FairNM: Fairness in Name Matching

Yuan Liu[0009−0004−5345−4898] and Flavius Frasincar(✉)[0000−0002−8031−758X]

Erasmus University Rotterdam, Burgemeester Oudlaan 50, Rotterdam 3062 PA, the Netherlands
yuanliuenzo@gmail.com, frasincar@ese.eur.nl

Abstract. Ensuring fairness in automated systems is critical in today's data-driven landscape. Prior work has exposed algorithmic bias in popular name matching systems, particularly affecting names from certain racial backgrounds, but the root causes and effective solutions remain underexplored. This paper introduces FairNM, a novel system that reduces bias through token-based similarity scoring, a Siamese Neural Network-based Short Name Module, and Name Weighting. Using a novel test bench and fairness metric, we show that FairNM achieves accurate fuzzy name matching while significantly improving fairness. Its performance is especially valuable in Web-based contexts requiring high recall and fairness, such as fraud detection, where mismatches can have serious impact.

Keywords: Fairness · Entity matching · Fuzzy name matching · NLP

1 Introduction

With the rise of machine learning, data-driven algorithms are increasingly used in high-stakes decisions, such as loans [14] and hiring [15]. A key method in these systems is Entity Matching (EM), which identifies records referring to the same real-world entity. Within EM, fuzzy name matching aims to match name variations caused by formatting differences, typos, or misspellings.

However, names often encode sensitive characteristics like race, ethnicity, or religion [12], making name-based systems vulnerable to bias. While many studies compare name matching algorithms [3,4,7,16], fairness has received little attention. One exception [10] found that popular algorithms perform worse for names of Asian origin, which was attributed to differences in name length. But, this work did not explore other linguistic factors or propose mitigation strategies.

This paper addresses those gaps by introducing FairNM, a novel system that improves fairness in fuzzy name matching without sacrificing accuracy. Unlike prior studies that classify individuals by race, we group names by linguistic origin, recognizing that language-based differences (e.g., Chinese vs. Japanese names) are more meaningful than broad racial categories. We define fairness as equal treatment across these linguistic groups.

Due to the lack of existing name datasets with language labels, we created our own, using methods from [1, 17]. The dataset and code are available at: https://github.com/yuanliueur/FairNM.

The rest of the paper continues as follows. Section 2 discusses data acquisition and cross-lingual name characteristics. Section 3 details the experimental setup and fairness metric. Section 4 presents the results and Section 5 the conclusion.

2 Data

This section describes the data used in the paper, including the construction of the name database and a descriptive analysis. Identifying bias in name characteristics across linguistic origins offers insight into potential algorithmic bias.

2.1 Name Database Generation

Due to the lack of available name databases with linguistic information, we generated our own data, inspired by previous work in ethnicity classification [1, 17]. Using a custom Web scraping tool, we collected data from Wikipedia.

Starting from a root category for a specific country on Wikipedia[1], we employed a Breadth-First Search (BFS) algorithm to retrieve all subcategories up to a depth of four. We then applied filtering criteria to these subcategories, as outlined below with examples:

1. Contains *people*: German people by occupation;
2. Contains *s of*: Members of the German Football Bond;
3. Ends with *s*: German footballers.

Non-person entries were filtered out. From each country's subtree, we extracted names at the leaf level and excluded entries unrelated to the target language. A schematic of the Chinese example is shown in Fig. 1.

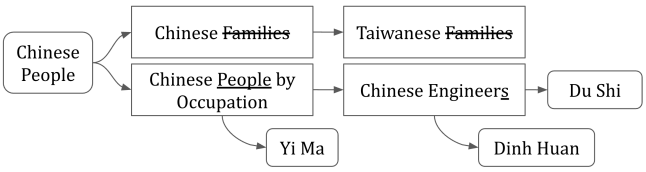

Fig. 1. Web scraping process for Chinese names, with filtered-out entries indicated.

This process was applied to various root categories, resulting in 260,150 names across 13 language groups, as detailed in Table 1.

Most names follow the Western format (first name followed by last name), but Chinese and Korean names often reverse this order. To standardize formatting, we swapped name order for these groups, except where a Vowel-Consonant-Vowel pattern or known compound surname suggested a different structure. For instance, 'Jin Baofang' is reordered to 'Baofang Jin', while 'Xiaolan Bao' remains unchanged, as the multiple vowels in 'Xiaolan' suggest a first name.

[1] The root page for 'Chinese': https://wikipedia.org/wiki/Category:Chinese_people.

Table 1. Obtained Wikipedia name database categorized per language with examples.

Language	# Names	Example Name	Root Category
BRI	57,385	Sarah Wigglesworth	British
FRA	36,683	Hervé Berville	French
GER	29,286	Magdalena Neuner	German
IND	26,146	Sukhbir Sinha	Indian
ITA	21,769	Marcello Farabegoli	Italian
RUS	17,801	Alexander Abrosimov	Russian
JAP	16,612	Toshizō Nishio	Japanese
SPA	14,462	Begoña Sánchez	Spanish
POR	13,632	Candido Rondon	Portuguese, Brazilian
CHI	13,378	Baofang Jin	Chinese
ARAB	7,036	Mohamed Al-Fayed	Egyptian, Iraq, Saudi Arabian
KOR	5,960	Seung-hyun Choi	South Korean
Total	260,150		

2.2 Data Descriptive Analysis

Two key patterns were identified across language groups in the name database: differences in name length and name uniqueness. As shown in Fig. 2, most languages exhibit name lengths between 13 to 18 characters, while Chinese and Korean names clearly peak at shorter lengths. Name uniqueness also varies substantially: Korean and Chinese surnames are far less diverse, consistent with reports that only 288 Korean surnames exist [5] and that the top five Chinese surnames are shared by 433 million people [18]. In contrast, these languages show increased variability in the first name.

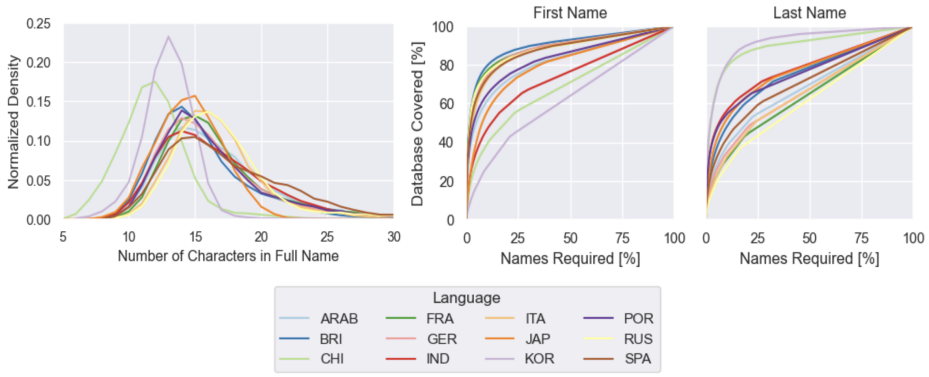

Fig. 2. Cross-lingual comparison of (left) full name length and (right) percentage of unique names required to cover database.

3 Methodology

This section describes the experimental setup and the components of the FairNM algorithm designed to enhance fairness and accuracy.

3.1 Test Bench

We simulate a fuzzy name matching task in which a query name must be linked to the correct entity within a screening list, where names are artificially modified. The test database contains 36,000 names (3,000 samples per language group). By running the test bench with screening lists of different linguistic origins, we can assess performance discrepancies across language groups. As shown in Fig. 3, the test bench consists of two steps: a High Recall Filter, which efficiently eliminates clear non-matches, and a Similarity Calculation step, where similarity scores are computed for the remaining candidate pairs using both the proposed FairNM algorithm and the normalized Levenshtein distance[2] [11], identified as the best-performing benchmark in previous works [9, 10].

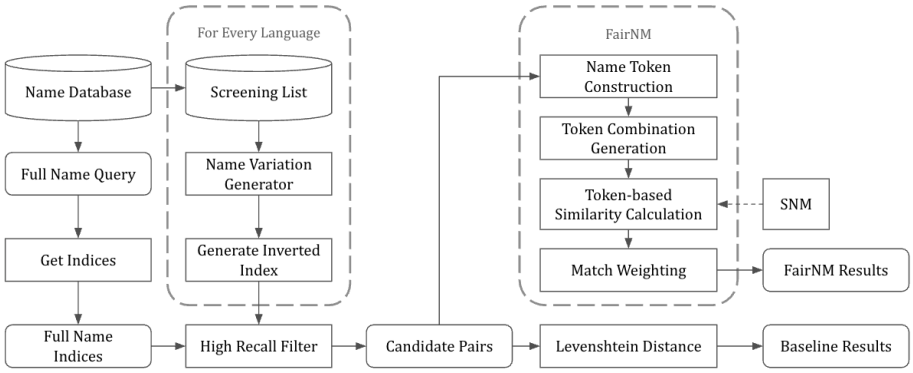

Fig. 3. Schematic overview of the test bench.

Name fuzziness is introduced through the Name Variation Generator by applying the variations proposed in [10], extended with swapped name tokens, of which examples are provided in Table 2[3].

To pre-filter candidates, we employ an inverted-index n-gram filter (using padded trigrams). Query and screening names are indexed, and only candidates with an overlap coefficient greater than 0.5 are retained for similarity scoring. This approach ensures full recall while substantially reducing computational cost.

[2] We use the RapidFuzz Python package: https://pypi.org/project/rapidfuzz/.

[3] A fat-finger error swaps a character with its neighboring key on a QWERTY keyboard.

Table 2. Example of output of name variation generator.

Name Variation	Output
Full Name	Juan de Salamanca
Swapped Names	de Salamanca Juan
Random Deletion	Jun Cerezo de Salamanca
Fat-finger Error	Juan Cerezl de Salamanca

3.2 FairNM

FairNM is a novel name matching solution that improves fairness by combining three key enhancements, as previously summarized in Fig. 3.

Token-based Similarity Calculation. We adapt a variation of the token-based approach in [17] to relieve the challenges of handling swapped and merged tokens. Given two names, we construct all name token combinations, including merged forms (e.g., matching 'JuanCerezo' with both 'Juan' and 'Cerezo'). Each pair is scored using normalized Levenshtein similarity, and an optimal one-to-one alignment is computed based on the highest scores. Merged tokens are kept only if their score exceeds all corresponding individual token matches. The resulting score is the average of the best-aligned token similarities. A full pseudocode of the alignment process is provided in Algorithm 1.

To illustrate the working principle, let us consider the matching name pair ('Ahmed Mohammed Ahmad', 'Ahmad Achmed') of which the optimal token-based normalized Levenshtein similarities are shown below.

	Ahmed	Mohammed	Ahmad
Ahmad	×	×	1.00
Achmed	0.83	×	×

The resulting similarity score is computed by averaging the matched scores, yielding $(1.0 + 0.83)/2 = 0.92$. In contrast, the conventional normalized Levenshtein similarity would have yielded a score of 0.45, whilst the similarity between the names can be considered significantly higher.

Short Name Module. Short Name Module (SNM) enhances accuracy in cases where both name tokens are shorter than four characters, as shorter tokens are more susceptible to distortion from typographical errors. For these instances, we employ a Siamese Neural Network (SNN) specifically trained on short-token similarity (see Appendix A for training details). This module is particularly effective in reducing fairness disparities for linguistic groups, such as Chinese and Korean, where shorter names are more prevalent.

Algorithm 1 optimalAlignment

Input: allSim, a dictionary which keys are the location of the assessed token in name1 (e.g., (0, 1) if merged first and second name, etc.) and the values are lists containing tuples that are sorted on simScore from high to low. The tuples are formatted as: (token1, token2, location token1, location token2, simScore).

- findNext(token) : Finds next best available match for a token in name1 and adds it to bestMatch dictionary

```
bestMatch ← {}
matchedNames ← {}
for all key, value ∈ allSim do
    token1, token2, loc1, loc2, simScore = first entry of value
    if token2 ∉ matchedNames.keys then
        bestMatch[key] = (token1, token2, loc1, loc2, simScore)
        matchedNames[token2] = key
    else                                   ▷ If we already found a match for this token2
        if simScore > bestMatch[key].get(simScore) then
            findNext(bestMatch[key].get(token1))
            bestMatch[key] = (token1, token2, loc1, loc2, simScore)
        else
            findNext(token1)
for all key ∈ bestMatch do                              ▷ Analyze merged tokens
    if len(key) > 1 then
        if sim merged tokens > sim separate tokens then
            remove separate tokens from bestMatch
        else
            remove merged tokens from bestMatch
return bestMatch
```

Match Weighting. Match Weighting aims to increase emphasize on rare names while reducing focus on common ones. To derive weights, we create an up-sampled dataset of 50,000 full names per language code. Each name is tokenized, and the frequency of each distinct token (n_t) is counted in this dataset. Hereafter, the weight (w_t) per name token is calculated using 'inverse document frequency', $w_t = \log(N/n_t)$, where N is the total number of distinct name tokens.

During similarity scoring, each matched token pair receives a combined weight based on the sum of the individual token weights. These combined weights are normalized by their total sum (w_{tot}), and the final weighted similarity score is computed as:

$$simScore = \sum_{m=1}^{M} \frac{w_{m1} + w_{m2}}{w_{tot}} \times sim_m$$

where M denotes the number of matched token pairs, w_{m1} and w_{m2} are the weights of the tokens from each name, and sim_m is the corresponding similarity

score. To illustrate, consider the following example, with token weights shown in parentheses:

	Ho-young (74.8%)	Lee (25.2%)
Hyun-Jun (72.0%)	0.36	×
Lee (28.0%)	×	1.0

This results in a weighted score of

$$\frac{0.720 + 0.748}{2} \times 0.36 + \frac{0.280 + 0.252}{2} \times 1.0 = 0.53.$$

Without weighting, the perfect match in the most common Korean surname 'Lee' would dominate the score (yielding 0.68). However, applying match weighting appropriately reduces the influence of common tokens, leading to a more balanced and fair similarity assessment.

A summarizing overview of the proposed FairNM Similarity Calculation is presented in Algorithm 2.

Algorithm 2 FairNM Similarity Calculation

Input: A name pair containing two full names, where name1 is the name with the least tokens.

- combGen(name1, name2) performs all required steps up until Token Combination Generation from Fig. 3;
- SNM(token1, token2) returns SNM token similarity;
- normLev(token1, token2) returns normalized Levenshtein token similarity;
- sortMatch(dict) returns a dictionary where the found matches per token of name1 are sorted from high to low based on their simScore;

```
allComb = combGen(name1, name2)
allSim ← {}
for all combination ∈ allComb do
    token1, loc1 = combination[0]
    token2, loc2 = combination[1]
    if (|token1| < 4 and |token2| < 3) or (|token1| < 3 and |token2| < 4) then
        sim = SNM(token1, token2)
    else
        sim = normLev(token1, token2)
    add (token1, token2, loc1, loc2, sim) to allSim
sortMatch(allSim)
bestMatch = optimalAlignment(allSim)
finalScore = matchWeight(bestMatch)
return finalScore
```

3.3 Evaluation Measures

In this section, we discuss the metrics used to assess and compare the performance and fairness of the matching algorithms. While performance evaluation in EM is well-studied, fairness measures specific to name matching are relatively scarce in literature.

Performance is assessed via F_1 scores under two conditions: a recall-restricted environment (recall > 0.98), relevant to high-stakes applications such as law enforcement or healthcare, and an unrestricted environment to reflect general contexts.

Most fairness metrics in machine learning assume binary classification, where each instance is assigned a positive or negative label [13]. In contrast, name matching or EM algorithms may return one, multiple, or no matches, making fairness assessment less straightforward.

Equalized Odds [8], a common fairness criterion, requires equal True Positive Rates (TPR) and False Positive Rates (FPR) across groups. However, in name matching, the abundance of true negatives makes satisfying the FPR condition trivial. As a result, Equalized Odds reduces to requiring equal recall (TPR), which is insufficient on its own, since many applications also demand equal precision.

To address this, we propose a stricter fairness measure combining Equalized Odds and Predictive Parity [6], requiring parity in both recall and precision across groups. It is defined as:

$$1 - \frac{\Delta p + \Delta r}{2} = 1 - \frac{1}{2}\left(\left(\max_{\forall x \in A} p_x - \min_{\forall x \in A} p_x\right) + \left(\max_{\forall x \in A} r_x - \min_{\forall x \in A} r_x\right)\right) \quad (1)$$

Here, Δp and Δr represent the largest differences in precision and recall between any two subpopulations $x \in A$, where A is the set of all groups. A fairness score near 1 indicates similar model behavior across groups, while larger disparities yield lower scores.

For example, if an algorithm yields similar precision and recall for Western, African, and Asian name groups, the fairness score approaches 1. Significant performance gaps between groups would reduce it accordingly.

4 Results

Figure 4 shows that all three FairNM components improve the $F1$ score, while Match Weighting and SNM enhance fairness. Each module contributes individually, and their combination results in both improved performance and fairness, especially in recall-restricted environments.

This is remarkable, as improving fairness often comes at the expense of other performance metrics. For example, fairness-aware decisions in a hiring decision system, such as adjusting criteria based on race, can reduce bias but may also lead to the rejection of qualified applicants who would have been approved under a purely performance-driven approach.

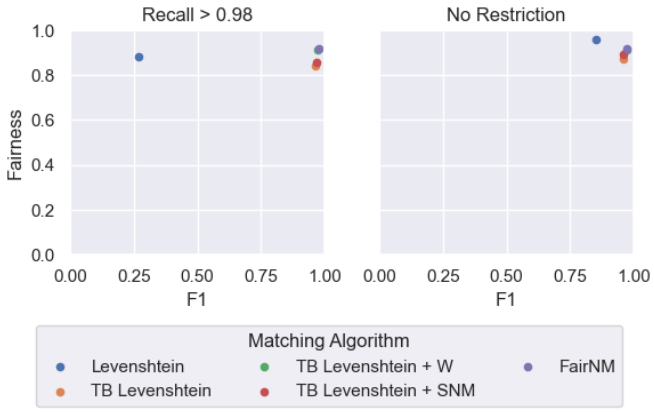

Fig. 4. Impact of FairNM modules (TB = Token-Based, W=Weighted). Dots represent results for the threshold with maximum harmonic mean between Fairness and $F1$.

In the recall-restricted setup, the token-based Levenshtein increases $F1$ but initially reduces fairness. Match Weighting and SNM mitigate this decline, with weighting offering the largest improvement. When combined, the modules show to complement each other, indicating that they tackle different fairness issues.

In the unrestricted setup, fairness drops slightly with token-based Levenshtein, but this is largely recovered (< 0.04 loss) when SNM and weighting are applied. Overall, FairNM shows to be most effective under recall constraints, which are relevant to high-stakes scenarios where fairness is often critical.

5 Conclusion

This study provides valuable insights into cross-lingual differences in name characteristics and their impact on name matching performance by categorizing person names based on their lingual origin rather than race or ethnicity. The data descriptive analysis reveals that names of Korean and Chinese origin tend to be shorter on average and display less variation in family names but more variation in first names.

A novel test bench architecture is proposed to simulate real-world scenarios of fuzzy name matching tasks, specifically screening operations. This test bench facilitates comprehensive testing of name matching across a wide spectrum of real-life name variations. Additionally, a novel fairness measure is introduced to analyze the fairness of results in name matching.

Drawing from insights gained in the data analysis and existing research, the FairNM algorithm is introduced as a novel solution. FairNM adopts a token-based approach designed to effectively manage challenges posed by missing or swapped name tokens. Further enhancements, such as the inclusion of SNM and Match Weighting, are integrated to enhance fairness in the name matching process.

Benchmarking against the normalized Levenshtein distance, the FairNM algorithm and its individual components demonstrate significant improvements in both performance and fairness, particularly in recall-constrained environments. This is particularly relevant as these constrained scenarios typically encompass high-stakes situations where fairness holds paramount importance.

While this work contributes to the understanding of fairness in the context of name matching, further research is deemed valuable to enhance this understanding. Specifically, conducting a reassessment using a non-artificial name database would help validate the proposed method in a real-life setting.

References

1. Ambekar, A., Ward, C.B., Mohammed, J., Male, S., Skiena, S.: Name-ethnicity classification from open sources. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 49–58. SIGKDD '09, ACM (2009)
2. Ardanuy, M.C., Hosseini, K., McDonough, K., Krause, A., van Strien, D., Nanni, F.: A deep learning approach to geographical candidate selection through toponym matching. In: Proceedings of the 28th International Conference on Advances in Geographic Information Systems. pp. 385–388. SIGSPATIAL '20, ACM (2020)
3. Christen, P.: Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications, Springer, 1st. edn. (2012)
4. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string metrics for matching names and records. In: Proceedings of the 2003 International Conference on Information Integration on the Web. pp. 73–78. IIWEB '03, ACM (2003)
5. Cruz, K.: A guide to Korean family names (2021)
6. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: Proceedings of the 3rd Conference on Innovations in Theoretical Computer Science. pp. 214–226. ITCS '12, ACM (2012)
7. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering **19**(1), 1–16 (2007)
8. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. pp. 3315–3323. NIPS '16, Curran Associates Inc. (2016)
9. Hosseini, K., Nanni, F., Ardanuy, M.C.: Deezymatch: A flexible deep learning approach to fuzzy string matching. In: Proceedings of the 25th Conference on Empirical Methods in Natural Language Processing. pp. 62–69. EMNLP '20, ACL (2020)
10. Karakasidis, A., Pitoura, E.: Identifying bias in name matching tasks. In: Proceedings of the 22nd International Conference on Extending Database Technology. pp. 626–629. EDBT '19, OpenProceedings.org (2019)
11. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics-Doklad **10**, 707–710 (1965)
12. Mateos, P.: A review of name-based ethnicity classification methods and their potential in population studies. Population, Space and Place **13**(4), 243–263 (2007)
13. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. ACM Computing Surveys **54**(6), 115:1–115:35 (2014)

14. Mukerjee, A., Biswas, R., Deb, K., Mathur, A.P.: Multi-objective evolutionary algorithms for the risk-return trade-off in bank loan management. International Transactions in Operational Research **9**(5), 583–597 (2002)
15. O'Keeffe, A., McCarthy, M.J.: The Routledge Handbook of Corpus Linguistics. Routledge, 1st. edn. (2010)
16. Snae, C.: A comparison and analysis of name matching algorithms. International Journal of Computer and Information Engineering **4**(1), 107–112 (2007)
17. Treeratpituk, P., Giles, C.L.: Name-ethnicity classification from open sources. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence. pp. 1141–1147. No. 1 in AAAI '12, AAAI Press (2012)
18. Yeung, J.: Why 1.2 billion people share the same 100 surnames in China (2021)

A SNN Training Setup

The setup of the utilized SNNs is inspired by a previously proposed algorithm called Deezymatch [9]. SNN consists of two identical artificial neural networks (i.e., they share their weights and biases) each capable of learning the hidden representation of a certain input name. Since the work of [2], which focused on matching location names, strongly resembles our goal, we adopt their best performing hyperparameters for this research.

To assess the performance of SNN, it is necessary to obtain representative training and test sets, where the training set can be subdivided in a training and validation part. The test set is constructed by extracting 3,000 names from each evaluated language group, specifically for the purpose of executing the previously presented test bench. Importantly, this test set is not used during the training process. To train the SNN, a large dataset consisting of name pairs categorized as either matching or non-matching is required. This dataset is obtained as follows:

1. Sample 50,000 full names, with replacement, from each language code out of the training set. We opt for a balanced training dataset in terms of language codes instead of a dataset that represents the original data distribution to reduce the chance of inherent bias in the training phase.
2. For half of the sampled names, pair each name with one of the four name variations: full name, swapped names, fat-finger error, or random deletion. This pairing results in the creation of the 'matching' training samples.
3. For the remaining half of the sampled names, 'mismatching' training samples are generated. Half of the samples are linked to a randomly selected mismatching name, whereas the other half is linked to a mismatching name with a padded tri-gram overlap coefficient of at least 0.5. This allows the model to better differentiate between non-matching names that exhibit slight similarities and those that are distinctly dissimilar.

Following the above steps, we obtain a balanced dataset consisting of 300,000 matching pairs and 300,000 non-matching pairs. This dataset can be further divided into training (90%) and validation (10%) subsets to facilitate the training of the algorithm. Note that for the short name module, solely names shorter than 4 characters are sampled.