LLM (Large Language Model) Usage Report

CTA Evaluation System - Comprehensive Analysis

Document Version: 1.0 | Date: 2025-11-16
Author: CTA Evaluation System Documentation Team

LLM (LARGE LANGUAGE MODEL) USAGE REPORT

CTA Evaluation System - Comprehensive Analysis

Document Version: 1.0

Date: 2024

Author: CTA Evaluation System Documentation Team

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

The CTA Evaluation System extensively utilizes Large Language Models (LLMs), specifically OpenAI's GPT models (GPT-4, GPT-4o, GPT-3.5-turbo), to perform sophisticated linguistic analysis tasks that would be extremely difficult or impossible to accomplish with traditional rule-based systems alone.

KEY FINDINGS:

- LLMs are used in 4 core analysis modules
- All LLM interactions are centralized through a unified LLMInterface class
- GPT-4o is the recommended model (default: gpt-4o)
- Temperature settings range from 0.1 (deterministic) to 0.3 (creative analysis)

- JSON-structured outputs are enforced for all LLM responses
- Error-tolerant JSON parsing handles various response formats

PRIMARY USE CASES:

1. Context-aware transliteration from 6 Turkic language alphabets to CTA
2. Phonetic ambiguity detection and risk analysis
3. Cognate word alignment and etymological analysis
4. Phonetic correspondence effectiveness measurement

WHY LLMs ARE ESSENTIAL:

- Context-aware phonetic analysis requires understanding linguistic nuances
- Cross-linguistic pattern recognition across Turkic languages
- Morphophonological environment consideration
- Systematic sound change detection
- Academic-quality explanations and rationales

2. LLM INTEGRATION ARCHITECTURE

2.1 Centralized Llm Interface

Location: modules/llm_interface.py

Class: LLMInterface

The LLMInterface class provides a unified, secure interface to OpenAI's API.
All modules use this single interface, ensuring consistency and maintainability.

KEY FEATURES:

- Secure API key management (from config.py or environment variables)
- Model selection (default: gpt-4o)
- Temperature and max_tokens configuration
- Response time measurement
- Token usage tracking
- Error handling and retry logic
- JSON parsing with multiple fallback strategies

ARCHITECTURE FLOW:

User Input → Module (Transliterator/RiskAnalyzer/etc.)
→ LLMInterface.call_llm()
→ OpenAI API
→ Response Parsing
→ JSON Extraction

→ Result Validation

→ Module Processing

→ Final Output

2.2 Configuration

Location: config.py

Configuration parameters:

- OPENAI_API_KEY: User's OpenAI API key
- OPENAI_MODEL: Model selection (default: "gpt-4o")
- DEFAULT_TEMPERATURE: 0.3 (balanced creativity/consistency)
- DEFAULT_MAX_TOKENS: 2000
- REQUEST_TIMEOUT: 30 seconds

2.3 Model Selection

Supported Models:

- gpt-4o (Recommended - latest, fast, high performance)
- gpt-4o-mini (More economical, fast alternative)
- gpt-4-turbo (High performance)
- gpt-4 (Standard GPT-4)
- gpt-3.5-turbo (Most economical option)

The system defaults to gpt-4o for optimal balance of performance and cost.

3. LLM USAGE BY MODULE

3.1 Transliteration Engine

Location: modules/transliterator.py

Class: TransliteratorEngine

Method: transliterate()

PURPOSE:

Converts text from different Turkic language alphabets (Turkish, Uzbek Latin, Kazakh Cyrillic, Azerbaijani Latin, Turkmen Latin, Kyrgyz Cyrillic) to the Common Turkic Alphabet (CTA).

WHY LLM IS NEEDED:

- Context-aware phonetic conversion requires understanding word meanings
- Morphophonological environment analysis (vowel harmony, consonant clusters)

- Systematic application of 5 new CTA letters (q, x, ñ, ə→ä, û)
- Handling ambiguous cases where rule-based mapping is insufficient
- Generating Turkish-language explanations of conversions

HOW LLM IS USED:

1. System Prompt: Defines CTA rules, mapping tables, and output format
2. User Prompt: Contains input text and source language
3. LLM analyzes phonetic context and applies CTA rules
4. Returns JSON with transliterated text and Turkish notes

EXAMPLE SYSTEM PROMPT EXCERPT:

```
ROLE: You are a transliteration engine for the Common Turkic Alphabet (CTA). CTA
RULES:
```

1. Turkish (29 letters) preserved: a,b,c,ç,d,e,f,g,ğ,h,ı,i,j,k,l,m,n,o,ö,p,r,s,ş,t,u,ü,v,y,z
2. Apply 5 NEW letters systematically:

- q: Kalın "K" sesi - kalın ünlülerle birlikte: Qazaq (Kazak)
- x: Gırtlaksı "H" (hırıltılı, Arapça ج sesi): Xəbər (Haber)
- ə: Açık/öne yakın "E" (e ile a arası): Ədəbiyyat (Edebiyat)
- ñ: Genizsi "N" (nazal n): Teñri (Tengri)
- û: Uzatmalı "U" (uzun ünlü göstergesi): Sû (uzun su sesi)

EXAMPLE USER PROMPT:

```
INPUT_TEXT: herkes geldi SOURCE_LANGUAGE: Turkish TASK: Transliterate from
Turkish to CTA following these rules:
```

```
1. Apply CTA systematic mappings for Turkish
2. Use new CTA letters (q, x, ñ, ə→ä, û) where phonetically appropriate
3. Preserve Turkish alphabet letters (29 letters) as base
4. Write notes in Turkish explaining changes made
```

CRITICAL: Return ONLY the JSON object, no additional text:

{"ok": true, "output_text": "...", "notes": ["Türkçe açıklama..."]}

EXAMPLE LLM RESPONSE:

```
{ "ok": true, "output_text": "hərkəs gəldi", "notes": [ "Türkçe metinde CTA fonetik
iyileştirmeleri uygulandı:", "- 'herkes' → 'hərkəs' (ə harfi açık e sesi için kullanıldı)", "- 'geldi' →
'gəldi' (ə harfi açık e sesi için kullanıldı)" ] }
```

```
LLM PARAMETERS:
- Temperature: 0.1 (low for consistency)
- Max Tokens: 2000
- Model: gpt-4o (default)
```

POST-PROCESSING:

- JSON parsing with error-tolerant extraction

- CTA character set validation

- Statistics calculation (input/output length, new letter usage)

- LLM metadata tracking (model, tokens, response time)

TECHNICAL DETAILS:

- File: modules/transliterator.py, lines 233-329

- LLM call: self.llm.call_llm(system_prompt, user_prompt, temperature=0.1, max_tokens=2000)

- Response parsing: self.llm.parse_json_response(response["content"])

- Validation: self.llm.validate_ota_characters(output_text)

3.2 Phonetic Risk Analyzer

Location: modules/risk_analyzer.py

Class: RiskAnalyzer

Method: analyze_risks()

PURPOSE:

Detects phonetic ambiguities and potential confusions in CTA-normalized text, specifically focusing on the 5 new CTA letters (x, ə, q, ñ, û).

WHY LLM IS NEEDED:

- Phonetic ambiguity detection requires linguistic knowledge

- Cross-linguistic confusion pattern recognition

- Morphophonological environment analysis

- Context-dependent risk assessment

- Systematic vs. sporadic confusion identification

- Academic-quality explanations in Turkish

HOW LLM IS USED:

1. System Prompt: Defines risk analysis criteria for each new letter

2. User Prompt: Contains CTA text and detected new letters

3. LLM analyzes phonetic contexts and identifies potential confusions

4. Returns JSON array of risk objects with confidence scores

"examples": ["example", "words", "showing", "confusion"],

```
"confidence": 0.0-1.0, "context": "brief explanation of when/why confusion
occurs", "severity": "low|medium|high" }
```

EXAMPLE USER PROMPT:

CTA_TEXT: "hərkəs xabər okuyor" DETECTED_NEW_LETTERS: ə, x TASK: Analyze the phonetic risks and potential confusions for the new CTA letters found in this text. Focus particularly on: ə, x Consider:

1. Cross-linguistic confusion patterns
2. Morphophonological environments
3. Dialectal variations
4. L1 transfer effects
5. Systematic vs. sporadic confusions

Return JSON array of risk analysis objects as specified in the system prompt.

"examples": ["xabər"],

"confidence"	0.8,
"context"	"Gırtlaksı h sesi ile normal h sesi karışımı, Arapça kökenli kelimelerde",
"severity"	"high"

```
LLM PARAMETERS:
- Temperature: 0.2 (low for consistent analysis)
- Max Tokens: 2000
- Model: gpt-4o (default)
```

POST-PROCESSING:

- JSON parsing (handles array or single object)

- Risk validation and enrichment

- Statistics calculation (severity distribution, average confidence)

- Language coverage analysis

TECHNICAL DETAILS:

- File: modules/risk_analyzer.py, lines 52-149

- LLM call: self.llm.call_llm(system_prompt, user_prompt, temperature=0.2, max_tokens=2000)

- Response parsing: self.llm.parse_json_response(response["content"])

- Validation: _validate_and_enrich_risk() method

3.3 Cognate Aligner

Location: modules/cognate_aligner.py

Class: CognateAligner

Method: align_cognates()

PURPOSE:

Aligns cognate words from different Turkic languages after CTA transliteration, grouping them by phonetic similarity levels (high/medium/low) and providing etymological analysis.

WHY LLM IS NEEDED:

- Phonetic similarity analysis requires understanding sound correspondences
- Systematic sound change pattern recognition (k~q, sh~ş, e~ä, etc.)
- Morphological analysis (root + suffix separation)
- Historical linguistics knowledge (Proto-Turkic roots)
- Cross-linguistic relationship detection
- Academic-quality explanations in Turkish

HOW LLM IS USED:

1. System Prompt: Defines similarity criteria and sound change patterns
2. User Prompt: Contains candidate words in format "lang_code:word"
3. LLM transliterates each word to CTA, then groups by similarity
4. Returns JSON with similarity groups, rationales, and root analyses

EXAMPLE SYSTEM PROMPT EXCERPT:

```
ROLE: You align potential cognates across Turkic languages AFTER CTA transliteration. PROCESS: 1) Transliterate each candidate word into CTA using the same mapping rules 2) Group words by phonetic similarity levels: "high" | "medium" | "low" 3) Add brief rationale explaining the relationship (shared root, systematic sound change) SIMILARITY CRITERIA: - HIGH: Clear cognates with systematic sound correspondences (e.g., tr:şehir ~ uz:şahar) - MEDIUM: Probable cognates with some sound changes or semantic drift - LOW: Possible distant relationships or borrowings SOUND CHANGE PATTERNS TO RECOGNIZE: - Consonant shifts: k~q (front/back), g~ğ, sh~ş, ch~ç, x~h - Vowel changes: e~ä, o~ò, u~ü, ı~i (vowel harmony effects) - Length variations: û marker differences - Nasal variations: n~ñ (ng realization) OUTPUT FORMAT (JSON only): { "groups": [ { "similarity": "high|medium|low", "items": [ {"lang": "language_code", "original": "original_word", "ota": "ota_form"} ], "rationale": "TÜRKÇE açıklama - ses değişimleri ve ilişkiler", "confidence": 0.0-1.0, "root_analysis": "TÜRKÇE kök analizi - ortak köken açıklaması" } ] }
```

EXAMPLE USER PROMPT:

```
CANDIDATE WORDS: - tr:şehir (Turkish) - az:şəhər (Azerbaijani) - uz:şahar (Uzbek) TASK: 1) Transliterate each word according to CTA rules 2) Group by phonetic similarity after CTA normalization 3) Identify systematic sound correspondences 4) Provide confidence scores and rationales FOCUS POINTS: - Systematic sound changes (k~q, sh~ş, etc.) - Morphological analysis (root + affixes) - Semantic relationships - Historical sound correspondences CRITICAL: Write ALL "rationale" and "root_analysis" fields in TURKISH!
```

EXAMPLE LLM RESPONSE:

{ "groups": [{ "similarity": "high", "items": [{"lang": "tr", "original": "şehir", "ota": "şəhər"}, {"lang": "az", "original": "şəhər", "ota": "şəhər"}, {"lang": "uz", "original": "şahar", "ota": "şəhər"}], "rationale": "Ortak Türkçe kök, sistematik ses uyuşması (sh↔ş, e↔ə↔a)", "confidence": 0.9, "root_analysis": "Proto-Türkçe *şəhər kökünden türemiş, şehir anlamında" }], "statistics": { "total_words": 3, "groups_found": 1, "average_group_size": 3 } }

LLM PARAMETERS:

- Temperature: 0.3 (balanced for creative but consistent analysis)
- Max Tokens: 3000 (longer responses for multiple groups)
- Model: gpt-4o (default)

POST-PROCESSING:

- JSON parsing and validation
- Group structure validation
- Statistics calculation (alignment rate, similarity distribution)
- Language coverage analysis

TECHNICAL DETAILS:

- File: modules/cognate_aligner.py, lines 80-144
- LLM call: self.llm.call_llm(system_prompt, user_prompt, temperature=0.3, max_tokens=3000)
- Response parsing: self.llm.parse_json_response(response["content"])
- Validation: _validate_and_enrich_result() method

3.4 PCE (PHONETIC CORRESPONDENCE EFFECTIVENESS) ANALYZER

Location: modules/pce_analyzer.py

Class: PCEAnalyzer

Method: analyze_pce()

PURPOSE:

Objectively measures and compares the phonetic representation effectiveness of writing systems using academic metrics (UPC, TPC, ALC) and calculates improvement rates.

WHY LLM IS NEEDED:

- Phoneme extraction from text requires linguistic knowledge
- Distinguishing between unique and total phoneme counts
- Understanding phonetic structures and representations
- Calculating academic metrics accurately
- Providing detailed Turkish explanations of improvements

HOW LLM IS USED:

1. System Prompt: Defines PCE formulas and calculation methods
2. User Prompt: Contains original text and CTA text

3. LLM extracts phonemes, calculates metrics, and compares effectiveness

4. Returns JSON with detailed analysis and improvement percentages

EXAMPLE SYSTEM PROMPT EXCERPT:

```
ROLE: You are a PCE (Phonetic Correspondence Effectiveness) analyzer for writing
systems. PCE measures how effectively a writing system represents phonetic
structures: - UPC (Unique Phoneme Count): Number of distinct phonemes - TPC
(Total Phoneme Count): Total phonemes in text - ALC (Average Letter Count):
Average letters per phoneme PCE FORMULAS (from academic paper):
```

1. Weighted PCE = (UPC/TPC) × ALC × SF

where SF (Scale Factor) = large multiplier (typically 100-1000)

2. Logarithmic PCE = (log(1+UPC)/log(1+TPC)) × (1/ALC) × SF

3. Improvement % = ((CTA_PCE - Original_PCE) / Original_PCE) × 100

Expected results: 18-19% improvement (weighted), 11-12% improvement (logarithmic)

OUTPUT FORMAT (JSON only):

{

"original_analysis": {

"upc": number,

"tpc": number,

"alc": number,

"weighted_pce": number (using SF=100),

"logarithmic_pce": number (using SF=100)

},

"ota_analysis": {

"upc": number,

"tpc": number,

"alc": number,

"weighted_pce": number (using SF=100),

"logarithmic_pce": number (using SF=100)

},

"comparison": {

"weighted_improvement": percentage (18-19% expected),

"logarithmic_improvement": percentage (11-12% expected),

"effectiveness_gain": overall percentage

},

"detailed_analysis": "Turkish explanation of phonetic improvements"

}

EXAMPLE USER PROMPT:

```
ORIGINAL_TEXT: herkes geldi CTA_TEXT: hərkəs gəldi DATASET: Custom_Dataset TASK:
Perform ACCURATE PCE analysis using the correct formulas:
```

```
1. Count phonemes in both texts:
- UPC: Unique distinct phonemes (count each phoneme type once)
- TPC: Total phonemes in text (count all phoneme instances)
- ALC: Average letters per phoneme = Total_Letters / TPC
```

2. Calculate PCE using CORRECT FORMULAS:

- Weighted PCE = (UPC/TPC) × ALC × 100

- Logarithmic PCE = (log(1+UPC)/log(1+TPC)) × (1/ALC) × 100

3. Calculate improvement percentage:

- Improvement % = ((CTA_PCE - Original_PCE) / Original_PCE) × 100

EXPECTED RESULTS (based on academic research):

- Weighted PCE improvement: 18-19%

- Logarithmic PCE improvement: 11-12%

CRITICAL: Use Scale Factor = 100 to get meaningful PCE values.

Focus on how CTA letters (q, x, ñ, ə, û) increase UPC and improve phonetic representation.

Return JSON with complete PCE metrics and Turkish explanations.

EXAMPLE LLM RESPONSE:


```
{ "original_analysis": { "upc": 8, "tpc": 12, "alc": 1.2, "weighted_pce": 80.0, "logarithmic_pce":
65.5 }, "ota_analysis": { "upc": 9, "tpc": 12, "alc": 1.15, "weighted_pce": 86.25,
"logarithmic_pce": 72.8 }, "comparison": { "weighted_improvement": 7.81,
"logarithmic_improvement": 11.15, "effectiveness_gain": 9.48 }, "detailed_analysis": "CTA
metni ə harfi sayesinde daha fazla benzersiz fonem içeriyor, bu da fonetik temsil etkinliğini
artırıyor." }
```


```
LLM PARAMETERS:
- Temperature: 0.1 (very low for accurate calculations)
- Max Tokens: 3000 (detailed analysis)
- Model: gpt-4o (default)
```

POST-PROCESSING:

- JSON parsing and validation

- Result enrichment (metadata, assessment, recommendations)

- New letter detection

- Effectiveness rating

TECHNICAL DETAILS:

- File: modules/pce_analyzer.py, lines 79-154

- LLM call: self.llm.call_llm(system_prompt, user_prompt, temperature=0.1, max_tokens=3000)

- Response parsing: self.llm.parse_json_response(response["content"])
- Enrichment: _enrich_pce_result() method

4. TECHNICAL IMPLEMENTATION DETAILS

4.1 Llm Interface Class

Location: modules/llm_interface.py

CORE METHODS:

1. call_llm(system_prompt, user_prompt, temperature, max_tokens)
- Makes API call to OpenAI
- Returns structured response with content, model, usage, response_time
- Handles errors gracefully

2. parse_json_response(response_content)
- Error-tolerant JSON parsing
- Handles multiple formats (array, object, wrapped in markdown)
- Regex fallback for malformed JSON
- Returns parsed JSON object

3. validate_ota_characters(text)
- Validates CTA character set compliance
- Returns (is_valid, invalid_chars) tuple

4. check_idempotency(input_text, output_text, mapping_rules)
- Validates transliteration consistency
- Currently returns True (placeholder for future implementation)

4.2 Json Parsing Strategy

The system uses a multi-layered JSON parsing approach:

1. Direct JSON parsing (json.loads)
2. Markdown code block extraction (```json ... ```)
3. Prefix removal (OUTPUT:, OUTPUT_TEXT:)
4. Regex-based extraction (array/object patterns)
5. Fallback regex with multiline support

This ensures robust handling of various LLM response formats.

4.3 Error Handling

- API key validation before LLM calls
- Try-catch blocks around all LLM interactions
- Graceful degradation on errors
- User-friendly error messages
- Debug logging for troubleshooting

4.4 Response Validation

Each module validates LLM responses:
- Required fields check
- Data type validation
- Value range validation (confidence: 0.0-1.0)
- Structure validation (groups, items, etc.)
- Default value assignment for missing fields

5. PROMPT ENGINEERING STRATEGIES

5.1 System Prompts

System prompts are carefully crafted to:
- Define the LLM's role clearly
- Specify output format requirements
- Provide examples and guidelines
- Set language requirements (Turkish explanations)
- Include academic terminology

5.2 User Prompts

User prompts include:
- Clear task descriptions
- Input data (text, language codes, etc.)
- Specific instructions
- Format requirements
- Critical reminders

5.3 Temperature Settings

Temperature is adjusted per task:
- 0.1: Transliteration, PCE Analysis (deterministic, consistent)
- 0.2: Risk Analysis (consistent but allows some variation)
- 0.3: Cognate Alignment (balanced creativity/consistency)

5.4 Output Format Enforcement

All prompts emphasize:

- JSON-only output
- No additional text before/after JSON
- Specific field requirements
- Turkish language for explanations
- Academic terminology

5.5 FEW-SHOT EXAMPLES

Prompts include examples:

```
- Correct transliteration examples - Risk analysis examples - Cognate grouping
examples - PCE calculation examples ========================================
====================================
```

6. PERFORMANCE & OPTIMIZATION

6.1 Token Usage

Token tracking is implemented for:

- Cost monitoring
- Performance optimization
- Response length management

Typical token usage:

- Transliteration: 500-1500 tokens
- Risk Analysis: 800-2000 tokens
- Cognate Alignment: 1500-3000 tokens
- PCE Analysis: 2000-3000 tokens

6.2 Response Time

Response times are measured and tracked:

- Average: 2-5 seconds per request
- Depends on text length and model
- GPT-4o is faster than GPT-4
- Network latency affects times

6.3 Optimization Strategies

- Prompt length optimization
- Max tokens limits per task

- Caching considerations (future)
- Batch processing support (future)

6.4 Cost Considerations

- GPT-4o recommended for best cost/performance
- GPT-4o-mini for economical use
- Temperature settings affect token usage
- Max tokens limits prevent excessive costs

7. EXAMPLES & USE CASES

7.1 Transliteration Example

Input:
Text: "herkes geldi"
Language: Turkish

LLM Processing:
1. Analyzes phonetic structure
2. Identifies open e sounds
3. Applies ə letter for open e's
4. Generates Turkish explanation

Output:
{
"ok": true,
"output_text": "hərkəs gəldi",
"notes": [
"Türkçe metinde CTA fonetik iyileştirmeleri uygulandı:",
"- 'herkes' → 'hərkəs' (ə harfi açık e sesi için)",
"- 'geldi' → 'gəldi' (ə harfi açık e sesi için)"
]
}

7.2 Risk Analysis Example

Input:
CTA Text: "hərkəs xabər okuyor"

LLM Processing:
1. Detects new letters: ə, x
2. Analyzes phonetic contexts
3. Identifies potential confusions

4. Assigns confidence scores and severity

`Output:`

[

{

"letter": "ə",

"possible_confusions": ["e", "a"],

"languages": ["tr", "az"],

"examples": ["xabər"],

"confidence"	0.8,
"context"	"Gırtlaksı h sesi ile normal h sesi karışımı",
"severity"	"high"
Input	
Candidates	
tr	şehir
az	şəhər
uz	şahar
LLM Processing	

1. Transliterates each word to CTA
2. Analyzes phonetic similarities
3. Groups by similarity level
4. Provides etymological analysis

`Output:`

{

"groups": [

{

"similarity": "high",

"items": [

{"lang": "tr", "original": "şehir", "ota": "şəhər"},

{"lang": "az", "original": "şəhər", "ota": "şəhər"},

{"lang": "uz", "original": "şahar", "ota": "şəhər"}

],

"rationale": "Ortak Türkçe kök, sistematik ses uyuşması",
"confidence": 0.9,
"root_analysis": "Proto-Türkçe *şəhər kökünden"
}
]
}

7.4 Pce Analysis Example

`Input:`
Original: "herkes geldi"
CTA: "hərkəs gəldi"

LLM Processing:
1. Extracts phonemes from both texts
2. Calculates UPC, TPC, ALC
3. Computes Weighted and Logarithmic PCE
4. Calculates improvement percentages

`Output:`
{
"original_analysis": {
"upc": 8,
"tpc": 12,
"alc": 1.2,
"weighted_pce": 80.0,
"logarithmic_pce": 65.5
},
"ota_analysis": {
"upc": 9,
"tpc": 12,
"alc": 1.15,
"weighted_pce": 86.25,
"logarithmic_pce": 72.8
},
"comparison": {
"weighted_improvement": 7.81,
"logarithmic_improvement": 11.15
}
}

8. CONCLUSION

The CTA Evaluation System demonstrates sophisticated use of Large Language Models (LLMs) for academic linguistic research. The system leverages GPT models' capabilities in:

1. CONTEXT-AWARE ANALYSIS
- Understanding phonetic nuances
- Recognizing cross-linguistic patterns
- Analyzing morphophonological environments

2. SYSTEMATIC PROCESSING

- Consistent application of CTA rules
- Structured output generation
- Academic-quality explanations

3. MULTILINGUAL SUPPORT

- Handling 6 different Turkic language alphabets
- Cross-linguistic comparison
- Etymological analysis

4. ACADEMIC RIGOR

- Quantitative metrics (PCE)
- Risk assessment with confidence scores
- Detailed documentation and explanations

KEY STRENGTHS:
- Centralized LLM interface ensures consistency
- Error-tolerant JSON parsing handles variations
- Temperature optimization per task type
- Comprehensive validation and enrichment
- Academic-quality output

FUTURE ENHANCEMENTS:
- Response caching for repeated queries
- Batch processing optimization
- Custom fine-tuned models for specific tasks

- Enhanced error recovery mechanisms
- Performance monitoring dashboard

The integration of LLMs enables the system to perform complex linguistic analysis that would be extremely difficult with traditional rule-based approaches, while maintaining academic standards and reproducibility.

END OF REPORT