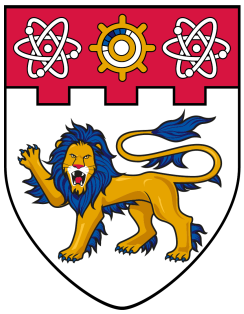

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4000 - Machine Learning

# Linking Writing Processes to Writing Quality

## Group Project Report


| Team Members | Matriculation No. |
|---|---|
| Taitsch Victor Jacques A | N2504070K |
| Ozbey Muhammed Ikbal | N2504208D |
| Kus Ahmet Bugra | N2503674F |
| Tomas Pravdik | N2503548J |
| Michael Velkoborsky | N2503549F |

**Abstract**

This report documents our approach to the *Linking Writing Processes to Writing Quality* [1]. We describe member contributions, the problem and its challenges, our progress to arriving to our final solution, experiment results and final result on the public leaderboard. We also share our lessons learned.

# Contents

# 1 Team Roles and Contributions

- **Taitsch Victor Jacques A**: Reviewed project documentation and video materials, examined model outputs, experimented with an additional LLM-based training approach, and summarized key insights to support the team.
- **Ozbey Muhammed Ikbal**: Contributed to model training and experimentation, reviewed training results, and assisted in refining model configurations.
- **Kus Ahmet Bugra**: Worked on model training and experimentation, supported evaluation procedures, and contributed to interpreting model performance.
- **Tomas Pravdik**: Worked on feature extraction process, assisted with data preprocessing, and contributed substantially to drafting, structuring, and refining the final report.
- **Michael Velkoborsky**: Collaborated on feature extraction, supported dataset preparation, and played a major role in writing, organizing, and reviewing the final project report.

# 2 Competition Overview and Ranking

## 2.1 Competition Description

In this competition, the goal is to predict essay scores on a **0–6** scale using only the *keystroke logs* recorded while each essay was written. Instead of the essay text, we receive time-stamped events (e.g., key presses, cursor positions) and actions such as `Paste`, `Replace`, `Remove/Cut`. All alphanumeric characters are anonymized to the letter `q`, so we must judge quality purely from the *writing process*. This makes the task challenging and places emphasis on careful **feature engineering**. Submissions are scored by **RMSE** on a hidden test set.

## 2.2 Evaluation Metric and Submission

Submissions are evaluated by **Root Mean Squared Error (RMSE)**, i.e. the square root of the mean of squared differences between predicted and true scores across essays. Each essay has a unique `id`, for every `id` you predict a real-valued score in $[0, 6]$. The final submission is a CSV with a header and two columns `id,score`.

## 2.3 Kaggle Score and Rank

- **Public LB RMSE**: 0.599980
- **Private LB RMSE**: 0.577114
- **Final Rank**: 1150 out of 1877 teams

- **Relative Ranking**: $\frac{1150}{1877} \approx 0.613 \Rightarrow$ top **61.3%** (better than **38.7%**)

### 2.4 Screenshots (Score and Ranking)


Figure 1: Submission (screenshot).

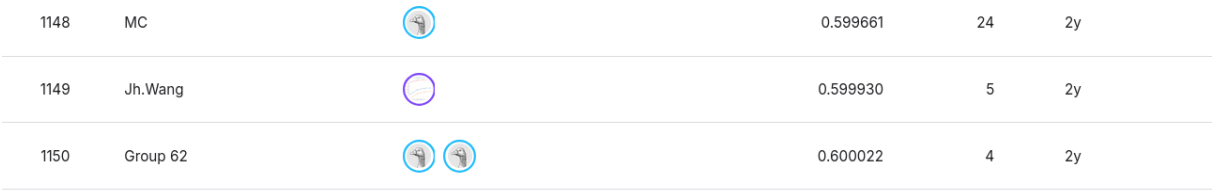


Figure 2: Public leaderboard score and final rank (screenshot).

## 3 Dataset Overview, Challenges, and Modeling Implications

### 3.1 Dataset specifications

The dataset provides event logs for each essay (`train_logs.csv`, `test_logs.csv`) and target scores per essay ID (`train_scores.csv`). Each essay's log is a chronologically ordered sequence of events with:

- **Timestamps**: `down_time`, `up_time`, `action_time`.
- **Activity** (categorical):
  - `Nonproduction` - events that *do not* alter the text (e.g., navigation).
  - `Input` - inserts new text.
  - `Remove/Cut` - deletes existing text.
  - `Paste` - inserts previously copied text.
  - `Replace` - substitutes a span (`old => new`).
  - `Move From [x1, y1] To [x2, y2]` - moves a text span from $[x_1, y_1]$ to $[x_2, y_2]$.
- **State**: `cursor_position`, `word_count`, and event labels (`down_event`/`up_event`).
- **Text change**: the textual delta caused by the event. For `Input`/`Paste` this is the inserted text; for `Remove/Cut`, the removed substring; for `Replace`, a pair `old => new`. For `Move From [x1,y1] To [x2,y2]`, it contains the moved span's text.

Alphanumeric characters are anonymized to `q`, punctuation and newlines are preserved.

## 3.2 Challenges of the problem

This task introduces a unique setting, where *event logs* with anonymized content are used in place of conventional textual features.

- **Anonymization.** The alphanumerics are masked to `q`, so the features must rely on structure (length, punctuation, layout), which limits expressiveness compared with semantic features.
- **Data size.** The usable training set is small ($\approx$ 2,471), which restricts model capacity and increases the risk of overfitting.
- **Uneven target distribution.** Scores are not uniform (see Fig. 3), so naive regressors gravitate to the majority range.
- **Feature engineering is critical.** The problem rewards well-crafted features, but selecting and calibrating them robustly is tricky and error-prone.

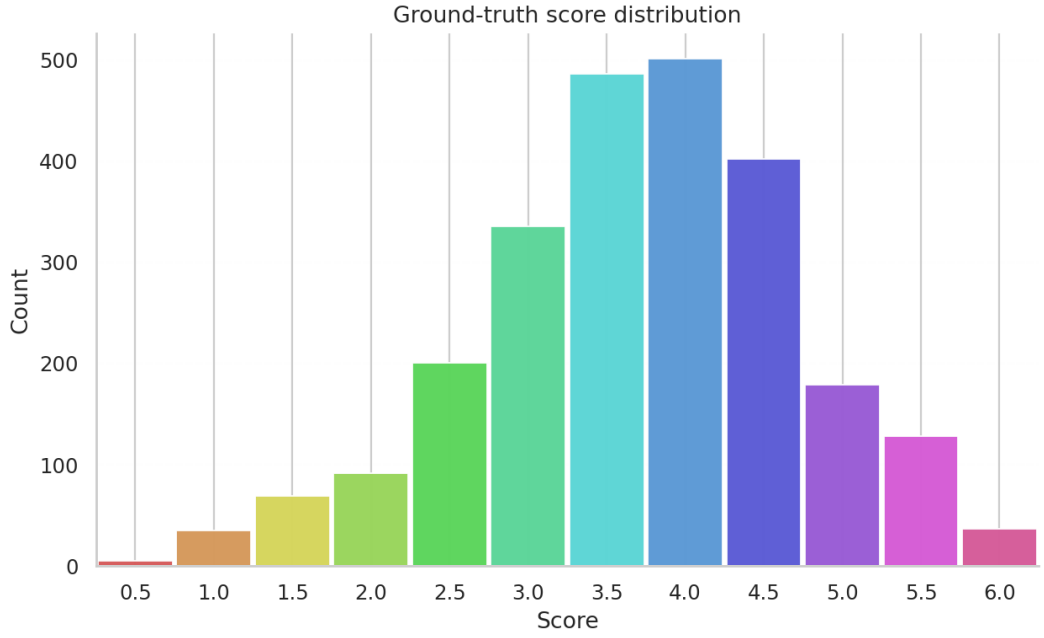


Figure 3: Ground-truth score distribution (0.5-step bins).

In short, we learn from anonymized, single-letter reconstructions with limited and imbalanced data, where results heavily rely on careful feature design.

# 4 Proposed Solution

## 4.1 Preprocessing

### 4.1.1 Introductory EDA

**Data hygiene and sanity checks.** All key columns had **0% missing**, so no imputation was needed. Timing was consistent: `action_time` = `up_time`−`down_time`. About **4.12%** of events had *non-positive holds* ($\leq 0$ ms), which could happen due to browser logging with asynchronous event handling. A IQR outlier check flagged **2 - 3%** of rows per numeric column (e.g., very long pauses or big cursor jumps). Typical sizes looked sensible: median hold **93** ms (long right tail to **447**,**470** ms) and median final `word_count` **200** (P75 = 327); see Tables 2 and 3.

Table 1: Timing sanity checks (global).

| Metric | Value | Note |
|---|---|---|
| Total events | 8,405,898 | from logs |
| Non-positive key-holds | 346,631 (4.12%) | $\texttt{up} - \texttt{down} \leq 0$ |
| Hold vs. action consistency | 0.0 ms | $\lvert\texttt{action} - (\texttt{up} - \texttt{down})\rvert$ |

Table 2: Selected distributions from numeric descriptives.

| Metric | Median | P75 | Max |
|---|---|---|---|
| Key-hold (`action_time`, ms) | 93 | 122 | 447,470 |
| `word_count` (final) | 200 | 327 | 1,326 |

Table 3: IQR outlier flags: fraction of rows flagged per numeric column.

| Column | Flag rate |
|---|---|
| `cursor_position` | 0.027235 |
| `word_count` | 0.023143 |
| `action_time` | 0.021023 |
| `down_time` | 0.003312 |
| `up_time` | 0.003312 |

### 4.1.2 Essay Reconstruction

**Goal and motivation.** We deterministically rebuild, per essay, the anonymized text buffer from the event stream using `id`, `event_id`, `activity`, `cursor_position`, and `text_change`. From the reconstructed text we derive informative, structure-based features (e.g., punctuation density, word count, line-break patterns). Because content is

anonymized to `q`, these structure/format cues are among the few reliable indications of writing quality we can utilize.

**Deterministic rules.** Let $B$ be the current text buffer (string) and $c$ the logged `cursor_position` for the current event. We use string slicing with 0-based indices.

1. **Input / Paste** ($\texttt{text\_change} = T, L = |T|$).
   Insert $T$ at $s = \max(c - L, 0)$: $\quad B \leftarrow B[0:s] \ + \ T \ + \ B[s:]$.
2. **Remove/Cut** ($\texttt{text\_change} = T, L = |T|$).
   Delete $L$ characters *starting at* $c$: $\quad B \leftarrow B[0:c] \ + \ B[c+L:]$.
3. **Replace** ($\texttt{text\_change} = \texttt{old} \Rightarrow \texttt{new}$).
   Let $s = \max(c - |\texttt{new}|, 0)$, $e = s + |\texttt{old}|$; if $e < s$ set $e = s$.
   Replace span $B[s:e)$ with `new`: $\quad B \leftarrow B[0:s] \ + \ \texttt{new} \ + \ B[e:]$.
4. **Move From [x1, y1] To [x2, y2]**.
   Treat the move as a stable reordering via slices:
   - If $x_1 < x_2$: $B \leftarrow B[0:x_1] \ + \ B[y_1:y_2] \ + \ B[x_1:y_1] \ + \ B[y_2:]$.
   - If $x_1 > x_2$: $B \leftarrow B[0:x_2] \ + \ B[x_1:y_1] \ + \ B[x_2:x_1] \ + \ B[y_1:]$.
   - If $x_1 = x_2$: no change.

Table 4: Operation semantics (exact string-slice formulas used).

| Activity | Buffer update |
|---|---|
| Input/Paste | $B \leftarrow B[0:s] + T + B[s:],\ s = \max(c - \lvert T\rvert, 0)$ |
| Remove/Cut | $B \leftarrow B[0:c] + B[c + \lvert T\rvert :]$ |
| Replace | $B \leftarrow B[0:s] + \texttt{new} + B[e:],\ s = \max(c - \lvert\texttt{new}\rvert, 0),\ e = s + \lvert\texttt{old}\rvert$ |
| Move ($x_1 < x_2$) | $B \leftarrow B[0:x_1] + B[y_1:y_2] + B[x_1:y_1] + B[y_2:]$ |
| Move ($x_1 > x_2$) | $B \leftarrow B[0:x_2] + B[x_1:y_1] + B[x_2:x_1] + B[y_1:]$ |

**Output.** For each essay we return the final reconstructed anonymized text $\hat{B}$, which we later use to derive structure-only features.

### 4.1.3 Feature Engineering from the Reconstructed Stream

**Overview.** From each essay we compute a set of process and structure features using (i) event timings (`down_time`, `up_time`, `action_time`) and (ii) the final text reconstructed from logs. These features are designed to capture *fluency*, *revision behavior*, and *document structure* - signals that remain informative even when words are anonymized to `q`. Some of the structure/edit features were inspired by community write-ups [2].

**What we compute (by family) and why.**

- **Temporal cadence and pauses (typing rhythm).** We look at how quickly and how steadily the writer types: median gaps between keystrokes (`med_IKI`), how long pauses get (`pause_p90/p95/p99`), and how variable the rhythm is (`iki_std`, `iki_cv`).
  *Why:* smoother, more regular pacing often signals fluency, very long pauses can indicate planning or getting stuck.
  *Note:* IKI = inter-keystroke interval
- **Production flow over time (when words are written).** We track how the essay accumulates over the session using cumulative shares at 10%, 20%, ..., 100% of time (`prod_d10`, `prod_d20`, ..., `prod_d100`). We also define the early/late ratio as `early_late_ratio` = `prod_d20`/(1 − `prod_d80`).
  *Why:* front-loading, steady progress, or last-minute surges reflect different planning styles. This timing pattern is informative even without seeing the content.
- **Editing behavior (how the writer revises).** We count and normalize edits (`replace_ratio`, `paste_ratio`, `move_ratio`, `edit_share`), watch when they happen (`frac_late_edits`), and capture "fixing" patterns like backspace usage and streaks.
  *Why:* purposeful revision is different from thrashing. Concentrated late edits may point to time pressure.
- **Structure from the reconstructed text (how the essay looks).** From the reconstructed essay we measure punctuation and layout (`n_dot`, `n_comma`, `n_line_breaks`, rates `p_dot`/`p_comma`), paragraph size (`mean_paragraph`), and word-length patterns (`sd_length_word`, counts like `n_q8..q10`).
  *Why:* with words anonymized to `q`, visible structure (punctuation, paragraph structure, length regularity) is one of the few remaining signals of organization and style.
- **Activity mix (balance between actions).** We summarize how time is split across actions using proportions and entropy (`act_entropy`).
  *Why:* an extreme skew (e.g., only typing or only editing) can be a red flag, a balanced mix often reflects a healthier drafting - revising cycle.

**Normalization.** We convert counts to rates where appropriate (e.g., per-word or per-minute) to compare essays of different lengths.

## 4.2 Evaluation Strategy

**Goal.** Estimate performance reliably using the same metric as the competition (RMSE) while avoiding overfitting and data leakage.

**Fold design.** We use **10-fold cross-validation** over essays and reuse the *same* train/-validation indices for all base models *Why:* consistent folds make model comparisons and stacking fair.

**Split unit.** Splits are at the **essay id** level (one row per essay in our feature table), so no essay contributes to both train and validation in the same fold.

**Metric and reporting.** For each fold we train on 9/10 and validate on 1/10, computing **RMSE**. We record per-fold RMSEs and retain **out-of-fold (OOF) predictions** for each base model.

**Hyperparameter use.** We tune each model's hyperparameters with **Optuna** [3] (automatic hyperparameter search framework) using **10-fold CV** on the training data only. We then keep those settings fixed and report results using the same 10-fold splits. For boosted trees we enable **early stopping** on the validation fold and use the *best iteration* for predictions.

**Post-processing.** Final predictions are **clipped to** $[0, 6]$ at inference to enforce the valid score range and prevent occasional overshoots, e.g., from linear regression, from inflating RMSE.

## 4.3 Models and Training

**Design principles.** Our features are tabular, heterogeneous (counts, rates, percentiles), and often interact in non-linear ways. The dataset is small ($\sim$2.5k essays) with $\sim$70 features, so we prioritize models that work well on small tabular datasets with many features.

**Simple baselines.**

- **Decision Tree** - a single decision tree as a sanity check: fast, interpretable.
- **Linear (Ridge) baseline** - a low-capacity linear model that exposes strong linear trends (e.g., document length).
- **$k$-Nearest Neighbors (kNN)** - a simple distance-based baseline that averages the targets of the closest essays in feature space (standardized inputs).

**Improving generalization: bagging, boosting, and ensembling.** Simple baselines set expectations but can miss richer dependencies among features. To improve generalization, we progressively add capacity: (1) *bagging* to reduce variance, (2) *boosting* to capture non-linear patterns and interactions, and (3) *ensembling/stacking* to combine complementary strengths across learners.

**Bagging / Boosting models.**

- **Random Forest (bagging).** Many de-correlated shallow trees averaged together. This stabilizes the noisy splits of a single tree and captures broad non-linearities with minimal tuning.
- **ExtraTrees [4] (Extremely Randomized Trees).** Similar to Random Forest but with fully random thresholds per feature at each split. It can further reduce variance and can help on small, noisy tabular sets.
- **Gradient-Boosted Trees (boosting).** We use three implementations - **XGBoost** [5], **LightGBM** [6], and **CatBoost** [7]. All build trees sequentially to correct residuals, but differ in split search, regularization, and growth strategy. This diversity yields slightly different error patterns, which is valuable when blending.

**Ensembling.** We first average each base learner's predictions across CV folds to stabilize them. Then we combine the base learners with a simple linear stacker trained on their out-of-fold predictions. At test time, we apply the same stacker.

# 5 Tools and Libraries

**Languages & runtime.** Python 3.11.11 (64-bit). Experiments run on a laptop with a GPU.

**Core data & utilities.** `pandas` (dataframes, groupby), `numpy` (vectorized ops), `scipy` (stats).

**Modeling.** `scikit-learn` (preprocessing, Ridge, DecisionTreeRegressor, RandomForestRegressor, ExtraTreesRegressor, KNeighborsRegressor, KFold, metrics), `XGBoost`, `LightGBM`, `CatBoost`, `Optuna`.

**Visualization & reporting.** `matplotlib` (figures), LaTeX (this report).

**Reproducibility.** Fixed random seeds; same 10-fold indices across models.

# 6 Experiments

## 6.1 Setup

**Data splits and metric.** We evaluate with **10-fold cross-validation** at the essay level and report **RMSE** (mean across folds).

**Feature sets.** We test three progressively richer feature sets:

- **F0** — Raw-log aggregates only (timings, pause stats, simple counts).
- **F1** — F0 + structure-from-reconstruction (punctuation, paragraphing).
- **F2** — F1 + edit/process features (late edits, bursts, backspace streaks).

## 6.2 Hyperparameters

**Shared settings.** All models use the **same 10-fold indices** and a fixed seed. Hyperparameters are tuned with **Optuna** using training data only within each fold. For gradient-boosted trees we enable **early stopping** per fold and use the best iteration. **Scale-sensitive models** (Ridge, kNN) are trained on standardized inputs. **Each feature set is tuned independently** for every model. Final predictions are clipped to $[0, 6]$ at inference.

Table 5: Key hyperparameters (tuned with Optuna per feature set).

| **Model** | **Key hyperparameters (search ranges)** |
|---|---|
| Decision Tree | `max_depth` [3-20], `min_samples_leaf` [1-50], `min_samples_split` [2-50], `max_features` [0.4-1.0]. |
| Ridge (scaled) | `alpha` [$10^{-3}$ - $10^{2}$] (log-scale); inputs standardized. |
| kNN (scaled) | `n_neighbors` [3-75], `weights` {uniform, distance}, `p` {1,2} (Manhattan/Euclidean), `leaf_size` [15-60]; inputs standardized per fold. |
| Random Forest | `n_estimators` [300-700], `max_depth` [3-20], `min_samples_leaf` [1-20], `max_features` [0.2-1.0], `bootstrap` {True, False}. |
| ExtraTrees | `n_estimators` [300-1000], `max_depth` [3-20], `min_samples_leaf` [1-20], `max_features` [0.2-1.0], `bootstrap` {True, False}. |
| XGBoost | `n_estimators` [1200-6000], `learning_rate` [0.005-0.15] (log), `max_depth` [3-10], `min_child_weight` [1.0-12.0], `subsample` [0.5-1.0], `colsample_bytree` [0.5-1.0], `reg_alpha`/`reg_lambda` [0.0-5.0]/[0.1-10.0]; early stopping. |
| LightGBM | `boosting_type` {gbdt, goss}, `num_leaves` [31-255], `learning_rate` [0.005-0.15] (log), `max_depth` [4-12], `feature_fraction` [0.6-1.0], *gbdt*: `bagging_fraction` [0.6-1.0], `bagging_freq` [1-7]; *goss*: `top_rate`/`other_rate` [0.1-0.5]; early stopping. |
| CatBoost | `iterations` [2000-9000], `learning_rate` [0.005-0.15] (log), `depth` [4-10], `l2_leaf_reg` [1.0-20.0] (log), `bagging_temperature` [0-3], `subsample` [0.5-1.0]; early stopping. |

**Why these choices.** We tune *each feature set separately* because different inputs benefit from different settings. Using the same folds and seed keeps comparisons fair. Early stopping helps prevent overfitting with minimal manual effort.

## 6.3 Single-Model Comparison across Feature Sets

**Protocol.** Each model is trained on *each* feature set (F0, F1, F2) using the same folds and seed. We report CV RMSE.

Table 6: CV RMSE across models and feature sets. *Bold* = lowest RMSE per model.

| Model | F0 | F1 | F2 |
|---|---|---|---|
| Decision Tree | 0.72719 | 0.68868 | **0.67891** |
| Linear (Ridge) | 0.69367 | 0.65795 | **0.65043** |
| KNN | 0.72746 | 0.67414 | **0.67264** |
| Random Forest | 0.67775 | 0.62608 | **0.61708** |
| Extra Trees | 0.68553 | 0.62432 | **0.61342** |
| XGBoost | 0.66421 | 0.60917 | **0.59938** |
| CatBoost | 0.65882 | 0.60749 | **0.59774** |
| LightGBM | 0.66928 | 0.61684 | **0.60379** |

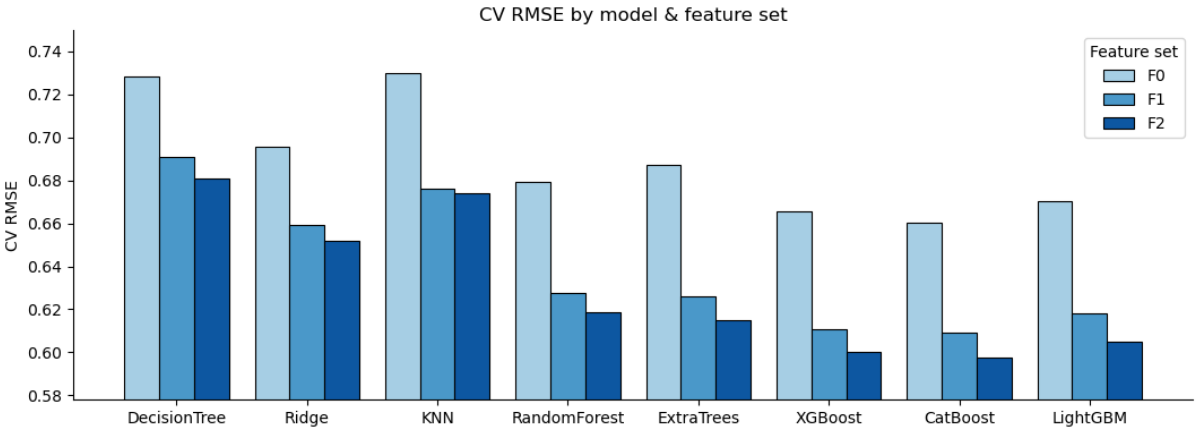


Figure 4: CV RMSE by model and feature set.

**Observation.** All models improve notably from **F0** to **F1**, and each gains further from **F1** to **F2**. The best overall scores are achieved on **F2** for every model, with **CatBoost (0.59774)** narrowly ahead of **XGBoost (0.59938)** and **LightGBM (0.60379)**. Among bagging methods, **Extra Trees (0.61342)** and **Random Forest (0.61708)** perform similarly. Simpler models (**Decision Tree**, **KNN** and **Rigde**) see consistent but smaller gains, remaining behind the boosted-tree models.

## 6.4 Ensembling

**Protocol.** We build an ensemble of three boosted models (XGBoost, LightGBM, CatBoost) and add kNN, Extra Trees, and Ridge to increase model diversity, as we presume their errors are less correlated with the boosters. All models are trained on **F2**. For each base model, we average its cross-validated predictions across folds to stabilize estimates,

and then blend the base predictions to form the final output. We experimented with the following blending strategies:

- **Unweighted average (voting):** mean of base learners.
- **Weighted average:** weights tuned using grid search (best weights: 'xgb': 0.3, 'cat': 0.35, 'lgb': 0.15, 'et': 0.08, 'ridge': 0.1, 'knn': 0.02).
- **Stacking (linear):** Ridge meta-learner fit on OOF prediction columns (10-fold CV), evaluated by OOF RMSE. We also tried `ElasticNetCV` and unpenalized `LinearRegression`, but the performed worse than ridge.

Table 7: Cross-model ensembling on F2 (CV RMSE).

| Setting | CV RMSE | Notes |
|---|---|---|
| Best single model | 0.597744 | reference |
| Unweighted average | 0.604322 | — |
| Weighted average | 0.596771 | — |
| Linear stack (OOF blend) | **0.594733** | ridge meta learner |

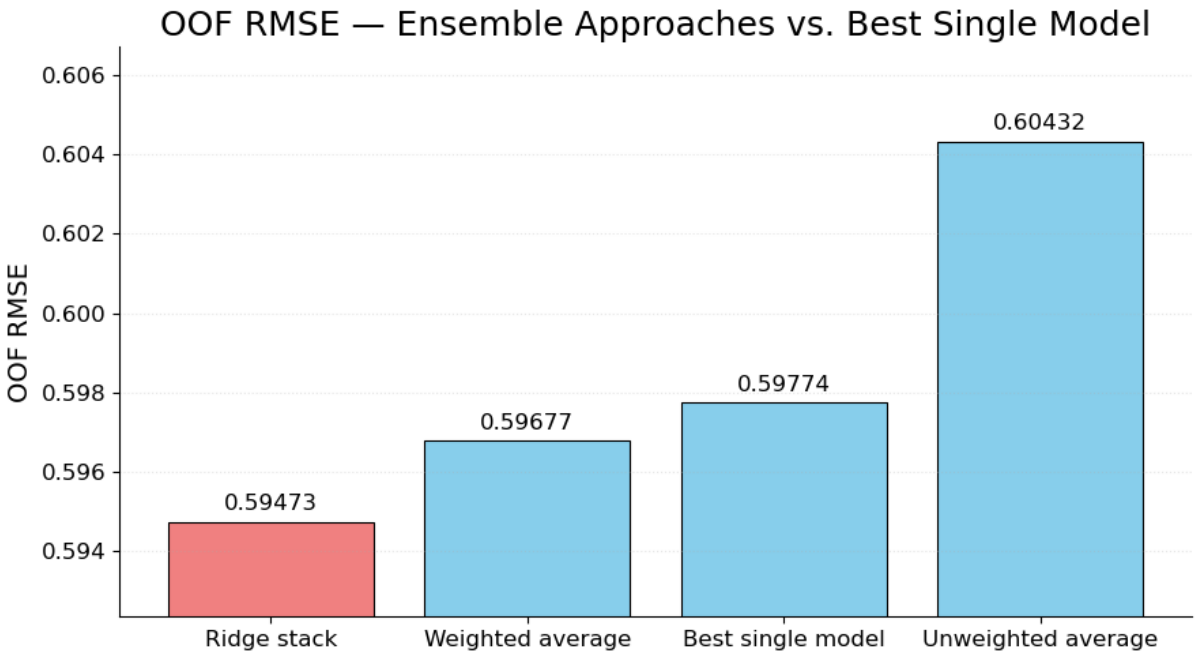


Figure 5: Best single vs. averaging vs. stacking (CV RMSE).

**Observation.** Unweighted averaging degrades performance (0.6043, worst). A tuned weighted average is only a small gain over the best single model (0.5968 vs. 0.5977; $\Delta \approx 0.001$). A ridge stack is best at **0.5947**, improving slightly by $\approx 0.003$ RMSE over the best single and $\approx 0.002$ over the weighted blend, so we adopt stacking for final results.

## 6.5 Feature Importance and Interpretation (best single model)

**Setup.** We use the **ensambled stacked model on F2** (from Table 6) and the same 10-fold CV splits/seeds as elsewhere. Importance is computed on *held-out folds only* to avoid leakage.

### 6.5.1 Permutation Importance

**Method.** For each feature $f$ and each fold, we shuffle $f$ *in the validation split only*, recompute RMSE, and define $\Delta\text{RMSE} = \text{RMSE}_{\text{shuffled}} - \text{RMSE}_{\text{original}}$. We repeat the shuffle $R = 30$ times per feature per fold and average across folds and repeats. Larger $\Delta\text{RMSE} \Rightarrow$ more important.

Table 8: Top-10 features by permutation importance (avg. $\Delta$RMSE across folds, $R = 30$).

| Rank | Feature | $\Delta$RMSE (avg) |
|---|---|---|
| 1 | `prod_total_char` | 0.109560 |
| 2 | `sd_length_word` | 0.023216 |
| 3 | `p_dot` | 0.013872 |
| 4 | `n_q7` | 0.012866 |
| 5 | `mean_input_30m` | 0.011068 |
| 6 | `n_typos_dot_comma` | 0.010765 |
| 7 | `mean_paragraph` | 0.009880 |
| 8 | `n_q10` | 0.008815 |
| 9 | `n_comma` | 0.008740 |
| 10 | `med_IKI_lag4` | 0.008160 |

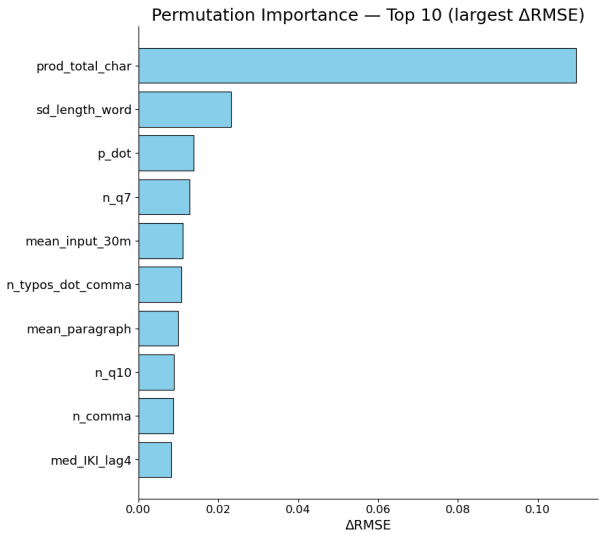


Figure 6: Top 10 features (PI)

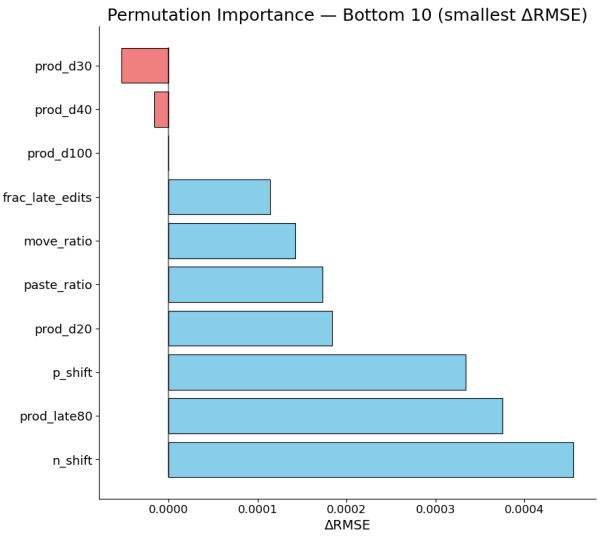


Figure 7: Bottom 10 features (PI)

Figure 8: Top and bottom 10 feature by permutation importance (bars show average $\Delta$RMSE across folds, $R = 30$). Red means negative PI.

### 6.5.2 Group LOFO by Feature Families

**Families.** We define the following families for group LOFO:

$$
\begin{aligned}
\mathcal{F}_{\text{pauses}} &= \{\texttt{med\_pause}, \texttt{n\_pause\_3s}, \texttt{pause\_p50}, \texttt{pause\_p75}, \texttt{pause\_p90}, \texttt{pause\_p95}, \texttt{pause\_p99}\},\\
\mathcal{F}_{\text{iki}} &= \{\texttt{med\_IKI}, \texttt{med\_IKI\_lag2}, \texttt{med\_IKI\_lag3}, \texttt{med\_IKI\_lag4}, \texttt{iki\_std}, \texttt{iki\_cv}\},\\
\mathcal{F}_{\text{counts}} &= \{\texttt{event\_count}, \texttt{c\_input}, \texttt{c\_remove}, \texttt{c\_paste}, \texttt{c\_replace}, \texttt{c\_move}, \texttt{edit\_cnt}, \texttt{\_total\_char}, \texttt{n\_backspace}, \texttt{n\_shift}, \texttt{burst\_input}\},\\
\mathcal{F}_{\text{rates}} &= \{\texttt{session\_min}, \texttt{events\_per\_min}, \texttt{act\_entropy}, \texttt{p\_input}, \texttt{p\_shift}, \texttt{p\_space}\},\\
\mathcal{F}_{\text{ratios}} &= \{\texttt{move\_ratio}, \texttt{replace\_ratio}, \texttt{paste\_ratio}, \texttt{edit\_share}\},\\
\mathcal{F}_{\text{prod\_curve}} &= \{\texttt{prod\_d10..d100}, \texttt{prod\_early20}, \texttt{prod\_mid50}, \texttt{prod\_late80}, \texttt{early\_late\_ratio}\},\\
\mathcal{F}_{\text{punct}} &= \{\texttt{n\_dot}, \texttt{n\_comma}, \texttt{n\_hyphen}, \texttt{n\_space}, \texttt{n\_line\_breaks}, \texttt{n\_spec\_char}, \texttt{p\_dot}, \texttt{p\_comma}, \texttt{p\_space}, \texttt{n\_typos\_dot\_comma}\},\\
\mathcal{F}_{\text{structure}} &= \{\texttt{sd\_length\_word}, \texttt{n\_q2}, \texttt{n\_q6}, \texttt{n\_q7}, \texttt{n\_q8}, \texttt{n\_q10}, \texttt{seq\_max\_min\_8\_12}, \texttt{mean\_paragraph}, \texttt{prod\_total\_char}\},\\
\mathcal{F}_{\text{timing}} &= \{\texttt{s\_action\_time}, \texttt{n\_words\_after\_20m}, \texttt{mean\_input\_30m}, \texttt{prod\_lbreak\_shift}\}.
\end{aligned}
$$

**Method.** For each family $\mathcal{F}$, we remove all features in that family, retrain the same model with the original 10-fold setup, and report $\Delta\text{RMSE} = \text{RMSE}_{\text{LOFO}(\mathcal{F})} - \text{RMSE}_{\text{full}}$. Positive $\Delta$ indicates the family contributes.

Table 9: Group-LOFO by feature family (positive ΔRMSE means the family helps).

| Family removed | ΔRMSE |
|---|---|
| Punctuation (`punct`) | +0.024045 |
| Structure (`structure`) | +0.020309 |
| Counts (`counts`) | +0.000522 |
| Pauses (`pauses`) | +0.000438 |
| Timing (`timing`) | +0.000419 |
| IKIs (`iki`) | +0.000290 |
| Rates (`rates`) | +0.000018 |
| Edit ratios (`ratios`) | −0.000751 |
| Production curve (`prod_curve`) | −0.001888 |

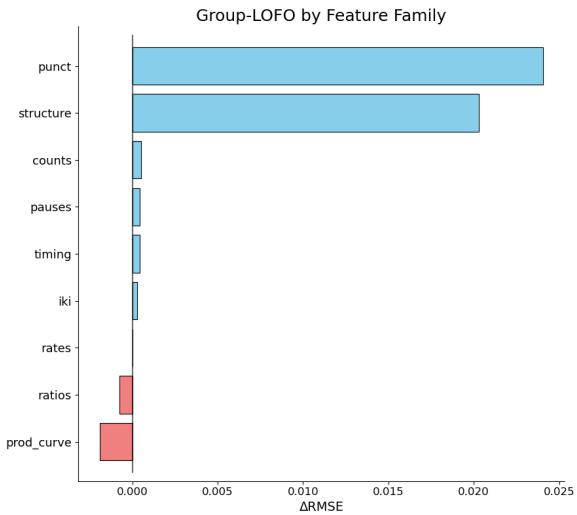


Figure 9: Group-LOFO bars (higher ΔRMSE ⇒ more contribution).

**Interpretation.** Both views tell a consistent story: *structure* and *punctuation* are the dominant sources of signal (largest positive ΔRMSE in Group-LOFO). *Tempo* (IKIs, pauses), *timing*, *counts*, and *rates* add smaller but consistent gains. In contrast, *edit ratios* and the *production curve* families offer little or no benefit (slightly negative ΔRMSE).

### 6.6 Per-Score-Bin Performance

**Setup.** Using final OOF predictions, we compute RMSE per true score bin (0 - 6) to see how our model perform for each category.

Table 10: RMSE by true score bin.

| Score bin | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| RMSE | 1.503 | 1.120 | 0.628 | 0.537 | 0.458 | 0.676 | 1.162 |
| Count $n$ | 5 | 104 | 293 | 822 | 903 | 307 | 37 |

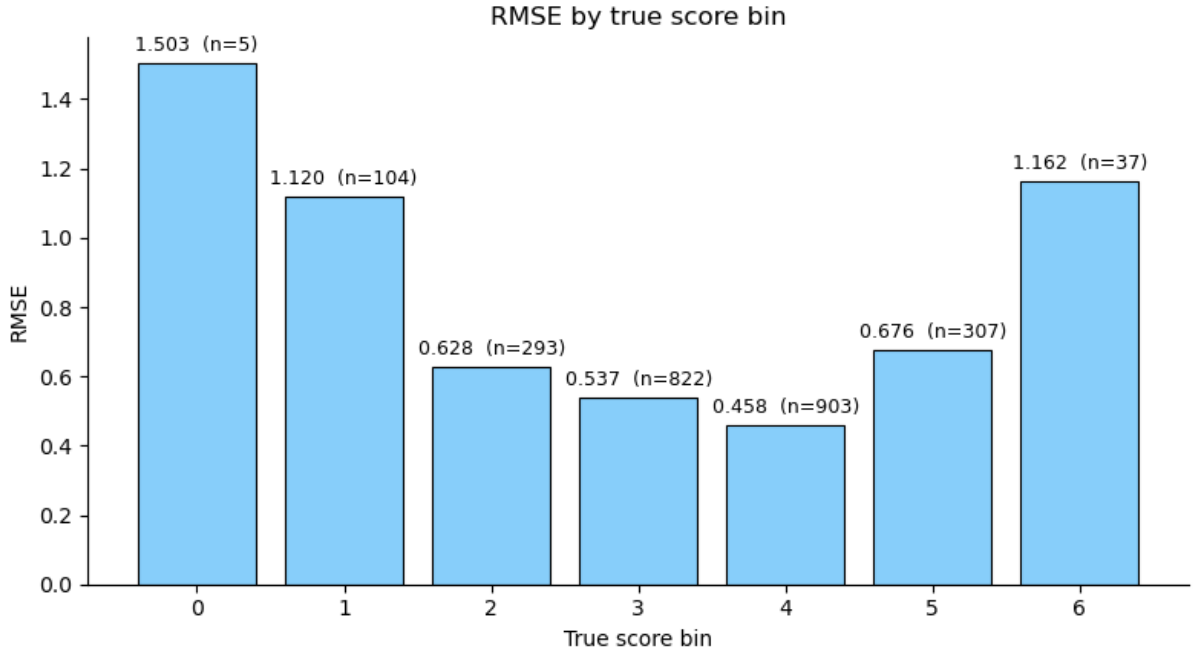


Figure 10: RMSE by true score bin.

**Observation.** Mid-range bins tend to be most stable, extremes (0, 1 and 6) have higher RMSE due to smaller ammount of samples.

## 7 Conclusion and Lessons Learned

This project was both **interesting** and **unique**: predicting essay quality from *keystroke logs* forced us to think beyond standard NLP and focus on writing *process* and document

*structure*. We worked on it intensively and, along the way, rewrote large parts of our pipeline (reconstruction logic, feature sets, and validation) multiple times to fix pitfalls and squeeze out reliable gains.

**What we learned (high level).**

- **Feature engineering matters most.** Moving from F0 $\rightarrow$ F1 $\rightarrow$ F2 showed that structure- and process-aware features dominate performance in this setting. Reconstructing the anonymized text enabled punctuation/paragraph features that carried surprising signal.
- **Boosted trees are a strong fit for small, tabular, non-linear data.** Gradient boosting (XGBoost, LightGBM, CatBoost) consistently outperformed simpler baselines, especially with careful regularization and early stopping.
- **Ensembling helps when models are complementary.** Simple averaging was not enough, but a light *stacking* layer on OOF predictions provided a small yet consistent improvement over the best single booster.
- **Interpreting models.** We computed *permutation importance* and *group LOFO* to understand where signal comes from. Both pointed to structure (length/word-length variability, punctuation, paragraphing) as key drivers, while some edit-ratio and production-curve features contributed little.
- **Error is not uniform.** Per-bin analysis showed mid-range scores are easier; extremes (very low/high) are harder due to class imbalance and limited examples. This shaped our expectations and next-step ideas.

Overall, this project taught us to turn raw logs into useful features and introduced us to boosted-tree models, which, while simple, work very well for this kind of tabular data. We were also introduced to various techniques to compute feature importance, which is helpful for analysis. It was challenging but very educational, and we learned a lot through many rewrites.

# References


[1] Kaggle. *Linking Writing Processes to Writing Quality*. 2024. URL: https://www.kaggle.com/competitions/linking-writing-processes-to-writing-quality (visited on 11/02/2025).

[2] lucaskna and shindera. *Efficiency: 1st Place Solution*. Kaggle write-up. 2024. URL: https://www.kaggle.com/competitions/linking-writing-processes-to-writing-quality/writeups/lucaskna-shindera-efficiency-1st-place-solution (visited on 11/02/2025).

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019. DOI: 10.1145/3292500.3330701.

[4] P. Geurts, D. Ernst, and L. Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63.1 (2006), pp. 3–42. DOI: 10.1007/s10994-006-6226-1.

[5] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. 2017.

[7] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. "CatBoost: Unbiased Boosting with Categorical Features". In: *Advances in Neural Information Processing Systems*. 2018.