	205	135	95
3	289	246	345	231	117
5	183	163	225	345	175
8	112	125	108	214	219

2. **Preprocessing**: We cleaned the data by removing URLs (web links)
    - **Results**: Using SVR with SBERT after this cleanup gave us an MAE of **1.3598**.

Table 4.12: Confusion Matrix for SVR with Preprocessed SBERT Embeddings

True/Predicted	1	2	3	5	8
1	389	976	418	48	0
2	190	856	504	43	0
3	46	388	645	149	0
5	19	180	572	318	2
8	11	127	370	269	1

- This shows that removing URLs has a little impact

***Table 4.13**: Final Ranking of Models for Story Point Estimation by MAE and Accuracy*

Model Approach	Feature Set	Test MAE	Test Accuracy
Regression (SVR)	SBERT (all-MiniLM-L6-v2)	1.367	33.5%
Classification (SVM)	SBERT + fastText	1.461	40.40%
Ordinal Regression	SBERT	1.4619	29.00%
Classification (SVM)	SBERT (all-MiniLM-L6-v2)	1.465	41.30%
Classification (SVM)	SBERT (all-mpnet-base-v2)	1.4775	41.00%
Regression (SVR)	GPT-2	1.5028	29.7%
XGBRegressor	SBERT	1.5038	26.73%
Classification (SVM)	fastText	1.5059	40.00%
LSTM	BERT	1.5351	37.57%
Classification (SVM)	TF-IDF	1.5504	38.66%
Classification (SVM)	SBERT + Metadata	1.601	37.10%
Classification (SVM)	BERT	1.6694	36.33%
Classification (SVM)	GloVe	1.8759	33.10%
Linear SVM (Classifier)	SBERT	1.8601	35.50%

Conclusion:

- We started with classic NLP (TF-IDF, fastText), achieved a breakthrough with contextual SBERT embeddings.
- **The Best Classifier:** The SVM Classifier with SBERT (MiniLM) gives the highest Accuracy (41.3%), making it the best choice if we need the exact story point.
- **The Best Regressor:** The SVR with SBERT (MiniLM) gives the lowest MAE (1.367), making it the most reliable model for being "close" to the right answer, which is arguably more useful for sprint planning.
- Finally, The choice of embedding technology (**SBERT**) was the single most important factor, and the choice between a classification or regression model depends on the business need: **perfect accuracy vs. overall closeness**.

Switching to a Separate Project

To make our test more realistic, like when a team works on a new project with no old data, we used a different project just for testing. This dropped our accuracy to 20-23%, showing it was hard to use what we learned on old data for something new. We think this happened because the random split let the training and test data share some similar projects, which isn't fair. A comparison between different settings has been made:

Table 4.14: *Results of Separate Project (Initial)*

	SVM + TF-IDF	**SVM + BERT**
C=15 gamma=0.001	Acc:0.235 MAE:1.295	Acc:0.2523 MAE:1.4363
C=5 gamma=0.001	Acc:0.198 MAE: 1.2584	Acc:0.2540 MAE:1.4376
C=1 gamma=0.01	Acc:0.2007 MAE: 1.254	Acc:0.2536 MAE:1.4196
C=10 gamma=auto	Acc:0.1968 MAE: 1.258	Acc:0.2509 MAE: 1.44
C=20 gamma=auto	Acc:0.1968 MAE: 1.258	Acc:0.2514 MAE:1.4346

Comment: This table shows that BERT with SVM started better than TF-IDF, but the accuracy didn't improve much with different settings

Trying to Improve with the Separate Project

To make the separate project work better, we augmented the training data to introduce variety, but this only slightly improved results. Then, we tried filtering to filter similar user stories if they were more than 90% alike, which improved results slightly.

Table 4.15: *Results of Separate Project with Augmentation and Filtering.*

	SVM + TF-IDF	**SVM + BERT**	**SVM + BERT (Augmented Data)**	**SVM + BERT (Augmented + Filtered Data)**
C=15 gamma=0.001	Acc:0.235 MAE:1.295	Acc:0.2523 MAE:1.4363	Acc:0.262 MAE: 1.420	Acc:0.258 MAE: 1.393
C=5 gamma=0.001	Acc:0.198 MAE: 1.2584	Acc:0.2540 MAE:1.4376	Acc:0.2509 MAE: 1.458	Acc:0.245 MAE: 1.4205
C=1 gamma=0.01	Acc:0.2007 MAE: 1.254	Acc:0.2536 MAE:1.4196	Acc:0.2611 MAE: 1.418	Acc:0.258 MAE: 1.385
C=10 gamma=auto	Acc:0.1968 MAE: 1.258	Acc:0.2509 MAE: 1.44	Acc:0.2597 MAE:1.430	Acc:0.2589 MAE: 1.3967

C=20 gamma=auto	Acc:0.1968 MAE: 1.258	Acc:0.2514 MAE:1.4346	Acc:0.2659 MAE: 1.407	Acc:0.2668 MAE: 1.37

Comment: Adding more data and filtering similar stories helped a bit, showing small gains, but still room for improvement.

Breaking Tasks into Smaller Parts

Recognizing the ambiguity in the user stories writing format, we used a Large Language Model (LLM) to break them into subtasks. We tried different ways:

1. We trained the SVM using BERT and TF-IDF embeddings, where the training text was the original user story text. For testing, we calculated the story points by averaging the points predicted for each subtask. (Columns 5, 6)
2. We trained the SVM using BERT embeddings, where the training text was all subtasks combined into one piece of text. For testing, we predicted the story points directly using the original user story text from the test set. (Column 7)
3. We trained the SVM using BERT embeddings, where the training text was all subtasks combined into one piece of text. For testing, we predicted the story points by averaging the points calculated for each subtask of every user story. (Column 8)

This approach improved explainability and boosted performance on the separate project, with the best result at **28.7% accuracy** and MAE of 1.169 on the XD project using SVM + BERT, where training BERT on subtasks combined into a single text, and testing by averaging the story points of subtasks for each user story.

Table 4.16: *Results of Separate Project with Subtask Decomposition.*

	SVM + TF-IDF	**SVM + BERT**	**SVM + BERT (Augmented Data)**	**SVM + BERT (Augmented + Filtered Data)**	**SVM + BERT (Averaging SP of Subtasks)**	**SVM + TF-IDF (Averaging SP of Subtasks)**	**SVM + BERT (Subtasks as Single Text, Direct Prediction)**	**SVM + BERT (Subtasks as Single Text, Averaging SP of Subtasks)**
C=15 gamma=0.001	Acc: 0.235 MAE:1.295	Acc: 0.2523 MAE:1.4363	Acc: 0.262 MAE: 1.420	Acc: 0.258 MAE: 1.393	Acc: 0.2395 MAE: 1.2346	Acc:0.21576 MAE: 1.1836	Acc: 0.2862 MAE:1.3434	Acc:0.287 MAE:1.169
C=5 gamma=0.001	Acc: 0.198 MAE: 1.2584	Acc: 0.2540 MAE:1.4376	Acc: 0.2509 MAE: 1.458	Acc: 0.245 MAE: 1.4205	Acc: 0.2404 MAE: 1.3069	Acc: 0.2228 MAE 1.808	Acc:0.2814 MAE:1.356	Acc:0.282 MAE:1.166

C=1 gamma=0.01	Acc: 0.2007 MAE: 1.254	Acc: 0.2536 MAE:1.4196	Acc: 0.2611 MAE: 1.418	Acc: 0.258 MAE: 1.385	Acc: 0.2417 MAE: 1.247	Acc: 0.2034 MAE: 1.2487	Acc:0.2870 MAE:1.339	Acc: 0.2567 MAE: 1.2659
C=10 gamma=auto	Acc: 0.1968 MAE: 1.258	Acc: 0.2509 MAE: 1.440	Acc: 0.2597 MAE:1.430	Acc: 0.2589 MAE: 1.3967	Acc: 0.2408 MAE: 1.246	Acc:0.2228 MAE: 1.808	Acc:0.2870 MAE:1.3412	Acc: 0.2774 MAE:1.1809
C=20 gamma=auto	Acc: 0.1968 MAE: 1.258	Acc: 0.2514 MAE:1.4346	Acc: 0.2659 MAE: 1.407	Acc: 0.2668 MAE: 1.37	Acc: 0.246 MAE: 1.210	Acc: 0.202 MAE: 1.2448	Acc:0.286 MAE:1.340	Acc: 0.2796 MAE: 1.1906

Comment: Splitting tasks into subtasks and averaging improved the model, with the best MAE of 1.169, showing that subtasks help a lot. The confusion matrix showed how well it predicted.

Table 4.17: *Confusion Matrix for Best Subtask Model at C=15 and gamma=0.001*

True/Predicted	1	2	3	5	8
1	138	123	101	98	42
2	92	78	100	117	60
3	90	99	159	163	77
5	26	47	96	172	92
8	11	14	49	122	105

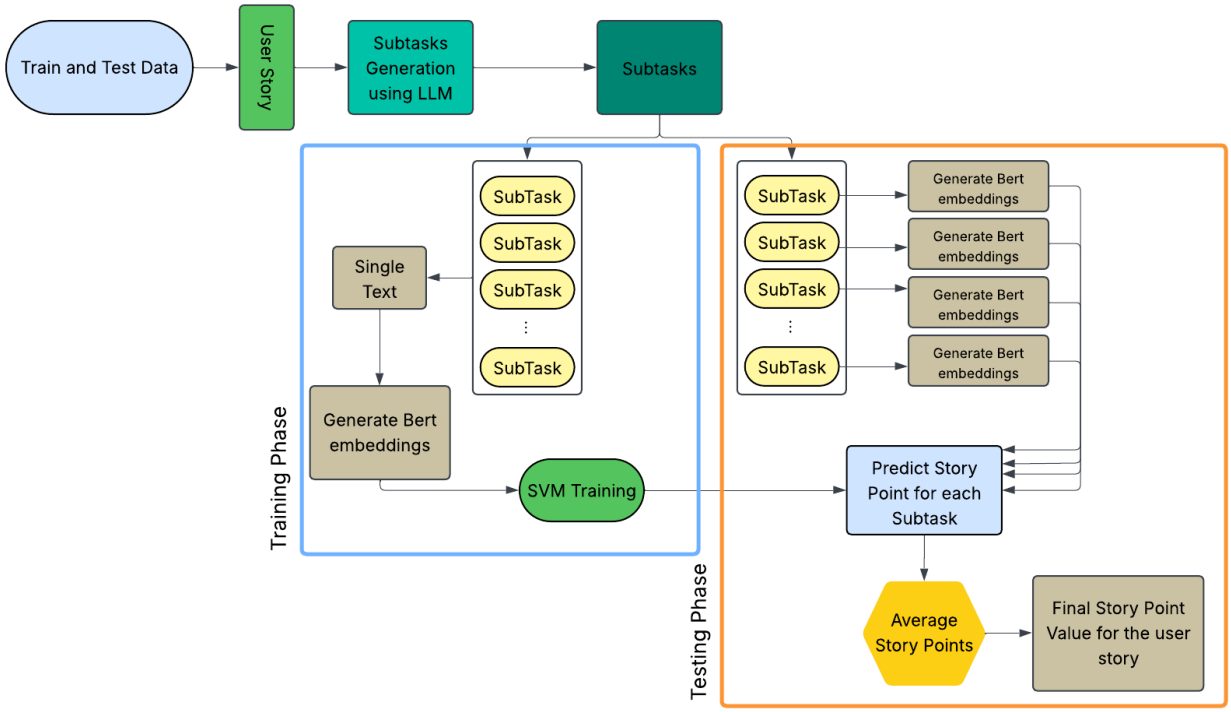

Figure 4.7: *Flow of Subtask Decomposition*

Testing on Different Projects

We tested our best approach (train the SVM using BERT embeddings, and the training text is all subtasks combined into one piece of text, and in testing, we predict the story points by averaging the points calculated for each subtask for every user story) on other projects like NEXUS, TIMOB, MESOS, and XD to see if it worked everywhere.

To further show the impact of the new approach, we also evaluated each project using **Normal SVM + BERT direct prediction on user stories**, which corresponds to Column 2 above. The subtask method usually did better.

Table 4.18: *Results Across Different Projects*

Hyperparameters	*NEXUS (Avg Subtasks)*	*NEXUS (Direct US)*	*TIMOB (Avg Subtasks)*	*TIMOB (Direct US)*	*MESOS (Avg Subtasks)*	*MESOS (Direct US)*	*XD (Avg Subtasks)*	*XD (Direct US)*
C=15, gamma=0.001	Acc: 0.309 MAE: 1.1135	Acc: 0.299 MAE: 1.201	Acc: 0.2730 MAE: 1.2442	Acc: 0.2576 MAE: 1.4243	Acc: 0.2646 MAE: 1.173	Acc: 0.2364 MAE: 1.3476	Acc: 0.287 MAE: 1.169	Acc: 0.2523 MAE: 1.4363
C=5, gamma=0.001	**Acc: 0.296 MAE: 1.1940**	**Acc: 0.303 MAE: 1.2017**	**Acc: 0.2788 MAE: 1.2375**	**Acc: 0.2688 MAE: 1.3909**	**Acc: 0.2655 MAE: 1.1668**	**Acc: 0.2246 MAE: 1.379**	**Acc: 0.282 MAE: 1.166**	**Acc: 0.2540 MAE: 1.4376**

C=1, gamma=0.01	Acc: 0.2425 MAE: 1.323	Acc: 0.2899 MAE: 1.212	Acc: 0.2872 MAE: 1.2426	Acc: 0.2655 MAE: 1.3918	Acc: 0.259 MAE: 1.163	Acc: 0.2320 MAE: 1.3455	Acc:0.2567 MAE: 1.265	Acc: 0.2536 MAE: 1.4196
C=10, gamma=auto	Acc: 0.289 MAE: 1.1565	Acc: 0.2976 MAE: 1.202	Acc: 0.275 MAE: 1.241	Acc: 0.2559 MAE: 1.427	Acc: 0.2598 MAE: 1.178	Acc: 0.236 MAE: 1.350	Acc:0.2774 MAE: 1.1809	Acc: 0.2509 MAE: 1.440
C=20, gamma=auto	Acc: 0.3065 MAE: 1.0926	Acc: 0.3042 MAE: 1.186	Acc: 0.2796 MAE: 1.238	Acc: 0.2576 MAE: 1.4285	Acc: 0.2677 MAE: 1.1429	Acc: 0.239 MAE: 1.332	Acc:0.2796 MAE: 1.1906	Acc: 0.2514 MAE: 1.4346

Comment: The subtask method (Avg Subtasks) generally had lower MAE and higher accuracy, proving it works better across projects

The best settings across projects were C=5 and gamma=0.001. Subtasks helped because:

- **Granularity**: Breaking tasks into parts made them clearer.
- **Explainability**: Teams could see why a story point value was chosen, based on the story point of each subtask and how each subtask contributed to the final estimation of the user story
- **Performance**: Averaging points reduced mistakes, though new projects were still tricky.

Transitioning to a Robust Validation Framework

Following our initial exploratory experiments across multiple projects, we thought to formalize our model selection process. To ensure our model could generalize well to new, unseen projects and to select hyperparameters in an unbiased manner, we adopted **Group K-Fold cross-validation**. This methodology is critical as it prevents data leakage by ensuring that all user stories from a single project are contained within the same fold, either entirely in the training set or entirely in the validation set. This simulates the real-world scenario of deploying the model on a new project.

We also recognized that calculating Mean Absolute Error (MAE) on arbitrary class labels (e.g., 0, 1, 2, 3, 4) was not providing a true measure of real-world impact. An error between Class 0 and Class 4 is numerically "4", but an error between a 1-point and an 8-point story is "7 points". Therefore, we shifted to calculating MAE directly on the mapped story points (1, 2, 3, 5, 8) in all the following results.

Our primary goal was to rigorously compare our two main feature extraction strategies: using the high-level text versus the granular subtasks.

Results: Text vs. Subtasks with GroupKFold Validation

We applied our GroupKFold cross-validation pipeline to the SVM model with BERT embeddings for both feature sets.

1. **Model 1: SVM with BERT on text**
    - **Test MAE:** 2.1757
    - **Test Accuracy:** 26.38%
2. **Model 2: SVM with BERT on subtasks**
    - **Test MAE:** 1.8512
    - **Test Accuracy:** 27.12%

Analysis: The model using subtasks as its input feature demonstrated a clear and significant improvement. Not only did the overall accuracy increase from **26.4% to 27.1%**, but more importantly, the Mean Absolute Error (MAE) saw a substantial reduction from **2.18 to 1.85**. This indicates that the subtask model is more accurate at exact predictions and more reliable at making predictions that are closer to the true value.

The confusion matrix for the model with subtasks is shown below.

Table 4.19: *Confusion Matrix for SVM with Subtasks (GroupKFold Validated)*

True/Predicted	***1***	***2***	***3***	***5***	***8***
1	159	134	101	82	26
2	97	95	101	122	32
3	108	127	145	162	46
5	37	62	112	147	75
8	10	25	57	139	70

Investigating Class Imbalance

Despite the improved performance of the subtask model, we observed from its confusion matrix that it still struggled to correctly identify minority classes, particularly the highest-effort classes. To address this, we investigated a standard over-sampling technique, SMOTE (Synthetic Minority Over-sampling Technique).

Experiment: The Effect of SMOTE We integrated SMOTE into our GroupKFold pipeline, ensuring it was applied only to the training data within each fold to prevent data leakage.

- **Approach:** Generated synthetic samples for minority classes.
- **Results:**
    - Test MAE: **2.4042**
    - Test Accuracy: **22.10%**

Analysis: The result of applying SMOTE was definitive and highly informative. The model's performance degraded significantly, with the accuracy dropping and the MAE increasing. The confusion matrix revealed a case of **model collapse**, where the model learned almost exclusively to predict the majority class.

Table 4.20: *Confusion Matrix for Subtask Model with SMOTE*

True/Predicted	***1***	***2***	***3***	***5***	***8***
1	502	0	0	0	0
2	447	0	0	0	0
3	587	1	0	0	0
5	433	0	0	0	0
8	300	1	0	0	0

We concluded that SMOTE introduced non-beneficial noise for high-dimensional BERT embeddings where the feature space is already complex. This experiment was critical, as it proved that the root issue is the quality and differentiability of the user story for different classes, not the number of samples.