Project: Implementing a Query System for CMR Textures

1. Introduction

1.1 Project Overview

In this project, students will develop a system that translates natural language queries into a structured logical query language designed for Converged Mixed Reality (CMR) texture data. The system should be able to parse structured query strings, interpret the encoded data within textures, and return relevant results.

CMR textures encode metadata into texels, embedding spatial, operational, and analytical data directly within textures. This allows for advanced querying and retrieval of information about objects in a mixed-reality environment. However, extracting relevant data requires a structured query system that can interpret these encoded values.

1.2 Problem Statement

Currently, querying CMR textures requires manually constructing structured query objects. This is inefficient for real-world applications where users prefer asking natural questions, such as:

💬 *"Show me all of the areas that have galvanic corrosion on critical areas that have not been resolved in the last 30 days."*

To enable a more flexible and user-friendly approach, we need an LLM-powered or rule-based NLP system that:

1. Understands natural language input
2. Maps words and phrases to known schema fields and values
3. Generates a structured query string in a predefined format

---

2. Understanding the Query Language

2.1 Structured Query Format

Each structured query follows this syntax:

[Comparison]F[FieldIndex][V|K][ValueOrKey] [Logical Operator]
[Comparison]F[FieldIndex][V|K][ValueOrKey]

Where:

- F[FieldIndex] → Specifies the field being queried.
- V[value] → Searches for **discrete or continuous numeric values**.
- K[key] → Searches for **integer keys representing enumerations**.
- [Comparison] → Defines how the field is compared (e.g., =, !=, <, >, etc.).

- [Logical Operator] → Connects multiple queries (&&, ||, !).

Supported Data Types

The CMR schema encodes various types of data, which must be queryable:

Data Type	Syntax Example	Meaning
Integer	F0=V25	Field 0 must be 25
Float	F1≈V3.14	Field 1 must be approximately 3.14
Enumeration (Keyed by Int)	F2=K1	Field 2 must be enum key 1
Pointer	F3->TEXEL_12	Field 3 points to texel ID TEXEL_12
Flags	F6&FLAG_A	Field 6 contains FLAG_A
Timestamp	F7>=V2024-02-01T12:00:00Z	Field 7 timestamp after Feb 1, 2024

Logical Operators

Operator	Meaning	Example
&&	AND	F0=V25 && F1>V50
||	OR	F0=V25 II F1>V50
!	NOT	!F5=K2 (Field 5 is **not** Key 2)

2.2 Schema-Based Mapping

A key requirement is mapping human language terms to schema-defined fields. Example mappings:

Human Term	Schema Field	Data Type	Example Query Syntax
Corrosion	F1 (Corrosion Type)	K (Enum)	F1=K2 (Galvanic Corrosion)
Critical Areas	F2 (Risk Level)	K (Enum)	F2=K1 (Critical)
Not Resolved	F3 (Resolution Status)	K (Enum)	F3=K0 (Unresolved)
Last 30 days	F4 (Timestamp)	V (Date)	F4>=V(Today-30)

2.3 Example Query Translation

Natural Query: 💬 *"Show me all of the areas that have galvanic corrosion on critical areas that have not been resolved in the last 30 days."*

Structured Query:

```
F1=K2 && F2=K1 && F3=K0 && F4>=V(Today-30)
```

---

3. Approaches to NLP Query Processing

We consider two possible approaches:

3.1 LLM-Based Approach

Uses a large language model (LLM) to dynamically interpret queries.

Example Prompt for an LLM:

```
You are an AI assistant that converts natural language database queries into a structured query format. You have access to the following schema:

- "corrosion type" → Field F1 (Enumerated) [Galvanic = K2, Pitting = K3]

- "component risk level" → Field F2 (Enumerated) [Critical = K1, Moderate = K2]

- "resolution status" → Field F3 (Enumerated) [Resolved = K1, Unresolved = K0]

- "inspection timestamp" → Field F4 (Date)

Given the user query: "Show me all areas with galvanic corrosion on critical areas that have not been resolved in the last 30 days."

Convert this into the structured query format (Expected LLM Output):

F1=K2 && F2=K1 && F3=K0 && F4>=V(Today-30)
```

✅ **Pros:**

- Handles complex queries and variant phrasing.
- Learns dynamically from context.
- Supports fuzzy matching.

❌ **Cons:**

- May produce hallucinated results.
- Requires cloud-based LLM access.

3.2 Rule-Based NLP Approach

Uses keyword matching to map words to schema fields.

Example Implementation in Python:

```
mappings = {
    "galvanic corrosion": "F1=K2",
    "critical areas": "F2=K1",
    "not resolved": "F3=K0",
    "last 30 days": "F4>=V(Today-30)"
}
def map_natural_to_query(natural_query):
    query_parts = [mappings[phrase] for phrase in mappings if phrase in natural_query.lower()]
    return " && ".join(query_parts)
user_query = "Show me all areas with galvanic corrosion on critical areas that have not been resolved in the last 30 days."
structured_query = map_natural_to_query(user_query)
print(structured_query)
```

✅ **Pros:**

- **Deterministic** and **accurate**.
- Runs **locally without cloud APIs**.
- **Fast execution.**

❌ **Cons:**

- Requires manual schema updates.
- Can’t handle sentence structure variations.

---

4. Project Deliverables

1. **Query Parser**
    - Parses structured query language from input text or voice commands.
    - Validates and maps schema fields.
    - Supports optional voice input for natural language queries.
2. **Query Execution System**
    - Filters CMR texture data using parsed queries.

3. **Testing Suite**
    - Validates correct translation of queries.
4. **Documentation**
    - Clear guidelines for using the system.

---

5. Next Steps

Team on the approaches: LLM-based or Rule-based.
Define schema mappings: Ensure proper field-value conversions.
Implement & test parsing logic.

Test Schema for CMR Query System


```
{
 "schema_name": "CMR_Test_Schema",
 "fields": [
  {
   "field_id": 1,
   "name": "Corrosion Type",
   "description": "The type of corrosion affecting the structure",
   "data_type": "enum",
   "values": {
    "1": "Surface Corrosion",
    "2": "Galvanic Corrosion",
    "3": "Pitting Corrosion",
    "4": "Crevice Corrosion"
   }
  },
  {
   "field_id": 2,
```

```
  "name": "Component Risk Level",
  "description": "Risk level associated with the component",
  "data_type": "enum",
  "values": {
    "1": "Critical",
    "2": "High",
    "3": "Moderate",
    "4": "Low"
  }
},
{
  "field_id": 3,
  "name": "Resolution Status",
  "description": "Current resolution status of the issue",
  "data_type": "enum",
  "values": {
    "0": "Unresolved",
    "1": "In Progress",
    "2": "Resolved"
  }
},
{
  "field_id": 4,
  "name": "Inspection Timestamp",
  "description": "The last recorded inspection time",
  "data_type": "timestamp"
},
{
  "field_id": 5,
```

```
  "name": "Thickness Loss",
  "description": "Percentage of material loss due to corrosion",
  "data_type": "float",
  "unit": "percent"
},
{
  "field_id": 6,
  "name": "Inspector ID",
  "description": "Unique identifier for the inspector",
  "data_type": "integer"
},
{
  "field_id": 7,
  "name": "Structure Type",
  "description": "The type of structure affected",
  "data_type": "enum",
  "values": {
    "1": "Hull",
    "2": "Pipeline",
    "3": "Support Beam",
    "4": "Electrical Box"
  }
},
{
  "field_id": 8,
  "name": "Corrosion Severity",
  "description": "Severity level of corrosion",
  "data_type": "enum",
  "values": {
```

```
      "1": "Minor",
      "2": "Moderate",
      "3": "Severe",
      "4": "Extreme"
    }
  },
  {
    "field_id": 9,
    "name": "Location ID",
    "description": "Reference to a location within the structure",
    "data_type": "pointer",
    "references": "Location_Schema"
  },
  {
    "field_id": 10,
    "name": "Has Protective Coating",
    "description": "Whether the component has a protective coating",
    "data_type": "boolean"
  }
 ]
}
```

Example Queries Using This Schema

1. Find all critical components with severe corrosion that have not been resolved

```
F2=K1 && F8=K3 && F3=K0
```

2. Show components with over 50% material loss

```
F5>V50
```

3. List all structures of type "Pipeline" that have been inspected in the last 60 days

```
F7=K2 && F4>=V(Today-60)
```

4. Find all hull components that are missing protective coatings

```
F7=K1 && F10=V0
```