Error Handling in LLM Applications

1. Introduction to Error Handling in LLM Applications

Error handling is a critical component of any production-ready Large Language Model (LLM) application. Since LLMs depend on probabilistic models, external APIs, and network-based services, failures can occur at multiple stages. Proper error handling ensures system stability, graceful degradation, and a reliable user experience.

2. Why Error Handling is Important

Without robust error handling, LLM applications may crash, return misleading outputs, or expose sensitive information. Error handling helps in:

- Maintaining application uptime
- Improving user trust
- Preventing cascading failures
- Simplifying debugging and monitoring

3. Categories of Errors in LLM Systems

3.1 Input Validation Errors

These errors occur when user input is invalid or malformed.

Examples:

- Empty or null prompts
- Unsupported characters or encoding
- Excessively long inputs
- Invalid request formats

Mitigation:

- Validate input length and format
- Reject empty prompts early
- Apply schema validation before API calls

3.2 Token and Context Window Errors

LLMs have fixed context limits. Exceeding these limits causes request failures.

Examples:

- Context length exceeded
- Token overflow errors

Mitigation:

- Chunk large text inputs
- Use sliding window techniques
- Summarize text before processing

3.3 API-Level Errors

These arise from interactions with external LLM providers.
Common API errors:
• 401 – Unauthorized (Invalid API key)
• 403 – Forbidden (Access denied)
• 429 – Too Many Requests (Rate limit exceeded)
• 500/503 – Server-side errors

Mitigation:
• Validate API keys
• Implement retry logic with exponential backoff
• Respect rate limits
• Handle server errors gracefully

3.4 Model Output Errors

LLMs may return unexpected or invalid outputs.
Examples:
• Hallucinated or factually incorrect responses
• Ignoring prompt instructions
• Invalid JSON or structured output
• Partial or truncated responses

Mitigation:
• Enforce strict prompt instructions
• Validate response format
• Apply post-processing checks
• Use fallback responses when validation fails

3.5 Network and Infrastructure Errors

These errors are caused by connectivity or infrastructure issues.
Examples:
• Network timeouts
• DNS resolution failures
• SSL certificate issues
• Proxy or firewall restrictions

Mitigation:
• Set request timeouts
• Retry failed requests selectively
• Monitor network health
• Implement fallback mechanisms

4. Retry and Recovery Strategies

4.1 Retry Logic

Retries should be applied only to recoverable errors.
Recoverable:
• Rate limit errors
• Temporary network failures
• Transient server issues

Non-recoverable:
• Invalid input
• Authentication failures

4.2 Exponential Backoff

Gradually increase wait time between retries to avoid overwhelming APIs.

5. Graceful Degradation and Fallback Handling

When errors persist:
• Return cached responses
• Switch to smaller or local models
• Provide user-friendly error messages
• Disable non-critical features temporarily

6. Logging and Monitoring Errors

Proper logging is essential for diagnosing issues.
Log:
• Error type and code
• Timestamp
• Request identifiers
• Response latency

Avoid logging:
• API keys
• Personal user data
• Confidential prompts

7. Security Considerations in Error Handling

• Never expose internal error details to users
• Mask sensitive information in logs
• Follow compliance and data protection guidelines

8. End-to-End Error Handling Flow

1. Validate input
2. Check token limits
3. Make API call
4. Handle API and network errors
5. Validate model output
6. Apply fallback if needed
7. Log and monitor errors