On-the-Fly Deduplication and Classification of Products in Large-Scale Data Ingestion Pipelines

Troy Kopec, Creed McFall, Carlos Mejia, Zachary Meyer

ABSTRACT

Duplicate entries in product databases can lead to inaccurate analytics, degraded user experiences, and increased operational costs. Most deduplication systems address this problem only after it occurs—running as periodic batch processes that clean data retrospectively. This delayed approach allows inconsistencies to persist and propagate.

Recent research in entity resolution has explored advanced methods like clustering (Martinek et al., 2023), neural hashing (Wang et al., 2024), and deep semantic matching (Li et al., 2022). However, these techniques are often complex, resource-intensive, and rarely integrated directly into the data ingestion pipeline. Even powerful systems like BoostER (Li et al., 2024) and FlexER (Genossar et al., 2023) are designed for post-hoc deduplication, leaving a gap between academic advances and real-world deployment.

Our goal is to bridge this gap by introducing a lightweight, Python-based product deduplication model that operates at the point of data entry—before duplicates enter the system. Our solution uses TF-IDF vectorization combined with fuzzy matching to identify near-duplicate records in real time. By tackling entity resolution during initial ingestion, our system moves beyond theoretical proposals to deliver a practical, clean-by-design data solution.

OBJECTIVES

Our goal is to create a fast, accurate, and easily deployable deduplication system that operates in real-time at the point of data ingestion. By catching duplicates before they enter the database, we aim to reduce the need for costly and complex cleanup processes downstream. We also want to improve how the system recognizes and handles *new* product entries—especially nuanced variations like flavors or formats within an existing brand (e.g., a new "White Claw Apricot" that isn't yet in the database). While it's relatively simple to detect entirely new brands, future development will focus on subgroup recognition and adaptive classification, allowing the system to automatically create or expand product families as new variations emerge. Ultimately, we want this system to scale across a wide range of product categories and data environments.

MATERIALS & METHODS

Data Sources & Preprocessing

We combined two primary data sources:

- A labeled dataset containing ~10,000 alcoholic product entries (names, ABV, product types, and "duplicate/not-duplicate" labels).
- A 25,000-product brand-labeled corpus scraped from Uber, which was carefully cleaned and curated. We normalized product names (case-folded, punctuation removed) and organized them into a JSON file keyed by brand, each with associated brand variations and subgroups for flavors or types (see Fig. 1).

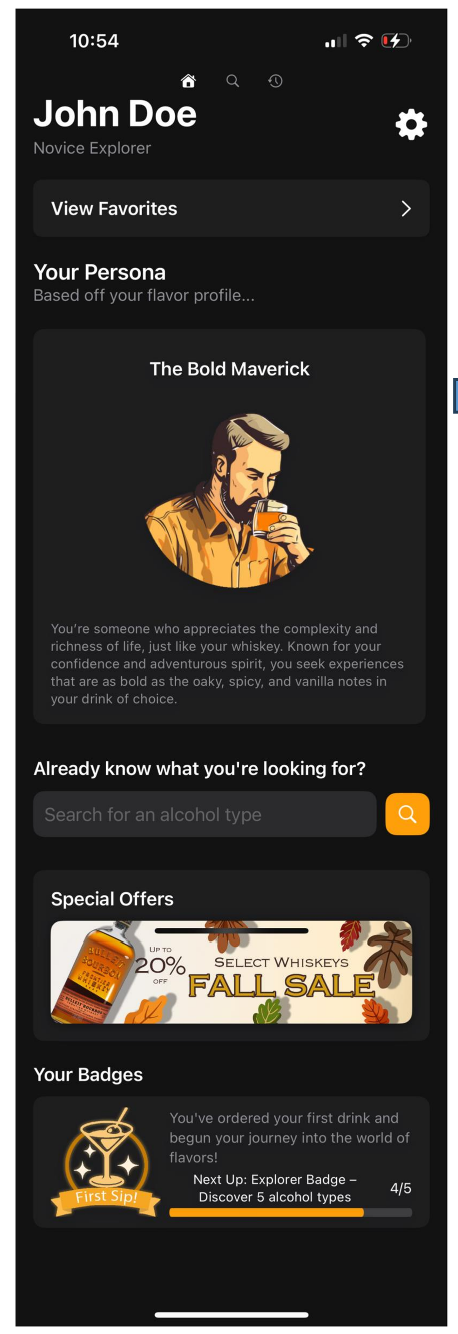
Initial Machine Learning Experiment

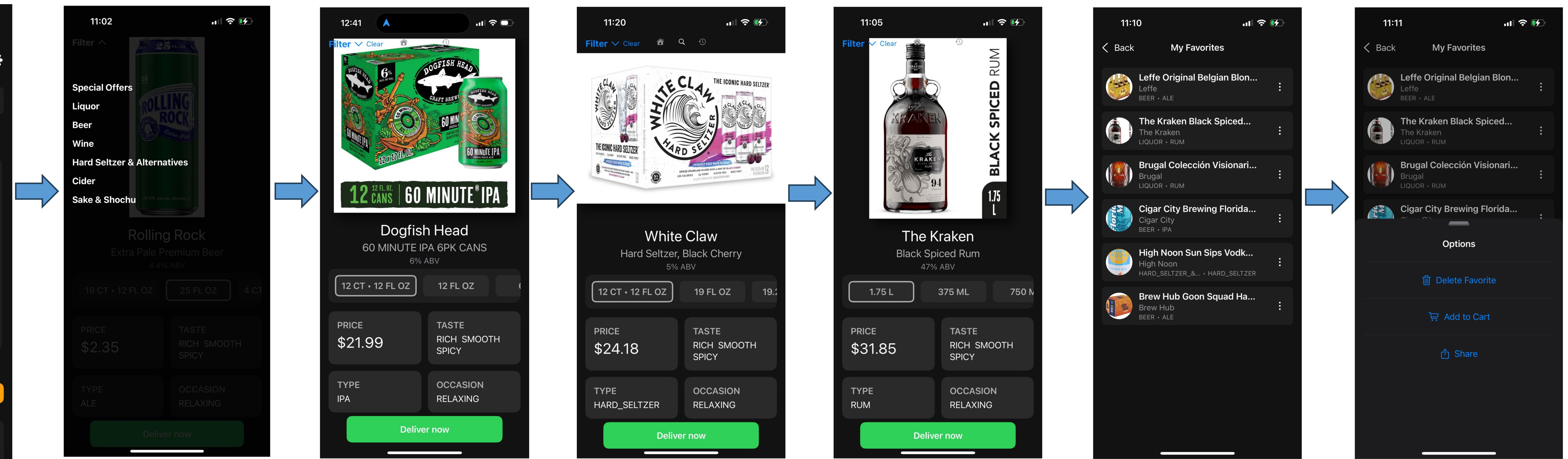
Our first approach employed an XGBoost classifier (see Fig. 2), generating features such as TF-IDF and Jaccard similarities, numerical entity overlaps, fuzzy matching scores, and product-type checks. Despite GridSearchCV tuning, this model reached only ~83% accuracy and was computationally heavy for real-time deduplication.

Brand-Based Deduplication System

Recognizing that brand identification was pivotal, we adopted a brand-blocking approach:

- Brand Matching via Aho-Corasick: Each new product name was normalized and passed through an Aho-Corasick automaton to quickly find candidate brands. Tokens from these matches received scores that scaled with how closely they matched brand variations (see Fig. 3).
- Subgroup & Flavor Scoring: Under each matched brand, subgroups (e.g., "Apple Vodka," "12-Year Whiskey") were scored via token overlap, allowing quick assignment to the correct flavor/type subgroup. New or unrecognized flavors (e.g., a variant not in the JSON) were flagged for manual or AI-based review.
- AWS Deployment: The final JSON structure was stored in Amazon S3. A Lambda function invoked the brand-matching routine as products arrived, placing confirmed duplicates in DynamoDB and routing unrecognized items ("fail") to a separate table for further verification.






```
{
  "Brand Name Here": {
    "brand_variations": [
      "Brand Name Variation"
    ],
    "subgroups": [
      {
        "display_name": "100 Nails Ranch Sonoma County (750 ml)",
        "product_type": "Wine",
        "subcategory": "White Wine",
        "variant": "",
        "products": [
          {
            "name": "100 Nails Ranch Sonoma County (750 ml)",
            "brand": "100 Nails"
          }
        ]
      }
    ]
  },
  "...": {}
}
```

Figure 1. Excerpt of Grouping JSON Structure

```python
# Feature Extraction Example
def compute_tfidf_cosine_similarity(p, q):
    vec = TfidfVectorizer()
    mat = vec.fit_transform([p, q])
    return cosine_similarity(mat[0], mat[1])[0][0]

# Training Process
xgb = XGBClassifier(eval_metric='logloss', random_state=42)
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid={
        'n_estimators': [100, 200],
        'max_depth': [3, 5],
        'learning_rate': [0.01, 0.1]
    },
    cv=5, scoring='accuracy', n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

Figure 2. Snippet of Initial XGBoost-Based Code

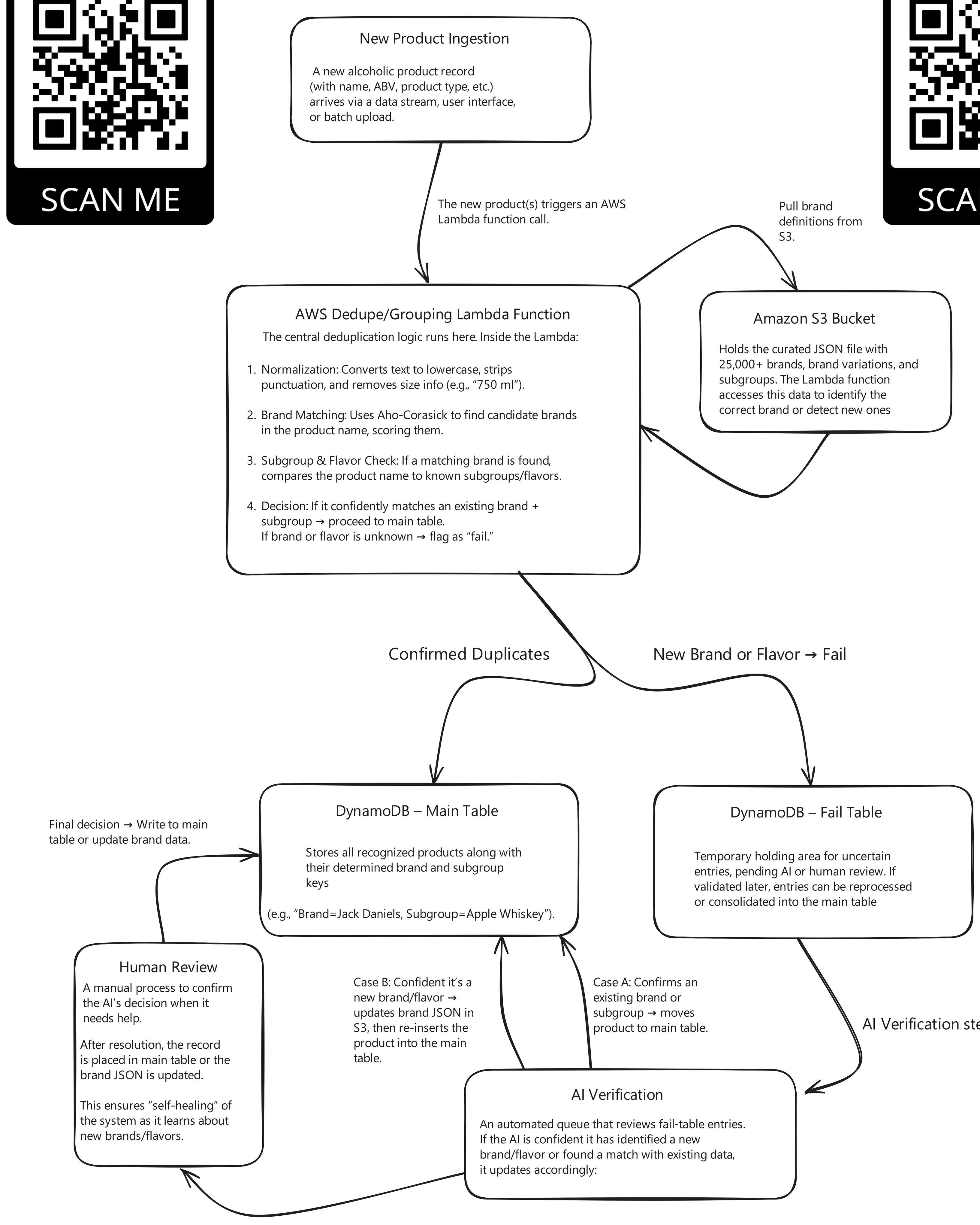
```python
import ahocorasick

def match_brand(product_name, automaton):
    """
    Uses Aho-Corasick to find all potential brand matches
    within a normalized product name and scores them.
    """
    matches = {}
    for end_idx, (brand, _) in automaton.iter(product_name):
        # Scoring logic omitted for brevity
        matches[brand] = matches.get(brand, 0) + 1
    return matches

def deduplicate_product(product_name, json_data):
    """
    1. Match brand using automaton.
    2. Find best subgroup match for flavors and product types.
    3. If no match or new flavor discovered, flag for review.
    """
    candidate_brands = match_brand(product_name, prebuilt_automaton)
    # Subgroup scoring, flavor detection, etc. (omitted)
    return best_match
```

Figure 3. Key Functions from Brand-Blocking Approach

SCAN ME



SCAN ME

RESULTS

Accuracy and Throughput

Our initial XGBoost-based model, which relied on blocking by ABV and product type, reached an **overall accuracy of ~83%** and proved inefficient for real-time product ingestion. In contrast, the **brand-blocking approach** demonstrated **~99.91% accuracy** on the curated brand dataset, a marked improvement over the machine learning model.

Large-Scale Deduplication

When tested in production with over **42,000** incoming products:

- More than **90%** were immediately assigned to correct brand subgroups.
- Approximately **6,000** entries were flagged for human or AI review, primarily due to new flavors, new brands, or ambiguous naming.

Real-Time Deployment

By storing the JSON brand hierarchy in **Amazon S3** and integrating the deduplication workflow into an **AWS Lambda** function, our system processed incoming product records in real time. Each new product was scored and inserted into the appropriate **DynamoDB** subgroup entry within seconds. Unmatched records routed to a "fail" table for further inspection.

Implications

These results confirm that **brand-focused blocking** significantly outperforms generic text-similarity classification for alcohol product deduplication. The approach facilitates **faster, more accurate** grouping, while providing an automated escalation path for unrecognized products—ensuring flexibility as the database of alcoholic beverages evolves.

CONCLUSIONS

We currently have a working algorithm that effectively identifies and merges duplicate entries within alcoholic beverage datasets. By adjusting a already existing algorithm by (insert author name) we were also able to increase its overall efficiency. However, currently this only works for alcoholic beverages, and in the future, we aim to expand its functionality to support a wider range of product types, especially those with categorical attributes such as brand, type, and packaging. This generalization will allow the system to be applied more broadly across diverse ingestion pipelines and industries.

REFERENCES

Genossar, B., Shraga, R., & Gal, A. (2023). FlexER: Flexible entity resolution for multiple intents. *Proceedings of the ACM on Management of Data*, 1(1), 1–27.

Li, H., Li, S., Hao, F., Zhang, C., Song, Y., & Chen, L. (2024). BoostER: Leveraging large language models for enhancing entity resolution. *Companion Proceedings of the ACM Web Conference 2024*, 1043–1046.

Li, Y., Li, J., Suh, J., & Xiong, C. (2022). Deep structured semantic model for entity resolution with self-attention mechanism. *Expert Systems with Applications*, 188, 116032. https://doi.org/10.1016/j.eswa.2021.116032

Martinek, A., Łukasik, S., & Gandomi, A. H. (2023). Text-based product matching - Semi-supervised clustering approach. *Proceedings of the NASK National Research Institute Research Symposium*. NASK National Research Institute; AGH University of Kraków; University of Technology Sydney.

Microsoft. (2021, July 27). *Combing through the fuzz: Using fuzzy hashing and deep learning to counter malware detection evasion techniques*. Microsoft Security Blog. https://www.microsoft.com/en-us/security/blog/2021/07/27/combing-through-the-fuzz-using-fuzzy-hashing-and-deep-learning-to-counter-malware-detection-evasion-techniques/

Wang, R., Kong, L., Tao, Y., Borthwick, A., Golac, D., Johnson, H., Hijazi, S., Deng, D., & Zhang, Y. (2024). Neural locality sensitive hashing for entity blocking. *SIAM Journal on Computing*. Society for Industrial and Applied Mathematics (SIAM).

Wu, R., Bendeck, A., Chu, X., & He, Y. (2023). Ground truth inference for weakly supervised entity matching. *Proceedings of the ACM on Management of Data*, 1(1), 1–28.