Technical Report
Smartphone Knowledge Graph Project

Benoît BARBIER   Massi-Nissa ABBOUD   Ilyasse SOUSSI

1 Use Case

Smartphone recommendation system based on user use cases (Gaming, Photography, Vlogging, Business, EverydayUse, MinimalistUse) and price segments (Budget, MidRange, Flagship).

2 Task 1: OWL and SKOS Modeling

2.1 OWL Ontology

2.1.1 Main Classes (8)

Class	Description
`sp:Brand`	Manufacturer (Apple, Samsung...)
`sp:BasePhone`	Phone model (iPhone 16, Galaxy S24)
`sp:PhoneConfiguration`	Storage/RAM variant (iPhone 16 128GB/8GB)
`sp:PriceOffering`	Price offer at a store
`sp:Store`	Retail point
`sp:User`	Synthetic user with preferences
`sp:TagSentiment`	Aggregated sentiment per tag (Camera, Battery, Value)

2.1.2 Defined Subclasses (OWL Restrictions)

- `sp:LargeBatteryPhone` $\equiv$ BasePhone $\cap$ (batteryCapacityMah $\geq$ 5000)
- `sp:FiveGPhone` $\equiv$ BasePhone $\cap$ (supports5G = true)
- `sp:NFCPhone` $\equiv$ BasePhone $\cap$ (supportsNFC = true)
- `sp:HighResolutionCameraPhone` $\equiv$ BasePhone $\cap$ (mainCameraMP $\geq$ 100)

2.1.3 Property Characteristics

- **owl:FunctionalProperty**: phoneName, brandName, storageGB, ramGB, priceValue, userId
- **owl:inverseOf**: hasBrand $\leftrightarrow$ manufactures, hasBasePhone $\leftrightarrow$ hasConfiguration, forConfiguration $\leftrightarrow$ hasPriceOffering

2.2 SKOS Thesaurus

ConceptScheme: `spv:SmartphoneVocabulary`

2.2.1 UseCase Hierarchy

```
spv:UseCase
  +-- spv:Gaming (refresh >= 120Hz, RAM >= 12GB, battery >= 4500mAh, 5G)
  +-- spv:Photography (camera >= 108MP, AMOLED/OLED)
  +-- spv:Vlogging (selfie >= 32MP, main >= 50MP)
  +-- spv:Business (NFC, 5G, battery >= 5000mAh, storage >= 256GB)
  +-- spv:EverydayUse (NFC, 5G, battery >= 4000mAh)
  +-- spv:MinimalistUse (no 5G, RAM <= 8GB)
```

2.2.2 PriceSegment Hierarchy

```
spv:PriceSegment
  +-- spv:Flagship (> 900 EUR)
  +-- spv:MidRange (400-900 EUR)
  +-- spv:Budget (< 400 EUR)
```

`skos:related` links between use cases and segments (e.g., Gaming ↔ Flagship).

3 Task 2: SHACL Constraints

Shape	Constraints
BasePhoneShape	phoneName: 1 non-empty string; hasBrand: 1 Brand; battery: 500–10000 mAh; camera: 1–300 MP; year: 2000–2030
BrandShape	brandName: 1 non-empty string
UserShape	userId: 1 non-empty string; interestedIn: ≥1 skos:Concept
PhoneConfigurationShape	hasBasePhone: 1 BasePhone; storage: 1–2000 GB; RAM: 1–128 GB
StoreShape	storeName: 1 non-empty string
PriceOfferingShape	priceValue: 1 decimal >0; forConfiguration: 1 PhoneConfiguration; offeredBy: 1 Store

4 Task 3b: RML Mapping

4.1 Heterogeneous Data Sources

Source	Format	Content
Phone specifications	JSON	Technical specs (148 models, 893 configs)
Store prices	JSON	Prices per store
Review sentiments	JSON	Extracted sentiments (tags: Performance, Value, Display...)
User preferences	CSV	500 users + use case
User-phone interactions	CSV	986 user-likes-phone relations

4.2 TriplesMaps Defined (9)

1. **BasePhoneMapping**: phone_id → BasePhone with specs (year, display, chipset, camera, battery, 5G, NFC)

2. **PhoneConfigurationMapping**: config_id → PhoneConfiguration (storage, RAM) + BasePhone link
3. **BrandMapping**: brand → Brand
4. **UserMapping**: user_id → User
5. **UserInterestMapping**: user → interestedIn → UseCase
6. **UserPhoneLikesMapping**: user → likes → PhoneConfiguration
7. **StoreMapping**: store_id → Store
8. **PriceOfferingMapping**: price_id → PriceOffering (price, config, store)
9. **TagSentimentMapping** + **BasePhoneSentimentMapping**: sentiments per tag

Execution: Python script using RMLMapper Java to generate RDF triples.

5 Task 3a: Unstructured Data Ingestion

5.1 Overview

To enrich the Knowledge Graph with up-to-date market data, we implemented a Neuro-Symbolic Extraction Pipeline. This module processes unstructured technical articles and reviews to extract smartphone specifications, price data, and sentiment. Unlike traditional regex-based scrapers, this approach utilizes LLMs constrained by SHACL shapes and Pydantic schemas to ensure data quality and ontological consistency.

5.2 Schema-Guided Extraction

The extraction process solves the "hallucination" problem common in LLMs by strictly enforcing a target schema derived from our ontology constraints.

- **Dynamic Schema Generation:** Pydantic models define the expected output structure. This schema is injected into the LLM's system prompt, forcing the model to output a strict JSON list of objects.
- **Controlled Vocabularies (SKOS):** The extraction logic is aware of our SKOS vocabulary. The system prompts the LLM to map entities (e.g., "Snapdragon 8 Gen 3") to standard terms rather than generating free text.
- **Null Handling:** Fields not explicitly mentioned in the source text are set to `null` rather than being guessed, preserving data integrity.

5.3 Two-Pass Review Extraction

For sentiment extraction from reviews:

1. **First pass (Filtering):** Each review is analyzed to determine whether it contains exploitable information. Reviews with no substantive content are discarded.
2. **Second pass (Extraction):** For retained reviews, the LLM extracts positive and negative sentiment tags (Camera, Battery, Performance, Value, Display, Build, Overall).
3. **Normalization:** Extracted tags pass through a normalization dictionary to map synonyms to canonical names (e.g., "battery life" → Battery; "screen quality" → Display).

5.4 Knowledge Graph Integration (Merge Logic)

A critical challenge in KG construction is duplicate handling. Our pipeline implements a "Gap-Filling" strategy:

- **Identity Resolution:** Unique URIs are generated (e.g., `sp:phone_brand_model_storage`) to identify instances.
- **Existence Check:** Before insertion, the system queries the existing KG to check if the entity already exists.
- **Selective Merging:**
  - *New Entities:* Full creation of the node and all properties.
  - *Existing Entities:* Only insert properties currently missing in the graph (e.g., adding a missing price to an existing phone node).

This ensures the KG acts as the "source of truth" while allowing for enrichment.

5.5 Output Example

iPhone 16 → Performance: +3/-1, Value: +2/-3, Overall: +7/-2

6 Task 3c: Alignment and Linking

6.1 T-Box Alignment

Method: LOV API (Linked Open Vocabularies) + LLM (GPT-4o-mini) for semantic selection

Local Term	**Relation**	**External Term**
sp:BasePhone	rdfs:subClassOf	schema:Product
sp:BasePhone	skos:exactMatch	wd:Q22645 (smartphone)
sp:Brand	owl:equivalentClass	schema:Brand, gr:Brand
sp:Store	owl:equivalentClass	schema:Store
sp:User	rdfs:subClassOf	schema:Person, foaf:Person
sp:PriceOffering	owl:equivalentClass	gr:Offering
sp:hasBrand	owl:equivalentClass	schema:brand, gr:hasBrand
sp:priceValue	owl:equivalentClass	schema:price
sp:manufactures	skos:closeMatch	schema:manufacturer

6.2 A-Box Linking

Method:

- **Brands → DBpedia:** DBpedia Spotlight API (confidence=0.8)
- **Phones → Wikidata:** Wikidata Search API with smartphone keyword filtering

Results:

- 11 brands linked (Apple → dbr:Apple_Inc., Samsung → dbr:Samsung, Google → dbr:Google...)
- 65+ phones linked (iPhone 16 → wd:Q130267745, Galaxy S24 → wd:Q124316718...)

7 Knowledge Graph Construction Pipeline

1. **Load:** Base facts + OWL ontology + SHACL shapes + SKOS vocabulary
2. **A-Box Linking:** owl:sameAs to DBpedia/Wikidata
3. **T-Box Alignment:** equivalences/subClassOf to Schema.org, GoodRelations, FOAF
4. **SPARQL CONSTRUCT:** materialize price segments + phone classifications
5. **OWL RL Reasoning:** inference via owlrl library
6. **Export:** Final knowledge graph (∼86,000 triples)

7.1 SPARQL CONSTRUCT Rules

- Price classification: Budget (<400€), MidRange (400–900€), Flagship (>900€)
- Phone classification: LargeBatteryPhone (≥5000mAh), HighResolutionCameraPhone (≥100MP)

8 Knowledge Graph Statistics

Metric	Value
Base phone models	148
Phone configurations	893
Users	500
Likes relations	986
Use cases	6
Total triples	∼86,000

9 Task 4: Knowledge Graph Exploitation - GraphRAG Approaches

We implemented two GraphRAG approaches for exploiting the Knowledge Graph.

9.1 Approach 1: NL to SPARQL

9.1.1 Methodology

This approach relies on a local LLM (Ollama with llama3.2) which transforms a natural language question into an executable SPARQL query. The core lies in prompt engineering: we provide the LLM with an exhaustive context describing our ontology.

Prompt Construction The system prompt includes:

- **Namespace declarations:** `sp:` for the main ontology, `spv:` for SKOS vocabulary
- **Class descriptions:** BasePhone, PhoneConfiguration, Brand, User, Variant
- **Property list with domains/ranges:** phoneName, batteryCapacity, mainCameraMP, refreshRate, has5G, displayType (data properties); manufacturedBy, hasBasePhone, variantOf, interestedIn (object properties)
- **Navigation documentation:** PhoneConfiguration → hasBasePhone → BasePhone

- **SKOS concepts:** Gaming, ProGaming, Photography, ProPhotography, Vlogging, Business, EverydayUse, MinimalistUse
- **SPARQL syntax rules:** "ORDER BY must be OUTSIDE the WHERE block", "Use integers for numeric filters", "Always include PREFIX declarations"
- **Valid query examples** to guide the LLM

Execution Pipeline

1. Reception of user question in natural language
2. Construction of complete prompt (ontological context + question)
3. Sending to Ollama LLM (llama3.2) for SPARQL generation
4. Extraction of SPARQL query from LLM response
5. Automatic injection of prefixes if missing
6. Query execution via RDFLib on local graph
7. Formatting and returning results in tabular form

9.1.2 Constraints Encountered

- **Properties invented by the LLM:** The LLM invented properties like `sp:price` instead of `sp:priceEUR`. Solved by including exhaustive property list with examples.
- **SPARQL syntax errors:** ORDER BY placed inside WHERE block, incorrect types in filters. Added explicit rules and counter-examples.
- **Missing prefixes:** Implemented automatic prefix injection before execution.
- **PhoneConfiguration/BasePhone confusion:** Documented the hierarchy with correct join examples.
- **SKOS vocabulary not recognized:** Added dedicated section listing all concepts with namespace.

9.1.3 Advantages

- Exact and reproducible results
- Full traceability (user can view/verify generated SPARQL)
- Native aggregation support (COUNT, AVG, SUM, GROUP BY, ORDER BY, LIMIT)
- Complex query capability (joins, OPTIONAL, UNION, subqueries)

9.1.4 Disadvantages

- Total failure on syntax error
- Critical dependency on prompt completeness
- Cannot process semantic questions ("premium phone", "futuristic")
- Heavy maintenance on ontology evolution

9.2 Approach 2: GraphRAG via Embeddings

9.2.1 Model Choice: From TransE to RotatE

First Attempt with TransE We initially implemented TransE to model relations as vector translations. Limitations observed:

- Poor modeling of symmetric relations
- Inability to represent composition relations

Migration to RotatE We migrated to RotatE, which models relations as rotations in complex space. Each entity is a 128-dimensional complex vector, each relation is a unit rotation.

Why RotatE fits our use case:

- **Symmetric relations:** 180° rotation models "A similar to B" and "B similar to A"
- **Antisymmetric relations:** non-180° rotations capture "A better than B" without inverse
- **Inverse relations:** if r_1 is "manufactured by", then $r_2 = r_1^{-1}$ is naturally "manufactures"
- **Composition:** $r_1 \circ r_2 = r_3$ representable by rotation multiplication

9.2.2 Methodology

Training Data Preparation Triples extracted from the KG:

- Phone → characteristics: (samsung_galaxy_s24, hasBattery, battery_4000to5000)
- Phone → use case: (rog_phone_9_pro, suitableFor, usecase/Gaming)
- User → preferences: (user_042, interestedIn, usecase/Photography)

Execution Pipeline

1. **Intent extraction via Ollama:** LLM extracts structured intent (features, brand, budget, use case)
2. **Intent → embeddings mapping:** via synonym dictionary (gaming → ProGaming)
3. **Query embedding computation:** averaging embeddings of identified use cases
4. **Cosine similarity search:** compute similarity for each phone embedding
5. **Filtering and ranking:** apply filters (brand, budget), sort by similarity
6. **Response generation:** send phones as context to Ollama for natural language response

9.2.3 Constraints Encountered

- **Identifier format incompatibility:** PyKEEN IDs (samsung_galaxy_s24) differed from JSON (Samsung Galaxy S24). Implemented normalization and bidirectional mapping.
- **Numeric values not usable:** Created discretized categories (battery_4000to5000, battery_5000to6000) for RotatE.
- **User vocabulary vs SKOS gap:** Built manual mapping dictionary for synonyms.

9.2.4 Advantages

- Handles semantic questions ("premium", "futuristic", "suitable for my grandmother")
- Similarity search ("like an iPhone but cheaper")
- Error tolerance (always returns top-k, never complete failure)
- Multiple/fuzzy criteria handling via weighted embedding average
- Exploits learned relational patterns

9.2.5 Disadvantages

- Approximate results (no exact match guarantee)
- No aggregation support
- Quality dependent on training triples

9.3 Comparison of the Two Approaches

9.3.1 Analysis by Question Type

Factual/statistical questions: NL→SPARQL excels. "How many Samsung phones support 5G?" generates exact COUNT. Embeddings cannot answer aggregative questions.

Semantic/subjective questions: Embeddings have the advantage. "A premium and futuristic phone" has no SPARQL equivalent, but embeddings find phones close to flagships in vector space.

9.3.2 Summary

Criterion	NL→SPARQL	Embeddings
Precision	Exact results	Approximate similarity
Explainability	Displays generated query	Scores difficult to interpret
Robustness	Fails on invalid syntax	Always returns top-k
Capabilities	Aggregations	Subjective questions, similarity

9.3.3 Complementarity

These approaches are complementary:

- **NL→SPARQL:** factual queries, statistics, filtered lists, traceability
- **RotatE embeddings:** personalized recommendations, vague questions, exploration, similarity search

10 Task 4b: Recommendation System via Link Prediction

10.1 Problem Definition

The recommendation module addresses two distinct scenarios:

- **Personalized Recommendation (Transductive):** Predicting which phones an existing user in the database is likely to interact with (`likes` relation).

- **Cold-Start Recommendation (Inductive):** Recommending phones to a new user based on natural language descriptions of their interests and budget, using the KG as a reasoning engine.

10.2 Data Preparation and Graph Enrichment

To enable effective learning, we transformed raw RDF triples into a learnable format:

- **Discretization:** Continuous variables were bucketed into categorical entities to serve as shared nodes. For example, prices of €899 and €950 both map to entity `vocab/price_flagship`. This allows the model to learn shared representations for high-end devices.
- **Latent Link Inference (`suitableFor`):** We enriched the graph by inferring implicit relationships. If user U likes phone P, and U is interested in concept C (e.g., Gaming), we generate: $(P, \texttt{suitableFor}, C)$. This explicitly teaches the model which phones satisfy specific use cases.

10.3 Link Prediction Model: RotatE

We utilized PyKEEN to train a Knowledge Graph Embedding (KGE) model.

10.3.1 Model Selection Rationale

We selected RotatE over simpler models like TransE:

- **TransE limitation:** Models relationships as translations ($h + r \approx t$), which struggles with 1-to-N relations (forcing target entities to collapse into the same vector space).
- **RotatE advantage:** Models relationships as rotations in complex vector space, capturing symmetric and inverse patterns more effectively for user-item interactions.

10.3.2 Training Strategy

We observed that inclusion of inverse triples (e.g., predicting "Who likes this phone?" in addition to "Which phone does this user like?") significantly impacted evaluation metrics.

Configuration: Final model trained without inverse triples in evaluation set.

Rationale: Adding inverse tasks dramatically increases the search space cardinality (choosing 1 user out of thousands vs. 1 phone out of dozens). Focusing purely on the forward prediction task achieved cleaner convergence.

10.3.3 Performance Metrics

Final model results on test set:

- **MRR (Mean Reciprocal Rank):** 0.3689
- **Hits@10:** 0.5137

The model correctly places the true positive phone in the top 10 recommendations more than 50% of the time.

10.4 Cold-Start Solution: GraphRAG Pipeline

Standard KGE models cannot generate embeddings for users that did not exist during training. We implemented a three-stage hybrid system:

10.4.1 Stage 1: Semantic Mapping (LLM Layer)

The user provides a natural language query. An LLM maps this intent to the Controlled Vocabulary:

- Input: "I need a cheap phone for gaming"
- Output: `['vocab/price_budget', 'vocab/Gaming']`

10.4.2 Stage 2: Sum-of-Scores Reasoning (KG Layer)

Instead of manufacturing a synthetic user vector, we implemented a "Sum-of-Scores" algorithm:

$$\text{Score}_{\text{total}}(P) = \sum_{C \in \text{Concepts}} \text{Score}(P, \texttt{suitableFor}, C)$$

This finds the geometric centroid in embedding space representing the best compromise between all requested features.

10.4.3 Stage 3: Explanation (Generative Layer)

Top-ranked phones from the KG are passed to the LLM with their metadata. The LLM generates a natural language explanation rooted in graph data (e.g., "I recommend the Redmi Note because our data links it strongly to High Refresh Rates and Budget Pricing").