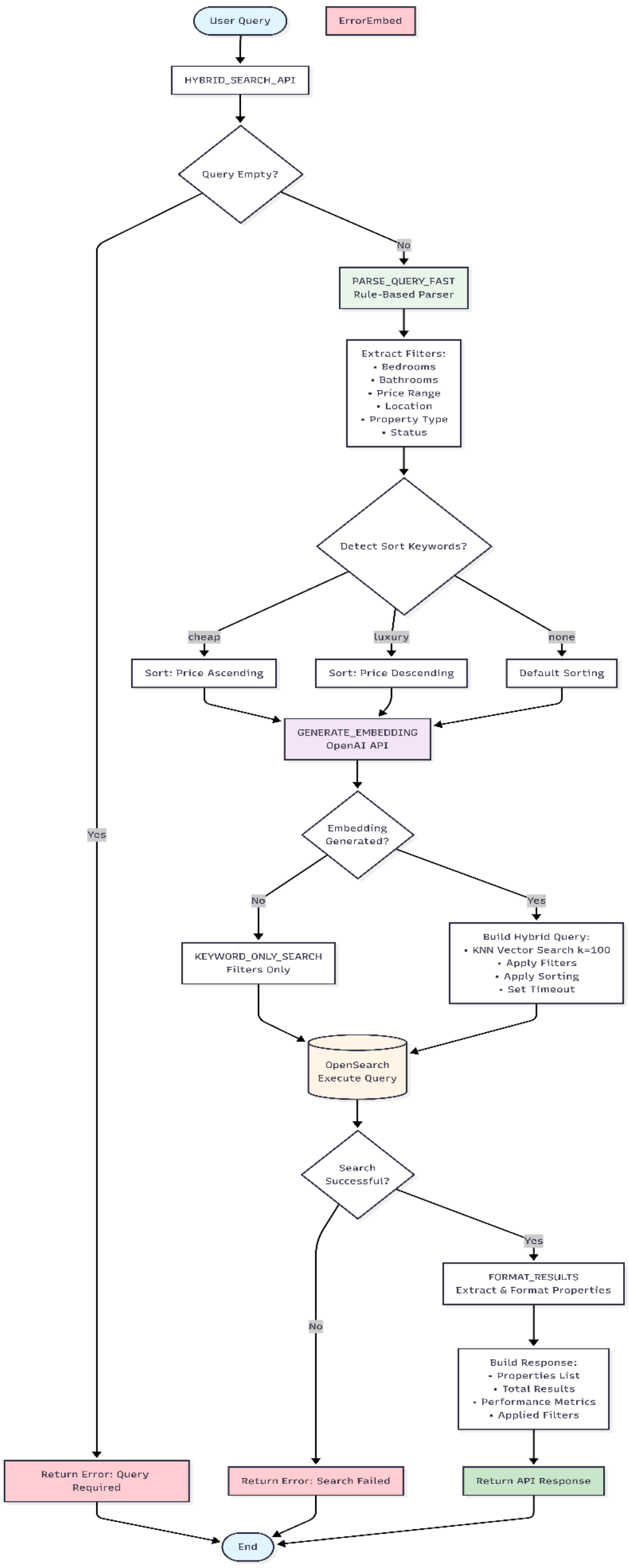
OpenSearch



User Query
ErrorEmbed
HYBRID_SEARCH_API
Query Empty?
Yes
No
PARSE_QUERY_FAST
Rule-Based Parser
Extract Filters:
• Bedrooms
• Bathrooms
• Price Range
• Location
• Property Type
• Status
Detect Sort Keywords?
cheap
luxury
none
Sort: Price Ascending
Sort: Price Descending
Default Sorting
GENERATE_EMBEDDING
OpenAI API
Embedding Generated?
No
Yes
KEYWORD_ONLY_SEARCH
Filters Only
Build Hybrid Query:
• KNN Vector Search k=100
• Apply Filters
• Apply Sorting
• Set Timeout
OpenSearch
Execute Query
Search Successful?
Yes
No
FORMAT_RESULTS
Extract & Format Properties
Build Response:
• Properties List
• Total Results
• Performance Metrics
• Applied Filters
Return Error: Query Required
Return Error: Search Failed
Return API Response
End

HYBRID SEARCH:

```
FUNCTION HYBRID_SEARCH_API(request):

    INPUT:
        user_query
        page
        size
        use_cache

    START total_timer

    IF user_query is empty:
        RETURN error "Query required"

    IF use_cache:
        cache_key = HASH(user_query + page)
        IF cache_key exists AND not expired:
            RETURN cached_response with method = "cached"

    START parse_timer
    parsed_filters = PARSE_QUERY_FAST(user_query)
    STOP parse_timer

    START search_timer
    results = HYBRID_SEARCH(user_query, parsed_filters, page, size)
    STOP search_timer

    IF results is empty:
        RETURN error "Search failed"

    formatted_results = FORMAT_RESULTS(results)

    IF use_cache AND page == 1:
        STORE formatted_results in cache

    STOP total_timer

    RETURN response:
        query
        total_results
        formatted_results
        performance_metrics
        applied_filters
```

Fast Rule-Based Query Parsing:

```
FUNCTION PARSE_QUERY_FAST(user_query):

    normalized_query = LOWERCASE(user_query)

    filters.must = []
    filters.filter = []
    sort = NULL

    IF bedroom pattern found:
        ADD term filter (bedrooms)

    IF bathroom pattern found:
        ADD term filter (bathrooms)

    IF price "under / below":
        ADD range filter (price <= X)

    IF price "over / above":
        ADD range filter (price >= X)

    IF city found:
        ADD term filter (city)

    IF state code found:
        ADD term filter (state)

    IF property type mentioned:
        ADD term filter (propertyType)

    IF status mentioned:
        ADD filter (status)
    ELSE:
        ADD filter (status = Active)

    IF keywords like "cheap":
        sort = price ascending
    ELSE IF keywords like "luxury":
        sort = price descending

    RETURN {
        filters,
        sort
    }
```

Hybrid Search Logic (Semantic + Filters):

```
FUNCTION HYBRID_SEARCH(user_query, parsed_filters, page, size):

  vector = GENERATE_EMBEDDING(user_query)

  IF vector is NULL:
    RETURN KEYWORD_ONLY_SEARCH(parsed_filters, page, size)

  query_body = {
    size,
    offset,
    semantic_knn(vector, k=100),
    filters (from parsed_filters),
    sorting (optional),
    timeout
  }

  TRY:
    response = OPENSEARCH_SEARCH(query_body)
    RETURN response
  CATCH error:
    RETURN NULL
```

Cache Management:

```
FUNCTION GET_CACHE_KEY(query):
  RETURN MD5_HASH(query)

FUNCTION CLEAN_CACHE():
  FOR each key in cache:
    IF current_time - cached_time > TTL:
      DELETE key
```

Similar Property Search (Vector-Only):

```
FUNCTION FIND_SIMILAR(listing_id):
  source_listing = GET listing from OpenSearch
  IF no embedding:
    RETURN error
  query = KNN_SEARCH(
    vector = source_listing.embedding,
    exclude = listing_id
  )
  results = OPENSEARCH_SEARCH(query)
  RETURN formatted_result
```

Response Formatting:

```
FUNCTION FORMAT_RESULTS(search_results):

  properties = []

  FOR each hit in search_results:
    property = {
      id,
      price,
      address,
      bedrooms,
      bathrooms,
      squareFeet,
      photos (first 3),
      status,
      short_description,
      relevance_score
    }
    ADD property to properties

  RETURN properties
```

Architecture Summary:

User Query
  ↓
Cache Check
  ↓
Fast Rule Parser (regex)
  ↓
Embedding Generation (OpenAI)
  ↓
Hybrid OpenSearch Query
  ↓
Ranking + Filters
  ↓
Formatted API Response

Property Indexing with Vector Embeddings pipeline

Indexing Pipeline:

```
MAIN():
  CREATE OpenSearch index with vector support

  LOAD property dataset from JSON file

  IF dataset is empty:
    EXIT program

  CONFIRM indexing with user

  FOR each batch of properties:
    FOR each property in batch:
      INDEX_SINGLE_PROPERTY(property)

    REFRESH OpenSearch index
    WAIT briefly (rate-limit protection)

  VERIFY indexed data
  PRINT success message
```

Index Creation:

```
FUNCTION CREATE_INDEX():
  IF index already exists:
    ASK user whether to delete
    IF user says no:
      RETURN

  DEFINE index settings:
    - shards = 1
    - replicas = 0
    - enable kNN
    - HNSW parameters

  DEFINE mappings:
    - structured fields (price, beds, city, status)
    - text fields (description)
    - geo_point (lat/lon)
    - knn_vector field (embedding)

  CREATE index in OpenSearch
```

Dataset Loading:

```
FUNCTION LOAD_DATASET(file_path):

    READ JSON file

    IF data is list:
        RETURN list

    IF data is object:
        TRY common keys (properties, listings, data)
        ELSE treat as single property

    RETURN property list
```

Property Description Generation:

```
FUNCTION CREATE_PROPERTY_DESCRIPTION(property):

    EXTRACT:
        - listingId
        - price, status
        - bedrooms, bathrooms, sqft
        - property type, style, year built
        - location (city, state, neighborhood)
        - amenities, interior, exterior features

    BUILD natural-language text:
        "3 bedroom, 2 bathroom condo in Austin, TX..."
        "Built in 2019 with modern interior..."
        "Available for immediate showing..."

    RETURN description text
```

Embedding Generation:

```
FUNCTION GENERATE_EMBEDDING(text):

    CALL OpenAI embedding API with text

    IF API fails:
        RETURN null

    RETURN vector (length = 1536)
```

Indexing a Single Property:

```
FUNCTION INDEX_SINGLE_PROPERTY(property):

    description = CREATE_PROPERTY_DESCRIPTION(property)
    vector = GENERATE_EMBEDDING(description)

    IF vector is null:
        SKIP property
        RETURN failure

    ADD description to property
    ADD vector to property

    IF latitude & longitude exist:
        CREATE geo_point field

    INDEX property document into OpenSearch
```

Batch Indexing Strategy:

```
FUNCTION BULK_INDEX(properties, batch_size):

    FOR i from 0 to total_properties STEP batch_size:
        batch = properties[i : i + batch_size]

        FOR each property in batch:
            success = INDEX_SINGLE_PROPERTY(property)
            UPDATE counters

        REFRESH index
        LOG batch progress
        SLEEP briefly
```

Index Verification:

```
FUNCTION VERIFY_INDEX():
    COUNT documents in index
    IF count > 0:
        FETCH one sample document
        CHECK:
            - description exists
            - vector dimension is correct
            - geo_point exists

    RETURN verification status
```

Model:

Raw MLS JSON

↓

Human-like Description Text

↓

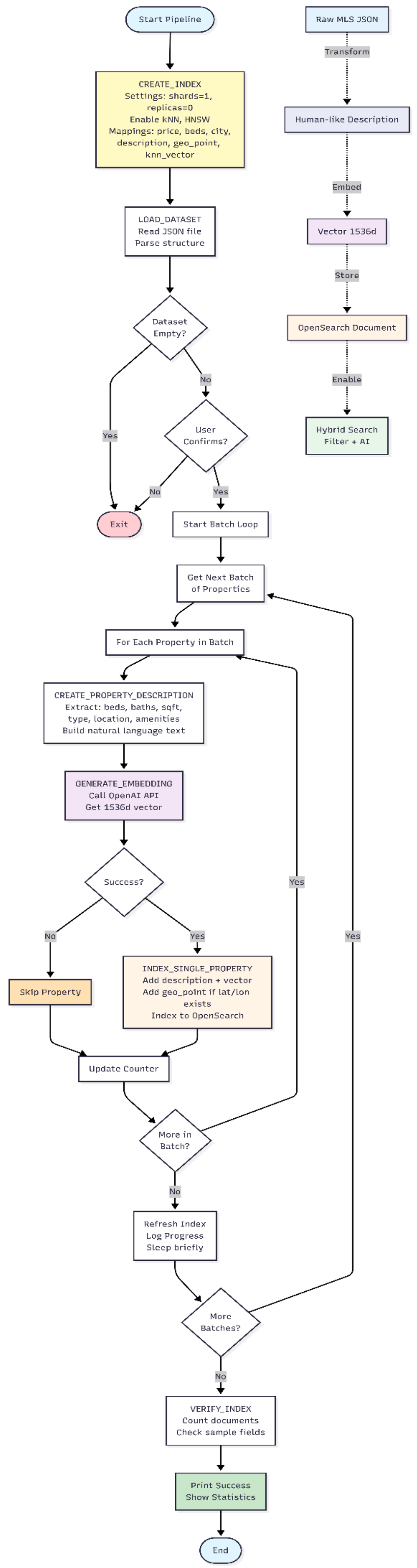
Vector Embedding (Meaning)

↓

OpenSearch Document

↓

Hybrid Search (Filter + AI)


Start Pipeline
CREATE_INDEX
Settings: shards=1, replicas=0
Enable kNN, HNSW
Mappings: price, beds, city, description, geo_point, knn_vector
LOAD_DATASET
Read JSON file
Parse structure
Dataset Empty?
Yes
No
User Confirms?
No
Yes
Exit
Start Batch Loop
Get Next Batch of Properties
For Each Property in Batch
CREATE_PROPERTY_DESCRIPTION
Extract: beds, baths, sqft, type, location, amenities
Build natural language text
GENERATE_EMBEDDING
Call OpenAI API
Get 1536d vector
Success?
No
Yes
Skip Property
INDEX_SINGLE_PROPERTY
Add description + vector
Add geo_point if lat/lon exists
Index to OpenSearch
Update Counter
More in Batch?
Yes
No
Refresh Index
Log Progress
Sleep briefly
More Batches?
Yes
No
VERIFY_INDEX
Count documents
Check sample fields
Print Success
Show Statistics
End
Raw MLS JSON
Transform
Human-like Description
Embed
Vector 1536d
Store
OpenSearch Document
Enable
Hybrid Search
Filter + AI