Enhanced Merchant Name Matching Pipeline

Comprehensive Technical Documentation

Executive Summary

The Enhanced Merchant Name Matching Pipeline represents a sophisticated solution to one of the most challenging problems in financial data processing: accurately connecting abbreviated or varied merchant names with their full canonical forms. This system employs a hybrid approach that combines the semantic understanding capabilities of BERT neural networks with traditional string matching algorithms, enhanced by dynamic weighting, pattern recognition, and contextual scoring based on merchant categories.

The system achieves high accuracy by employing multiple layers of analysis:

- Text preprocessing with domain-specific knowledge
- Multiple complementary similarity algorithms
- Context-aware dynamic weighting
- Pattern recognition for common business naming conventions
- Category-based scoring adjustments
- Confidence thresholds based on business rules

This documentation provides a detailed technical overview of the system architecture, the purpose and functionality of each component, and how these elements work together to create an accurate and efficient merchant matching pipeline.

Table of Contents

System Overview

The Merchant Matching Challenge

Merchant name matching is a fundamental requirement in financial data analysis, transaction reconciliation, and fraud detection. The challenge lies in connecting different representations of the same merchant entity across various datasets, systems, and formats.

Example Challenge Scenarios:

DBAName	RawTransactionName	Challenge Type
BofA	Bank of America	Abbreviation
AMZN	Amazon.com	Non-intuitive abbreviation
MCD	McDonald's	Apostrophe and abbreviation
TRGTH	Target Fourth Street	Location-specific abbreviation
USPS	United States Postal Service	Government agency acronym

DBAName	RawTransactionName	Challenge Type
SBUX DNTN	Starbucks Downtown	Location suffix and abbreviation

Traditional exact-matching approaches fail in these scenarios, and simple string similarity measures often produce inaccurate results. Our system addresses these challenges through a multi-faceted approach that mimics human recognition processes by considering character-level similarity, phonetic similarity, semantic meaning, and contextual information.

System Architecture

The merchant matching pipeline follows a modular architecture with distinct components that work together to provide accurate matching capabilities:

```
[Input Data] → [Preprocessing] → [Similarity Calculation] → [Weight Management] → [Score Computation] → [Match Categ
```

Data Flow

1. **Input**: Merchant data containing DBANames (potentially abbreviated) and RawTransactionNames (full names), along with categorical information.
2. **Preprocessing**: Names are cleaned, normalized, and standardized based on domain-specific rules.
3. **Similarity Analysis**: Multiple algorithms compute various aspects of similarity between name pairs.
4. **Dynamic Weighting**: Weights are assigned to each algorithm based on name characteristics and merchant categories.
5. **Score Computation**: A weighted combination of similarity scores is computed, then enhanced with pattern detection and contextual rules.
6. **Output**: Pairs are assigned final similarity scores and categorized according to confidence thresholds.

Core Components

BERT Embedder

At the heart of the system's semantic understanding capability is the EnhancedBERTEmbedder class, which leverages the powerful sentence-transformers/all-mpnet-base-v2 model to capture the meaning of merchant names beyond simple keyword matching.

How BERT Embeddings Work

BERT (Bidirectional Encoder Representations from Transformers) converts text into numerical vectors (embeddings) that represent the semantic meaning of words and phrases. The model has been pre-trained on billions of words and can understand:

- Contextual relationships between words
- Synonyms and related concepts
- Industry-specific terminology
- Semantic connections that transcend literal text matching

Example: The BERT model can understand that "AAPL" and "maker of iPhone" are related even though they share no characters in common, because it has learned the semantic connection between Apple Inc. and its products.

Embedding Process

1. Text tokenization: The merchant name is split into tokens (words or subwords)
2. Token encoding: Each token is converted to numeric IDs
3. Model processing: The BERT model processes these IDs in context
4. Pooling: Token-level embeddings are combined into a single sentence-level embedding
5. Output: A high-dimensional vector (typically 768 dimensions) representing the name's meaning

Fallback Mechanism

The system includes a graceful degradation strategy: if the BERT model is unavailable, it falls back to TF-IDF (Term Frequency-Inverse Document Frequency) vectorization using character n-grams. While less powerful than BERT, this approach still captures important character-level patterns in merchant names.

Merchant Matcher

The EnhancedMerchantMatcher class forms the core framework of the system, providing preprocessing functionality and domain-specific knowledge about merchant naming conventions.

Key Knowledge Resources

Okay, let's break down these questions.

1. **Why push sample data into BERT? (Fine-tuning/Adaptation)**
    - **Clarification:** First, it's important to note that the specific code you provided (`spfinalmain.txt`) *does not* actually fine-tune or push new sample data into the BERT model for retraining. It uses a pre-trained model called `sentence-transformers/all-mpnet-base-v2` directly off-the-shelf.
    - **Why is fine-tuning sometimes done?** The general idea behind fine-tuning (training a pre-trained model on new, specific data) is called **domain adaptation**. Pre-trained models like basic BERT are trained on vast amounts of general text (like Wikipedia and books). While they understand language broadly, they might not perform optimally on specialized tasks or text types (e.g., legal documents, medical notes, or even specific merchant name patterns) because the language, jargon, and context can be different.
    - **Benefits of Fine-tuning:** By training the model further (fine-tuning) on data *specific* to your domain or task, you can adapt its "knowledge" and improve its performance on that specific task. It helps the model better understand domain-specific vocabulary, nuances, and the desired output format.
    - **Why not done here?** The `sentence-transformers` models (like the `all-mpnet-base-v2` used in the script) are often *already* fine-tuned specifically for sentence and phrase similarity tasks. Therefore, for the goal of comparing merchant name similarity, this pre-trained model might already be well-suited, potentially making additional fine-tuning unnecessary for achieving good results in this specific script's context.
2. **Why the need for dictionaries? Can it work without them?**
    - **Purpose:** The dictionaries in the code (`self.abbreviations` , `self.merchant_category_abbreviations` , and the stopword lists) are used for **rule-based preprocessing**. They encode explicit, human-defined knowledge.
        - **Abbreviation Expansion:** They map known, common short forms or acronyms (like 'kfc', 'bofa', 'dr', 'dept') directly to their full names ('kentucky fried chicken', 'bank of america', 'doctor', 'department').
        - **Stopword Removal:** They list common words ('the', 'and', 'inc', 'llc') or category-specific words ('healthcare', 'financial', 'restaurant') that often don't help distinguish between merchants and can be considered noise. Removing them helps focus the analysis on the more meaningful parts of the name.
    - **Why not work without them? Possible Flaws:**
        - **Noise Interference:** Stopwords can interfere with simpler similarity algorithms (like TF-IDF or basic string metrics) and potentially add slight noise even to BERT comparisons. Removing them leads to cleaner input for the algorithms.
        - **Loss of Explicit Control:** Dictionaries provide a way to inject direct, reliable, human-curated knowledge into the system. Relying solely on a machine learning model makes the system entirely dependent on what the model learned (or didn't learn) from its training data, which might miss specific, important real-world variations like common abbreviations.
        - **Hybrid Approach:** Combining rule-based methods (like dictionaries) with machine learning (like BERT) is a standard and often very effective strategy in NLP. It leverages the strengths of both: rules provide precision and handle known cases reliably, while ML provides generalization and semantic understanding. Your script uses this hybrid approach.
3. **Proof of Concept Links:**
    - **BERT Fine-tuning / Domain Adaptation:**
        - Hugging Face Course on Fine-tuning: https://huggingface.co/learn/nlp-course/chapter7/3
        - Deepset Blog on BERT Models (mentions fine-tuning and adaptation): https://www.deepset.ai/blog/the-definitive-guide-to-bertmodels
        - Towards Data Science article on Fine-tuning: https://towardsdatascience.com/stepping-out-of-the-comfort-zone-through-domain-adaptation-a-deep-dive-into-dynamic-prompting-4860c6d16224/
        - Sentence Transformers Domain Adaptation Docs: https://sbert.net/examples/domain_adaptation/README.html
    - **Hybrid NLP (Rules + Machine Learning):**
        - Lexalytics Blog discussing Hybrid ML Systems: https://www.lexalytics.com/blog/machine-learning-natural-language-processing/
        - Algoscale article explaining Hybrid Approach: https://algoscale.com/blog/what-is-hybrid-approach-in-nlp/
        - ML6 Blog on combining NLP and Regex (Rules): https://www.ml6.eu/blogpost/hybrid-machine-learning-marrying-nlp-and-regex
        - Research paper on combining ML and Rule-based systems for Text Categorization: https://cdn.aaai.org/ocs/2532/2532-11166-1-PB.pdf
        - Survey paper on Hybrid Approaches: https://arxiv.org/html/2401.11972v2

The matcher incorporates extensive domain expertise through several knowledge bases:

Abbreviation Dictionaries: Maps common abbreviations to their expanded forms across various domains:

```
{
    'bofa': 'bank of america',
    'b of a': 'bank of america',
    'amex': 'american express',
```

```
    'sbux': 'starbucks',
    # Hundreds more entries...
}
```

Category-Specific Abbreviations: Specialized abbreviations for different merchant categories:

```
{
    'Medical': {
        'dr': 'doctor',
        'hosp': 'hospital',
        # More medical abbreviations...
    },
    'Financial': {
        'fin': 'financial',
        'svcs': 'services',
        # More financial abbreviations...
    },
    # Other categories...
}
```

Stopwords: Common words that add little value to matching and should be removed:

```
{
    'inc', 'llc', 'co', 'ltd', 'corp', 'plc', 'na', 'the',
    'and', 'of', 'for', 'in', 'a', 'an', 'by', 'to', 'at',
    # More stopwords...
}
```

Category-Specific Stopwords: Words that are common and non-distinctive within specific categories:

```
{
    'Medical': {'center', 'healthcare', 'medical', 'health', 'care', 'services', 'clinic', 'hospital'},
    'Government': {'department', 'office', 'agency', 'bureau', 'division', 'authority', 'administration'},
    # Other categories...
}
```

Preprocessing Logic

The enhanced_preprocessing method performs crucial text normalization:

1. **Lowercase conversion**: Standardizes case for comparison
2. **Punctuation handling**: Carefully removes non-essential punctuation while preserving meaningful characters
3. **Special character normalization**: Handles apostrophes and other special cases
4. **Business suffix expansion/removal**: Standardizes company designations like "Inc." and "LLC"
5. **Abbreviation expansion**: Applies general and category-specific abbreviation dictionaries
6. **Special case handling**: Specific rules for common patterns like "McDonald's" variations
7. **Stopword removal**: Eliminates common words that don't contribute to matching

This preprocessing ensures names are in a standardized format before similarity calculations, dramatically improving matching accuracy.

Similarity Algorithms

The EnhancedMerchantMatcherWithSimilarity class implements multiple complementary similarity algorithms, each capturing different aspects of name similarity. This multi-algorithm approach provides a more comprehensive similarity assessment than any single algorithm could achieve.

Algorithm Portfolio

Algorithm	Purpose	Strengths	Example
Jaro-Winkler	Character-level similarity with prefix emphasis	Good for typos and variations in short strings	"Amzn" vs "Amazon"
Damerau-Levenshtein	Edit distance with transposition handling	Catches transposed character errors	"Wlmart" vs "Walmart"

Algorithm	Purpose	Strengths	Example
TF-IDF Cosine	Keyword-based similarity	Focuses on important distinctive terms	"Bank of America ATM" vs "Bank of America Branch"
Jaccard Bigram	Character-level structural similarity	Identifies similar character patterns	"McD" vs "McDo"
Soundex	Phonetic similarity	Matches names that sound similar	"Smith" vs "Smyth"
Token Sort Ratio	Word-order invariant comparison	Handles word reordering	"Pizza Hut" vs "Hut Pizza"
Contains Ratio	Substring containment	Identifies when one name contains the other	"Walmart" vs "Walmart Supercenter"
Fuzzy Levenshtein	Standard edit distance similarity	General-purpose string similarity	"Target" vs "Targt"
Trie Approximate	Acronym detection	Identifies first-letter acronyms	"BOA" vs "Bank of America"
Aho-Corasick	Character containment	Efficiently finds character overlaps	"WF" vs "Wells Fargo"
BERT Similarity	Semantic understanding	Captures meaning relationships	"Apple" vs "iPhone Maker"
DBAName Formation	Acronym pattern recognition	Multiple approaches to acronym detection	"USPS" vs "United States Postal Service"

Each algorithm produces a similarity score between 0 and 1, where higher scores indicate greater similarity according to that particular method.

Weight Management

The system implements a sophisticated dynamic weighting mechanism that adjusts the importance of each algorithm based on the characteristics of the merchant names being compared.

Base Weights

Each algorithm starts with a base weight reflecting its general reliability for merchant name matching:

```
self.base_weights = {
    'jaro_winkler': 0.10,
    'damerau_levenshtein': 0.05,
    'tfidf_cosine': 0.05,
    'jaccard_bigram': 0.05,
    'soundex': 0.05,
    'token_sort_ratio': 0.10,
    'contains_ratio': 0.10,
    'fuzzy_levenshtein': 0.05,
    'trie_approximate': 0.10,
    'bert_similarity': 0.15,
    'aho_corasick': 0.05,
    'DBAName_formation': 0.15
}
```

BERT similarity and DBAName formation receive the highest weights (0.15 each) due to their effectiveness in handling semantic relationships and acronym patterns, which are particularly common in merchant names.

Dynamic Weight Adjustment

The get_dynamic_weights method adjusts these base weights according to several factors:

Name Length Adjustments:

- For very short DBANames (2-3 characters), likely to be acronyms, increase weights for DBAName_formation (0.25), enhanced_DBAName_formation (0.20), bert_similarity (0.15), and contains_ratio (0.15)
- For longer DBANames (≥4 characters), increase weights for bert_similarity (0.25) and token_sort_ratio (0.15)
- For very long RawTransactionNames (>30 characters), increase weights for bert_similarity (0.30) and tfidf_cosine (0.15)

Keyword-Based Adjustments:

- When names contain banking terms, increase weights for bert_similarity and DBAName_formation
- When names contain location indicators (east, west, north, south), increase token_sort_ratio and bert_similarity

Enhanced Merchant Name Matching System: Technical Documentation

Executive Summary

This document details a sophisticated merchant name matching system designed to accurately match abbreviated business names (DBANames) with their full transaction names. The system utilizes advanced natural language processing techniques, multiple similarity algorithms, and contextual business understanding to achieve high matching accuracy even with challenging merchant name variations.

The solution addresses a common challenge in financial transaction processing: reliably connecting abbreviated merchant names that appear in transaction data with their complete business names, enhancing transaction categorization, spend analytics, and customer experience.

Business Problem

Financial institutions and payment processors frequently encounter merchant names in multiple formats:

- **DBANames**: Short forms or abbreviations used in transaction records (e.g., "AMZN", "SBUX")
- **Full Transaction Names**: Complete business names (e.g., "Amazon.com", "Starbucks Coffee #123")

Accurately mapping between these variations is critical for:

1. **Transaction categorization and analytics**
2. **Customer-facing transaction descriptions**
3. **Merchant reconciliation and fraud detection**
4. **Regulatory reporting and compliance**

Traditional approaches using simple string matching often fail due to:

- Inconsistent abbreviation patterns
- Industry-specific naming conventions
- Variations in business entity suffixes (Inc., LLC, etc.)
- Regional prefixes/suffixes
- Punctuation and formatting differences

Technical Solution

Our Enhanced Merchant Matcher implements a multi-algorithm approach with contextual awareness to address these challenges:

Key Features

1. **Advanced NLP Techniques**:
   - BERT-based semantic understanding with MPNet model
   - Fallback to TF-IDF vectorization when transformers aren't available
   - Named entity recognition and pattern matching
2. **Multiple Similarity Algorithms**:
   - Phonetic similarity (Soundex)
   - Edit distance (Damerau-Levenshtein, Jaro-Winkler)
   - Token-based similarity (token sort ratio)
   - N-gram similarity (Jaccard bigram)
   - Semantic similarity (BERT embeddings)
   - Pattern recognition (acronym detection, word order variations)
3. **Business Context Awareness**:
   - Merchant category-specific processing
   - Industry-specific abbreviation dictionaries
   - Dynamic algorithm weighting based on input characteristics
   - Pattern detection for common business naming conventions

4. **Optimization Features**:
    - Batch processing with progress tracking
    - GPU acceleration when available
    - Comprehensive result analysis and visualization

System Architecture

The system consists of four primary components:

1. Enhanced BERT Embedder

```
EnhancedBERTEmbedder
├── Model initialization (MPNet or TF-IDF fallback)
├── Text encoding with different pooling strategies
├── Merchant category adaptation
└── Similarity computation
```

2. Merchant Matcher Core

```
EnhancedMerchantMatcher
├── Abbreviation dictionaries (general and category-specific)
├── Stopword lists (general and category-specific)
├── Enhanced text preprocessing
└── Context-aware name pair preprocessing
```

3. Similarity Methods

```
EnhancedMerchantMatcherWithSimilarity
├── String-based methods (Jaro-Winkler, Levenshtein, etc.)
├── Token-based methods (token sort ratio, containment)
├── Phonetic methods (Soundex)
├── Semantic methods (BERT embeddings)
├── Pattern detection (complex business patterns)
├── Dynamic weighting based on input characteristics
└── Contextual scoring with category constraints
```

4. Processing Pipeline

```
OptimizedMerchantMatcher and Processing Functions
├── Data standardization and preprocessing
├── Batch processing with progress tracking
├── Comprehensive result analysis
└── Visualization and reporting
```

Algorithms and Methods

Text Preprocessing

The system employs sophisticated preprocessing tailored to business names:

1. **Enhanced tokenization** with special handling for business-specific tokens
2. **Abbreviation expansion** using comprehensive dictionaries
3. **Stop word removal** with category-specific considerations
4. **Business suffix normalization** (Inc., LLC, Corp., etc.)
5. **Special handling for apostrophes** in business names (e.g., McDonald's → McDonalds)

Similarity Calculation

The system calculates similarity using 12+ distinct algorithms, each capturing different aspects of name similarity:

Algorithm	Focus	Strength
Jaro-Winkler	Character similarity with position weighting	Common prefixes
Damerau-Levenshtein	Edit distance with transpositions	Typos and reordering
TF-IDF Cosine	Term frequency with inverse document frequency	Keyword matching
Jaccard Bigram	Character bigram overlap	Partial matches
Soundex	Phonetic encoding	Similar-sounding names
Token Sort Ratio	Normalized token comparison	Word order differences
Contains Ratio	Substring containment	Abbreviation detection
Fuzzy Levenshtein	Normalized edit distance	General string similarity
Trie Approximate	First-letter matching	Acronym detection
Aho-Corasick	Efficient pattern matching	Character overlap
BERT Similarity	Semantic vector space similarity	Meaning-based matching
DBAName Formation	Custom acronym formation scoring	Business abbreviation patterns

Dynamic Weighting

Instead of fixed weights, the system dynamically adjusts algorithm importance based on:

1. **Merchant name characteristics** (length, structure, etc.)
2. **Merchant category** (banking, restaurant, retail, etc.)
3. **Pattern detection** (word order inversions, acronym matching, etc.)

For example:

- Short DBANames (2-3 chars) increase weight for acronym formation metrics
- Financial institutions prioritize acronym formation and semantic understanding
- Restaurants give higher weights to phonetic and fuzzy matching
- Location indicators (east, west, etc.) boost token-based algorithms

Context-Aware Scoring

The final matching score is calculated with business-specific constraints:

1. **Base weighted score** from all similarity algorithms
2. **Pattern-based boosting** when specific business name patterns are detected
3. **Category constraint application**:
    - If categories match and name similarity is good: score exceeds 0.75
    - If categories don't match: score is capped at 0.75

Implementation Details

Dependencies

The system requires the following key libraries:

- pandas & numpy for data handling
- torch for neural network operations
- Levenshtein, textdistance, fuzzywuzzy for text similarity
- scikit-learn for TF-IDF vectorization
- transformers for BERT embeddings (optional)
- pyahocorasick for pattern matching (optional with fallback)
- matplotlib & seaborn for visualization

Hardware Requirements

- Works on CPU but benefits from GPU acceleration
- Memory requirements scale with batch size and data volume

Primary Interfaces

The system exposes several key interfaces:

1. **Initialization**:

```
bert_embedder = EnhancedBERTEmbedder()
merchant_matcher = OptimizedMerchantMatcher(bert_embedder)
```

2. **Single Pair Matching**:

```
score = merchant_matcher.compute_contextual_score(
    DBAName, DBA_Merchant_Category,
    RawTransactionName, RawTransaction_Merchant_Category
)
```

3. **Batch Processing**:

```
results_df = run_merchant_pipeline(input_file, output_file)
```

4. **Analysis**:

```
comprehensive_results = run_comprehensive_analysis(input_file, merchant_matcher)
```

Performance and Results

Scoring System

The match quality is categorized by score thresholds:

Category	Score Range	Interpretation
Excellent	0.95-1.0	Exact match with high confidence
Very High	0.85-0.95	Strong match with minimal variations
High	0.75-0.85	Good match with some variations
Medium	0.65-0.75	Probable match requiring verification
Moderate	0.5-0.65	Possible match with significant variations
Low	0.0-0.5	Unlikely match

Outputs and Visualization

The system generates:

1. **Detailed match results** with scores for each algorithm
2. **Score distribution analysis** across the entire dataset
3. **Category-based performance metrics**
4. **Visualizations** including histograms and boxplots

Use Cases and Examples

Banking and Financial Institutions

Example: "BOA" → "Bank of America"

- Leverages DBAName formation patterns
- Uses financial institution-specific abbreviations
- Applies category constraints for validation

Retail and E-commerce

Example: "AMZN" → "Amazon.com Marketplace"

- Utilizes semantic understanding for marketplace variations
- Handles regional and store number variations
- Recognizes e-commerce specific patterns

Restaurant and Food Services

Example: "MCD" → "McDonald's Restaurant #1234"

- Applies phonetic algorithms for brand names
- Handles apostrophes and franchise locations
- Normalizes location indicators

Implementation Plan

1. **Environment Setup**:
   - Install required dependencies
   - Configure hardware resources (CPU/GPU)
2. **Data Preparation**:
   - Standardize input data format
   - Ensure DBAName and full name columns are properly defined
   - Add merchant category information when available
3. **System Deployment**:
   - Initialize the matcher with appropriate configurations
   - Process data in batches for large datasets
   - Generate and analyze results
4. **Evaluation and Tuning**:
   - Review score distributions
   - Analyze performance by merchant category
   - Fine-tune algorithm weights if needed

Conclusion

The Enhanced Merchant Name Matching system provides a sophisticated solution to the challenging problem of connecting abbreviated business names with their full transaction descriptions. By leveraging multiple similarity algorithms, contextual business understanding, and advanced NLP techniques, the system achieves high matching accuracy across diverse merchant categories and naming patterns.

This solution enables:

- Improved transaction categorization
- Enhanced customer experience with clear merchant descriptions
- Better spend analytics and reporting
- Reduced manual reconciliation effort

The modular architecture allows for continued enhancement and adaptation to specific business domains and use cases.

Appendix: Technical Details

BERT Model Configuration

The system uses the MPNet base model (`sentence-transformers/all-mpnet-base-v2`) which provides:

- 768-dimensional embeddings
- State-of-the-art semantic understanding

- Effective representation of short business names
- Superior performance compared to vanilla BERT models

Dynamic Weight Adjustment Logic

The weight adjustment follows this pattern:

```
# For very short DBANames (2-3 chars)
weights['DBAName_formation'] = 0.25
weights['enhanced_DBAName_formation'] = 0.20
weights['bert_similarity'] = 0.15

# For longer DBANames
weights['bert_similarity'] = 0.25
weights['token_sort_ratio'] = 0.15
```

Complex Pattern Detection

The system recognizes business-specific patterns:

- Word order inversions (e.g., "America Bank" vs "Bank of America")
- Connector word acronyms (e.g., "B&T" from "Bread & Tulips")
- Complete and partial acronyms (e.g., "IBM" from "International Business Machines")
- Descriptor variations (e.g., "Chase" vs "Chase Manhattan")
- Significant substring matches

These patterns boost matching scores for recognized business name formats.

Conclusion

The Enhanced Merchant Name Matching Pipeline represents a sophisticated approach to the challenging problem of connecting abbreviated merchant names with their full canonical forms. By combining the semantic understanding capabilities of BERT with traditional string matching techniques, and enhancing these with domain-specific knowledge, the system achieves high accuracy across diverse merchant naming patterns.

The multi-layered approach—from preprocessing and similarity calculation to dynamic weighting, pattern recognition, and contextual scoring—ensures robust performance even when dealing with complex abbreviations, industry-specific terminology, and unusual naming conventions.

Future enhancements could include:

- Integration with machine learning for weight optimization
- Additional specialized algorithms for emerging merchant categories
- Extended pattern recognition capabilities
- Real-time feedback integration for continuous improvement
- Parallel processing for larger datasets

This system demonstrates the power of combining modern AI techniques with domain expertise to solve complex real-world data matching challenges.