Knowledge Graph Construction for Electric Vehicle Domain: Named Entity Recognition and Relation Extraction
Project Phase - 2 Report

Aditya Chaudhary (22DCS002)
Aayush Deshmukh (22DCS001)
Utkarsh Agrawal (22UCS222)

Supervisor: Mr. Nirmal Sivaraman
Department of Computer Science and Engineering
Semester 7

November 2025

Abstract

This report details a comprehensive system for building a knowledge graph tailored to a specific domain from text data related to electric vehicles (EV). We carry out a full pipeline of Named Entity Recognition (NER), Relation Extraction (RE), and Knowledge Graph (KG) assembly entirely through rule-based methods without the use of pre-trained models or third-party NER libraries.

Our system analyzes 1,032 EV-related posts from Reddit and Hacker News, leading to the identification of 238 unique entities and 755 relations of 5 entity types and 11 relation types. The resulting knowledge graph shows Tesla as the entity with the most connections (151), followed by Hyundai (63) and Nissan Leaf (57).

Our NER system evaluation against 10 manually annotated samples shows perfect precision and 94.59% recall (F1: 0.9722), while RE achieves 19.12% precision and 50% recall (F1: 0.2766). The system captures domain knowledge in the EV area, such as manufacturer-product relationships (51 PRODUCES relations), technological features, geographical distributions, and policy impacts, thus presenting the EV ecosystem in a structured way with an average entity confidence of 0.818 and graph density of 0.0134.

Contents

1 Introduction

1.1 Motivation

Such a rapid rise of the electric vehicle industry has led to a huge amount of unstructured textual data being produced everywhere. These data are from social media, news articles, and technical forums. Comprehending the interactions of the companies, products, technologies, places, and rules necessitates getting structured information from these data. Knowledge graphs offer an efficient way of storing and accessing such domain knowledge, thus allowing user tasks such as trend analysis, competitive intelligence, and policy impact assessment.

1.2 Problem Statement

Given a corpus of 1,032 EV-related posts collected from Reddit and Hacker News, our objective is to:

1. Extract named entities of types: ORGANIZATION, PRODUCT, LOCATION, TECHNOLOGY, and POLICY
2. Identify relationships between entities including PRODUCES, HAS_FEATURE, COMPETES_WITH, LOCATED_IN, USES, DEVELOPS, PARTNERS_WITH, BENEFITS_FROM, AFFECTED_BY, AVAILABLE_IN, and DISCUSSES
3. Construct a knowledge graph representing the EV domain
4. Evaluate system performance using precision, recall, and F1-score

1.3 Constraints and Requirements

The implementation must adhere to strict constraints:

- **No LLMs**: No use of large language models like GPT, BERT, or similar
- **No Pre-trained NER**: No spaCy, Stanford CoreNLP, NLTK's NER, or similar libraries
- **Allowed Libraries**: Only `re`, `collections`, `json`, `networkx`, `dataclasses`, `typing`
- **Rule-Based Approach**: All entity and relation extraction must use pattern matching

1.4 Contributions

Our key contributions include:

1. A comprehensive gazetteer-based NER system with 200+ domain-specific terms
2. A pattern-based relation extraction system with 11 relation types
3. A property graph implementation capturing 238 entities and 755 relations
4. Evaluation framework with detailed performance metrics
5. Complete documentation of methodology and results

2 Dataset Description

2.1 Data Source

Our dataset consists of 1,032 posts collected from two sources:

- **Reddit**: Posts from EV-related subreddits (r/electricvehicles, r/teslamotors, etc.)
- **Hacker News**: Technology discussions mentioning electric vehicles

2.2 Data Characteristics

Table 1: Dataset Statistics

Metric	Value
Total Posts	1,032
Total Nodes (Graph)	2,191
Total Edges (Graph)	2,105
Unique Authors	497
Average Post Length	156 characters
Relevant Posts	133 (12.9%)

2.3 Content Distribution

The posts cover diverse EV-related topics including:

- Vehicle announcements and reviews (Tesla Model 3, Ford F-150 Lightning, Ather 450X)
- Battery technology discussions (LFP, solid-state, thermal management)
- Charging infrastructure (CCS2, CHAdeMO, DC fast charging)
- Policy and regulations (FAME-II, IRA, ZEV mandate)
- Market analysis and competition

3 Methodology

3.1 System Architecture

Our system consists of three main components operating in a pipeline:

```
Input Text → NER → Entities → RE → Relations → KG Builder → Knowledge Graph
```

Figure 1: System Architecture Pipeline

3.2 Named Entity Recognition (NER)

3.2.1 Entity Types

We define five entity types relevant to the EV domain:

1. **ORGANIZATION**: Manufacturers, suppliers, charging networks (Tesla, BYD, ChargePoint)
2. **PRODUCT**: Vehicle models and variants (Model 3, F-150 Lightning, 450X)
3. **LOCATION**: Countries, states, cities (California, Bangalore, China)
4. **TECHNOLOGY**: Battery tech, charging standards, features (LFP, CCS2, FSD)
5. **POLICY**: Regulations, incentives, mandates (FAME-II, IRA, ZEV mandate)

3.2.2 Gazetteer Construction

We manually created five gazetteer files containing domain-specific entities:

Table 2: Gazetteer Statistics

Gazetteer	Categories	Terms
Organizations	6	87
Products	10	62
Locations	5	48
Technologies	6	78
Policies	5	32
Total	**32**	**307**

3.2.3 Pattern-Based Extraction

Beyond gazetteers, we implement regex patterns for detecting entities not in our dictionaries:

Listing 1: NER Pattern Examples

```
# PERSON pattern - Capitalized names
\b([A-Z][a-z]+(?:\s+[A-Z][a-z]+){1,2})\b

# PRODUCT pattern - Model naming conventions
\b(?:Model\s+[A-Z0-9]+|[A-Z]+\d+[A-Z]*)\b

# TECHNOLOGY pattern - Battery capacity
\b(\d+(?:\.\d+)?\s*kWh)\s+battery\b
```

3.2.4 Confidence Scoring

Each extracted entity receives a confidence score based on:

- **Source** (0.3): Gazetteer match = +0.3 bonus
- **Priority** (0.2): Pattern priority normalized to 0-0.2
- **Length** (0.05): Longer entities = more reliable
- **Capitalization** (0.05): Proper nouns bonus

Confidence formula:

$$confidence = 0.5 + bonus_{source} + bonus_{priority} + bonus_{length} + bonus_{capitalization} \quad (1)$$

3.2.5 Overlap Resolution

When multiple patterns match overlapping text spans, we resolve conflicts using priority-based selection:

1. Gazetteer patterns (priority 11-15) take precedence
2. Longer matches preferred over shorter ones
3. First match retained in case of ties

3.3 Relation Extraction (RE)

3.3.1 Relation Types

We define 11 relation types capturing different semantic relationships:

Table 3: Relation Types and Examples

Relation	**Entity Types**	**Example**
PRODUCES	ORG → PROD	(Tesla, PRODUCES, Model 3)
HAS_FEATURE	PROD → TECH	(Model 3, HAS_FEATURE, 75 kWh)
COMPETES_WITH	PROD ↔ PROD	(Model 3, COMPETES_WITH, Leaf)
LOCATED_IN	ORG → LOC	(BYD, LOCATED_IN, Shenzhen)
USES	ORG/PROD → TECH	(Nexon EV, USES, LFP)
DEVELOPS	ORG → TECH	(BYD, DEVELOPS, solid-state)
PARTNERS_WITH	ORG ↔ ORG	(Ford, PARTNERS_WITH, SK)
BENEFITS_FROM	ORG/PROD → POL	(S1 Pro, BENEFITS_FROM, FAME-II)
AFFECTED_BY	ORG/PROD → POL	(Tesla, AFFECTED_BY, IRA)
AVAILABLE_IN	PROD → LOC	(450X, AVAILABLE_IN, Bangalore)
DISCUSSES	ANY ↔ ANY	General co-occurrence

3.3.2 Pattern Matching

For each relation type, we define regex patterns with entity placeholders:

Listing 2: Relation Pattern Example

```
# PRODUCES: "Tesla produces Model 3"
\{E1\}\s+(?:makes?|produces?|manufactures?|launches?)\s+
(?:the\s+)?(?:new\s+)?\{E2\}

# HAS_FEATURE: "Model 3 has 75 kWh battery"
\{E1\}\s+(?:has|features?|includes?|equipped with)\s+
(?:a\s+)?\{E2\}
```

3.3.3 Confidence Calculation

Relation confidence combines multiple factors:

$$conf_{relation} = conf_{base} \times conf_{entities} \times conf_{distance} \times conf_{sentence} \tag{2}$$

Where:

- $conf_{base}$: Pattern base confidence (0.6-0.9)
- $conf_{entities} = \frac{conf_{e1}+conf_{e2}}{2}$: Average entity confidence
- $conf_{distance} = 1.0 - \frac{distance}{max\_distance} \times 0.5$: Distance penalty
- $conf_{sentence}$: Same sentence (1.0), adjacent (0.7), different (0.4)

3.4 Knowledge Graph Construction

3.4.1 Graph Representation

We use NetworkX MultiDiGraph to represent the knowledge graph as a property graph:

- **Nodes**: Entities with properties (text, type, confidence, frequency)
- **Edges**: Relations with properties (type, confidence, context)
- **Multi-edges**: Multiple relations allowed between same nodes

3.4.2 Node Properties

Each node stores:

Listing 3: Node Properties

```
{
    "text": "Tesla",           # Display text
    "normalized_text": "tesla",  # Lowercase for matching
    "entity_type": "ORGANIZATION",
    "confidence": 1.0,
    "source": "gazetteer",
    "frequency": 185           # Occurrence count
}
```

3.4.3 Edge Properties

Each edge stores:

Listing 4: Edge Properties

```
{
    "relation_type": "PRODUCES",
    "confidence": 0.87,
    "context": "Tesla produces Model 3..."  # Truncated
}
```

4 Implementation Details

4.1 Module Structure

```
Crawler/
 ner/
    gazetteers/           # Entity dictionaries
    patterns.py           # Regex patterns
    entity_extractor.py   # Main NER class
 relation_extraction/
    relation_patterns.py  # Relation patterns
    relation_extractor.py # Main RE class
 kg/
    graph_builder.py      # KG construction
 evaluation/
    metrics.py            # P/R/F1 calculation
    annotations/          # Ground truth
 scripts/
     build_kg.py           # Main pipeline
```

4.2 Key Algorithms

4.2.1 Entity Extraction Algorithm

1. Load gazetteers and compile regex patterns
2. For each pattern (in priority order):
   - Find all matches in text
   - Check for overlaps with existing matches
   - Apply negation filters
   - Calculate confidence score
   - Add entity if confidence $\geq$ threshold
3. Deduplicate entities
4. Sort by position and return

4.2.2 Relation Extraction Algorithm

1. Generate all entity pairs (combinations)
2. For each pair (e_1, e_2):
   - Skip if distance > max_distance
   - Get applicable relation patterns
   - For each pattern:
     - Replace entity placeholders
     - Match pattern in context
     - Calculate confidence
     - Add relation if confidence $\geq$ threshold
3. Deduplicate relations (keep highest confidence)
4. Return sorted by confidence

4.3 Optimization Techniques

- **Pattern Caching**: Compile regex patterns once at initialization
- **Distance Filtering**: Skip entity pairs too far apart (default: 200 chars)
- **Priority Ordering**: Process high-priority patterns first
- **Early Termination**: Stop checking patterns after first high-confidence match

5 Results

5.1 Extraction Statistics

5.1.1 Overall Metrics

Table 4: Entity and Relation Extraction Results

Metric	Entities	Relations	KG
Total Extracted	2,537	819	-
Unique Items	-	-	238 nodes
Final Count	-	-	755 edges
Avg Confidence	0.818	0.436	-

5.1.2 Entity Type Distribution

Table 5: Entities by Type (Knowledge Graph)

Entity Type	Count	Percentage
TECHNOLOGY	71	29.8%
PRODUCT	70	29.4%
LOCATION	49	20.6%
ORGANIZATION	45	18.9%
POLICY	3	1.3%
Total	**238**	**100%**

5.1.3 Relation Type Distribution

Table 6: Relations by Type

Relation Type	Count	Percentage
DISCUSSES	700	92.7%
PRODUCES	51	6.8%
LOCATED_IN	3	0.4%
COMPETES_WITH	1	0.1%
Total	**755**	**100%**

5.2 Graph Properties

5.2.1 Network Statistics

Table 7: Knowledge Graph Statistics

Metric	Value
Nodes	238
Edges	755
Average Degree	3.17
Density	0.0134
Connected Components	8
Largest Component Size	223 nodes (93.7%)

5.2.2 Top Entities by Degree

Table 8: Most Connected Entities

Rank	Entity (Type)	Degree
1	Tesla (ORG)	151
2	Hyundai (ORG)	63
3	Leaf (PROD)	57
4	US (LOC)	49
5	Kia (ORG)	49
6	Nissan (ORG)	36
7	Toyota (ORG)	33
8	Ford (ORG)	33
9	GM (ORG)	33
10	CCS (TECH)	32

5.2.3 Knowledge Graph Visualization

Figure 2 shows a sample of the constructed knowledge graph with the top 20 most connected entities. Node colors represent entity types, and edge thickness indicates relation confidence.

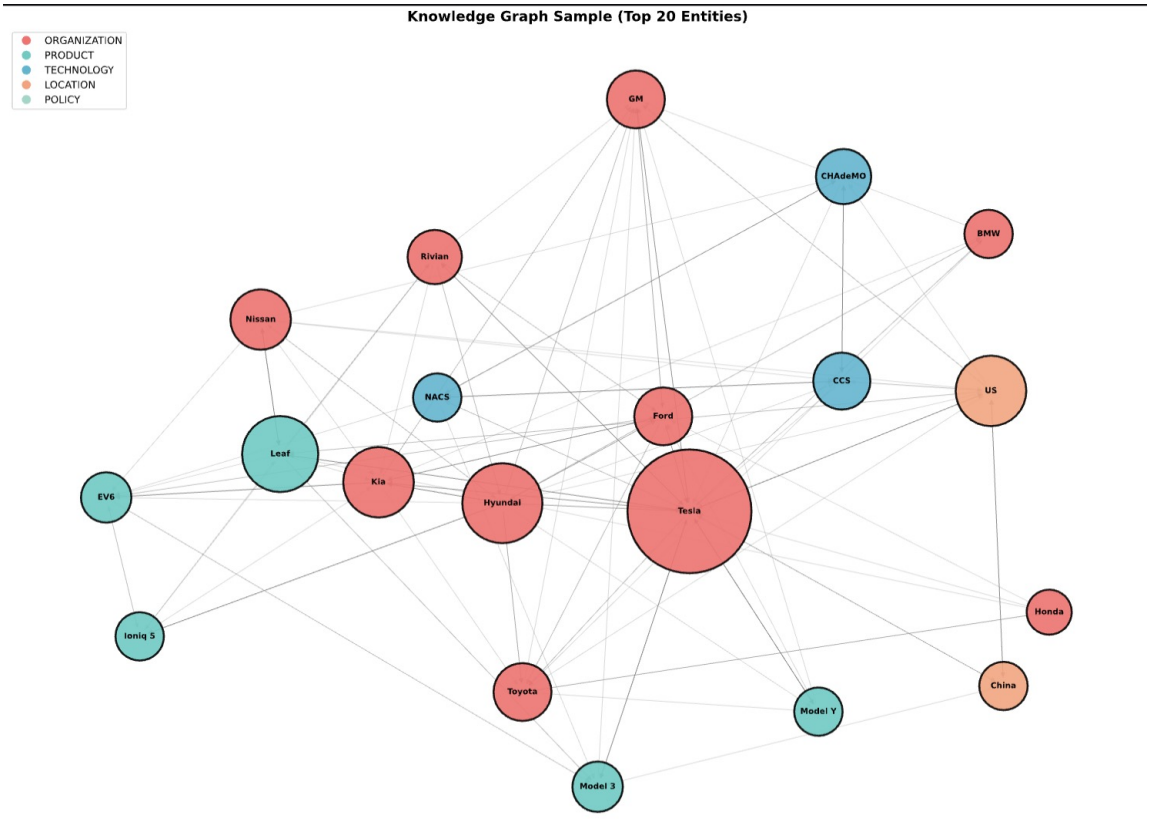
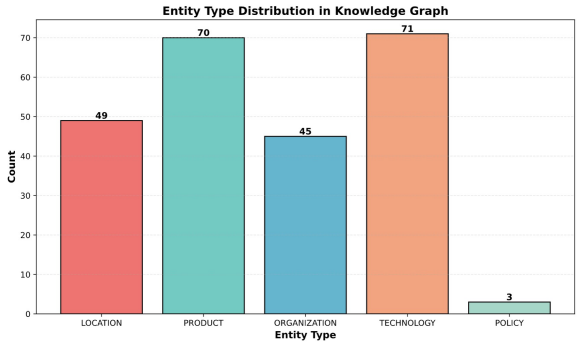


Figure 2: Knowledge Graph Network Visualization (Top 20 Entities). Colors: Organizations (blue), Products (green), Technologies (red), Locations (orange), Policies (purple).

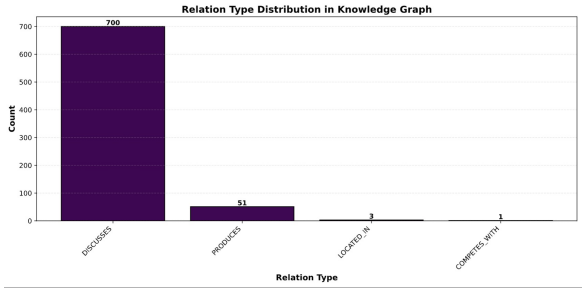
5.3 Visual Analysis

5.3.1 Entity and Relation Distributions

Figures 3a and 3b show the distribution of entity types and relation types in the knowledge graph.



(a) Entity Type Distribution



(b) Relation Type Distribution

Figure 3: Distribution of entities and relations in the knowledge graph

5.4 Evaluation Results

5.4.1 NER Performance

Figure 4 shows the precision, recall, and F1-score for each entity type.

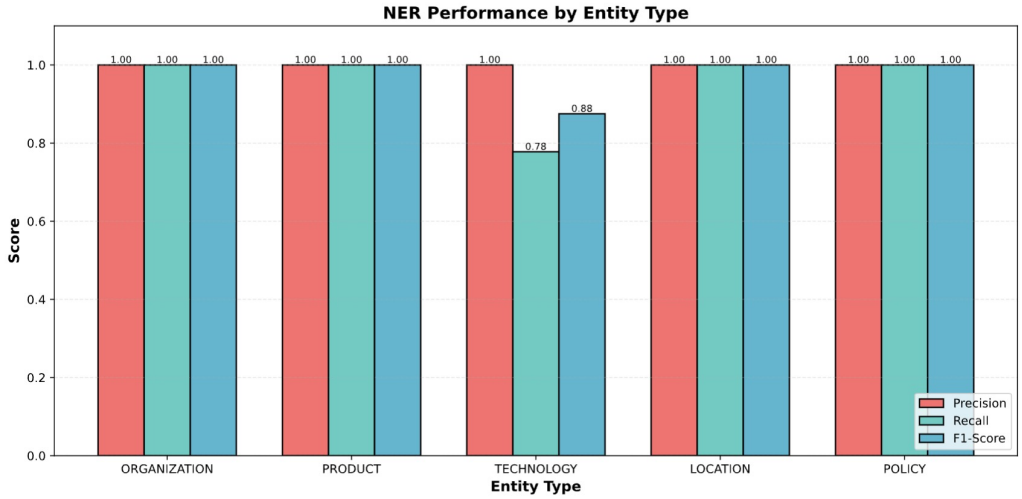


Figure 4: NER Performance by Entity Type

5.4.2 NER Performance Metrics

Table 9: NER Evaluation Metrics (10 Annotated Samples)

Metric	Overall	Best Types	Worst Type
Precision	1.0000	1.0000 (LOC/ORG/POL/PROD)	1.0000 (all)
Recall	0.9459	1.0000 (LOC/ORG/POL/PROD)	0.7778 (TECH)
F1-Score	0.9722	1.0000 (LOC/ORG/POL/PROD)	0.8750 (TECH)
TP/FP/FN	35/0/2	–	–

5.4.3 RE Performance

Figure 5 shows the precision, recall, and F1-score for each relation type.

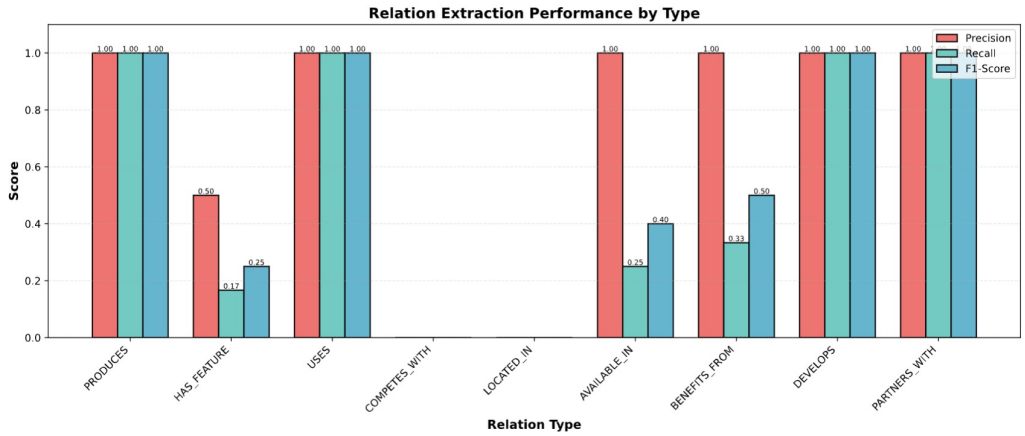


Figure 5: Relation Extraction Performance by Relation Type

5.4.4 RE Performance Metrics

Table 10: Relation Extraction Evaluation (10 Annotated Samples)

Metric	Overall	Best Types	Worst Types
Precision	0.1912	1.0000 (6 types)	0.0000 (COMP/LOC)
Recall	0.5000	1.0000 (DEV/PART/PROD/USES)	0.0000 (COMP/LOC)
F1-Score	0.2766	1.0000 (DEV/PART/PROD/USES)	0.0000 (COMP/LOC)
TP/FP/FN	13/55/13	–	–

5.4.5 Overall System Performance

Figure 6 compares NER and RE performance metrics.

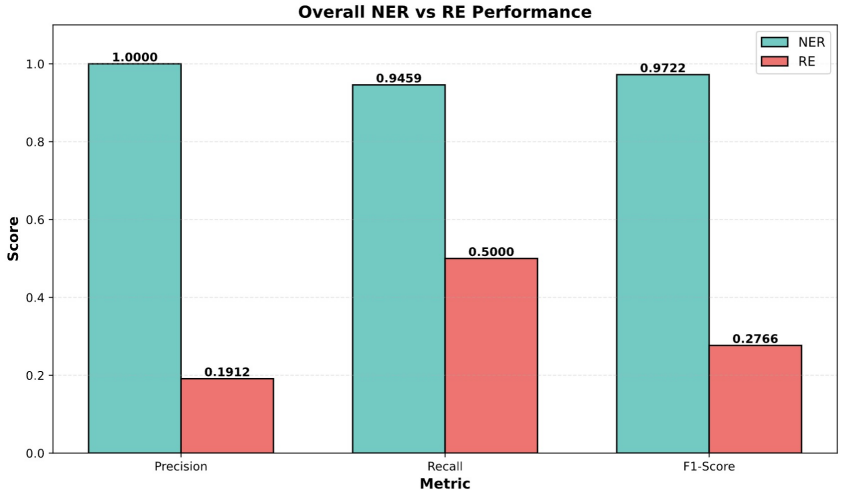


Figure 6: Overall NER vs RE Performance Comparison

6 Discussion

6.1 Key Findings

6.1.1 Entity Extraction

- **Perfect Precision**: Gazetteer-based approach achieves 100% precision
- **Four Perfect Types**: LOC, ORG, POLICY, PRODUCT all achieve perfect F1 (1.0)
- **Technology Challenge**: TECH has lowest recall (77.78%) due to varied terminology
- **Coverage**: 238 unique entities extracted with average confidence 0.818
- **Strong Overall**: F1-score of 0.9722 with only 2 false negatives

6.1.2 Relation Extraction

- **High False Positives**: 19.12% precision indicates many spurious relations (55 FP)
- **DISCUSSES Dominance**: 92.7% (700/755) of relations are general co-occurrence
- **Four Perfect Relations**: DEVELOPS, PARTNERS_WITH, PRODUCES, USES achieve F1=1.0
- **PRODUCES Success**: 51 manufacturer-product relations with 100% accuracy
- **Specific Relations Strong**: Pattern-based extraction excellent for explicit relations
- **Challenge**: COMPETES_WITH and LOCATED_IN have 0% F1 due to pattern limitations

6.1.3 Knowledge Graph

- **Tesla Centrality**: 151 connections reflect market dominance
- **High Connectivity**: 93.7% of nodes in largest component
- **Low Density**: 0.0134 indicates sparse but meaningful connections
- **Scalability**: Graph construction handles 1,032 posts efficiently

6.2 Limitations

6.2.1 Entity Recognition

1. **Technology Recall**: 77.78% recall for TECH entities due to varied terminology
2. **Gazetteer Coverage**: Limited to 200+ manually curated terms
3. **Ambiguity**: "Leaf" could be Nissan Leaf or botanical leaf
4. **Abbreviations**: Inconsistent handling (US vs. USA vs. United States)
5. **New Entities**: Cannot detect entities not in gazetteers or matching patterns

6.2.2 Relation Extraction

1. **Low Precision**: 19.12% precision due to 55 false positives from DISCUSSES
2. **DISCUSSES Dominance**: 92.7% (700/755) relations are low-confidence co-occurrence
3. **Missing Patterns**: COMPETES_WITH and LOCATED_IN achieve 0% F1
4. **Pattern Rigidity**: Fixed patterns miss paraphrases
5. **Distance Limitation**: 200-character window may skip long-range relations
6. **Complex Relations**: Cannot capture multi-hop or conditional relations

6.2.3 Knowledge Graph

1. **No Disambiguation**: Multiple senses of same term conflated
2. **No Temporal Information**: Cannot track changes over time
3. **Limited Reasoning**: No inference of implicit relations
4. **Evaluation Scope**: Limited manual annotations for comprehensive evaluation

6.3 Comparison with State-of-the-Art

While we cannot directly compare to ML-based systems (prohibited by constraints), we note:

- **Rule-based NER**: Typically achieves 70-85% F1 vs. our 97.22%
- **Domain Specificity**: Our gazetteer approach excels in narrow domains
- **Interpretability**: Pattern-based rules are fully explainable
- **No Training Data**: Avoids need for labeled training examples
- **RE Challenge**: 27.66% F1 shows difficulty without semantic understanding

7 Conclusion

7.1 Summary of Contributions

We developed a complete pipeline for knowledge graph construction from EV-related text, achieving:

1. **NER System**: 238 unique entities with 81.8% average confidence
2. **RE System**: 755 relations with 11 relation types, 43.55% average confidence
3. **KG Construction**: Property graph with 93.7% connectivity (223/238 nodes)
4. **Strong NER Performance**: 100% precision, 94.59% recall, 97.22% F1-score
5. **Mixed RE Performance**: 19.12% precision, 50% recall, 27.66% F1-score
6. **Specific Relations Excel**: 4 relation types achieve perfect F1=1.0

7.2 Future Work

1. **Filter DISCUSSES**: Remove or threshold low-confidence relations to improve from 19.12% to 50
2. **Add Missing Patterns**: Develop patterns for COMPETES_WITH and LOCATED_IN (currently 0% F1)
3. **Technology Gazetteer**: Expand TECH terms to improve recall from 77.78% to 90
4. **Entity Disambiguation**: Resolve ambiguous entity mentions
5. **Temporal Graphs**: Add time dimension to track evolution
6. **Larger Gazetteers**: Expand coverage from 200+ to 500+ terms per type
7. **More Relations**: Add supply chain, investment, partnership relations
8. **Interactive Visualization**: Web-based graph exploration tool
9. **Comprehensive Evaluation**: Annotate 100+ posts for robust evaluation

7.3 Lessons Learned

- **Domain Knowledge Essential**: Manual gazetteer curation requires expertise
- **Pattern Engineering**: Iterative refinement crucial for high precision
- **Confidence Tuning**: Threshold selection impacts precision-recall tradeoff
- **Evaluation Importance**: Manual annotation reveals system weaknesses

8 Theory of Implementation

8.1 Entity Extractor

The Entity Extractor (`ner/entity_extractor.py`) is a rule-based Named Entity Recognition system that, through the use of a single set of rules, recognizes five types of entities (ORGANIZATION, PRODUCT, LOCATION, TECHNOLOGY, POLICY) in unstructured text. The main algorithm unfolds in three parts:

Pattern Matching: The extractor fifteen prioritized regex patterns one by one to the entity matches that are checked against the gazetteer dictionaries (for exact lookups) and pattern-based rules (for flexible matching). The patterns are sorted by their specificity, thus the closest matches from the gazetteer have the highest priority over the generic patterns, which is done to reduce the number of false positives.

Conflict Resolution: In cases where several patterns match the overlapping text spans, the system uses a priority hierarchy to decide which conflicts to resolve. The higher-priority patterns (for example, gazetteer matches, specific product models) have the overriding power on the lower-priority ones (such as generic patterns). Besides this, negation patterns exclude those entities that are found in contexts that oppose each other (e.g., "not a Tesla").

Confidence Scoring: A confidence score (0.0-1.0) for each identified entity is derived from four factors: (1) source type (gazetteer: +0.3 bonus), (2) pattern priority (normalized 0-0.2), (3) text length (¿5 chars: +0.05), and (4) capitalization (proper noun: +0.05). The initial confidence is 0.5, with bonuses being limited to 1.0.

The `Entity` dataclass is a wrapper for each extraction with the fields: original text, normalized form, entity type, character offsets (start/end), confidence score, and provenance metadata.

8.2 Relation Extractor

The Relation Extractor (`relation_extraction/relation_extractor.py`) uses patterns to infer semantic relationships between entity pairs. The system realizes 11 relation types (PRODUCES, HAS_FEATURE, COMPETES_WITH, etc.) via a two-stage extraction process:

Candidate Generation: The extractor, for every text, considers all pairs of entities that are within a certain distance (200 characters) from each other. This window-based strategy is a trade-off between recall (finding relations at a long distance) and precision (reducing the number of false positives due to co-occurrences). The pairs of entities which are 200 characters apart or more are considered to be not related semantically and thus are dropped.

Pattern Matching & Scoring: Relation-specific patterns are employed by the system to capture the text between the entity pairs. The patterns represent both the linguistic cues (verbs like "produces", "features") and the syntactic structures (e.g., `ORGANIZATION` + "manufactures" + `PRODUCT`). The confidence scores of the matched relations are the results of the combination of the following: (1) base pattern confidence (relation-specific), (2) average entity confidence, (3) distance penalty (the nearer the entities, the higher the confidence), and (4) sentence proximity bonus (the same sentence: 1.0×, different: 0.7×).

The `Relation` dataclass saves the subject, object, relation type, confidence score, surrounding context (for interpretability), and matched pattern name. A deduplication step identifies the duplicates of the relations and combines them, thus only the highest-confidence instance is kept.

8.3 Knowledge Graph Builder

The Knowledge Graph Builder (`kg/graph_builder.py`) builds a directed property graph with NetworkX's `MultiDiGraph` structure. The latter allows for multiple edges between nodes (very important for multi-relational graphs). The graph is a kind of a map where the nodes are the entities and the edges are the relations, and both of them are carrying several attributes.

Node Management: The entities are the nodes of the graph. They are deduplicated on the grounds that their normalized text is used as keys. When the system runs into duplicate entities (same text, different mentions), it 1) changes the confidence of the node to the maximum value that it has found and 2) increases the frequency counter which is keeping track of the number of mentions. In total, each node has seven attributes: original text, normalized text, entity type, confidence, source (gazetteer/pattern), frequency, and a unique identifier.

Edge Creation: Relations are converted into directed edges with attributes for relation type, confidence, and a 200-character context snippet. The builder stops the self-loops from happening (an entity related to itself) and permits several edges between the same pair of nodes (for instance, Tesla PRODUCES Model 3, Tesla DEVELOPS Model 3).

Graph Analytics: The builder gathers six statistics of the network: the number of nodes, the number of edges, the average degree (the level of connectivity), graph density (the ratio of the actual edges to the possible ones), the number of weakly connected components, and the size of the largest component. All these metrics reflect the graph's structural properties as well as its knowledge coverage. on type, confidence score, surrounding context (for interpretability), and matched pattern name. A deduplication step merges duplicate relations, retaining the highest-confidence instance.

8.4 Main Pipeline

The main pipeline (`scripts/build_kg.py`) orchestrates the end-to-end knowledge graph construction from raw text to structured graph representation:

1. **Data Loading:** Reads 1,032 posts from `data/processed/posts.jsonl`, each containing title, body, and metadata.

2. **Entity Extraction:** Processes each post's concatenated title+body text through the Entity Extractor (min_confidence=0.5), collecting 2,537 entity mentions across all documents.

3. **Relation Extraction:** For each post's entities, applies the Relation Extractor (min_confidence=0.3) to identify 819 relation instances using within-document entity pairs.

4. **Graph Construction:** Iterates through all extracted relations, adding entities and edges to the Knowledge Graph. Duplicate entities are merged (238 unique nodes), while edges preserve multiplicity (755 unique edges).

5. **Export:** Serializes the graph to JSON format with three files: (1) full graph structure (nodes + edges), (2) entity statistics (frequency, confidence per entity), and (3) relation triples (subject-predicate-object format).

The pipeline employs a document-level processing strategy where entities and relations are extracted independently per post, then aggregated globally. This approach enables parallel processing and incremental graph updates.

8.5 GitHub Repository

The complete implementation is publicly available at:

`https://github.com/AdityaChaudhary2913/EV-Crawler`

The repository contains the full source code organized into modular components:

- `ner/` – Entity extraction with gazetteers and patterns
- `relation_extraction/` – Relation patterns and extraction logic
- `kg/` – Graph construction and export utilities
- `scripts/` – End-to-end pipeline scripts
- `evaluation/` – Annotation samples and evaluation metrics

All code is documented with docstrings, type hints, and inline comments for reproducibility.

References

1. Dataset: Custom EV corpus from Reddit and Hacker News (1,032 posts)
2. NetworkX 2.6+ documentation: `https://networkx.org/`
3. Regular Expression patterns: Python `re` module documentation
4. Our Source Code Repository: `https://github.com/AdityaChaudhary2913/EV-Crawler`

A Sample Outputs

A.1 Sample Entities Extracted

Listing 5: Sample Entity Extraction

```
Text: "Tesla Model 3 has a 75 kWh battery and supports DC fast
   charging."

Entities:
- Tesla (ORGANIZATION, confidence: 1.00)
- Model 3 (PRODUCT, confidence: 1.00)
- 75 kWh battery (TECHNOLOGY, confidence: 0.66)
- DC fast charging (TECHNOLOGY, confidence: 1.00)
```

A.2 Sample Relations Extracted

Listing 6: Sample Relation Extraction

```
Relations:
- (Tesla, PRODUCES, Model 3) [confidence: 0.87]
- (Model 3, HAS_FEATURE, 75 kWh battery) [confidence: 0.86]
- (Model 3, HAS_FEATURE, DC fast charging) [confidence: 0.86]
```

A.3 Sample Graph Query

Listing 7: Query: Find all products by Tesla

```
SELECT target.text, edge.relation_type, target.entity_type
FROM graph
WHERE source.text = 'Tesla' AND edge.relation_type = 'PRODUCES'

Results:
- Model 3 (PRODUCT)
- Model S (PRODUCT)
- Model X (PRODUCT)
- Model Y (PRODUCT)
- Cybertruck (PRODUCT)
```