LLM Code Architecture Evaluation Framework

Automated Structural & Contextual Cohesion Benchmark

Executive Summary

This framework establishes a systematic approach to evaluate LLM code generation capabilities beyond functional correctness, focusing on architectural coherence and context window degradation through sequential, interdependent coding tasks.

Core Framework Components

1. Pattern-Based Structure Definition

Concept: Machine-readable architectural DNA that defines code organization and interaction patterns.

Key Elements: - **Architectural Layers**: Data access through Repository layer, Business logic in Service classes - **Design Patterns**: Enforced patterns (Observer, Strategy, Factory) with appropriate restrictions - **Component Interaction Rules**: No circular dependencies, specific inter-service communication protocols - **Naming Conventions**: Interface prefixes ('I'), Abstract class prefixes ('Abstract') - **Error Handling Policies**: Custom ApplicationError types handled at Service layer boundaries - **Dependency Management**: Standardized injection and management protocols - **Testing Philosophy**: Unit test requirements for all public methods

Implementation Formats: - Custom Domain-Specific Language (DSL) - YAML/JSON schema specifications - Semantic graph/tree representations - Optimized system prompt encoding

2. System Prompt Architecture

Purpose: Encoded architectural knowledge serving as the LLM's structural conscience.

Components: - Formalized blueprint in XML/JSON/Markdown - Direct adherence directives - Compliant/non-compliant code examples - Conflict resolution protocols - Justification requirements for deviations

3. Testing Methodology

Sequential Command Structure: Interdependent task sequences building complex codebases.

Example Sequence: 1. Generate UserRepository interface 2. Implement SqlUserRepository with Repository pattern 3. Create UserService with dependency injection 4. Add GetUserById method with repository updates 5. Refactor error handling to ApplicationError standard

Evaluation Dimensions: - Context stress testing - Model comparison across platforms - Deterministic output (temperature=0) - Reproducible degradation measurement

---

Quantifiable Metrics

1. Prompt Adherence (PA)

Similarity-Based Formula

```
PA = S(Code_actual, P) / S(Code_ideal, P)
```

Where: - `S(X,P)` = Similarity function measuring code structure X conformance to pattern P (0-1 scale) - `Code_actual` = LLM-generated code structure - `Code_ideal` = Perfect adherence benchmark code structure - `P` = Defined architectural pattern

Violation-Based Formula

```
PA = 1 - (V_actual) / (R × L)
```

Where: - `V_actual` = Number of observed pattern violations - `R` = Total applicable rules/constraints - `L` = Normalization factor (lines of code or architectural components)

Severity-Weighted Formula

```
PA = 1 - (∑(k=1 to V_actual) Severity_k) / Max_Possible_Severity_Score
```

Where: - `Severity_k` = Predefined weight for violation type k - Weight scale: 0.1 (minor), 0.5 (moderate), 1.0 (critical)

2. Context Window Degradation

Degradation Percentage (CWD%)

```
CWD% = (1 - PA_final / PA_initial) × 100%
```

Degradation Rate (DR)

```
DR = ΔPA / ΔContext = (PA_t1 - PA_t2) / (Context_Length_Change or Turn_Diff
```

Cohesion Loss Score

```
Cohesion_Loss = V_cohesion / N_total_checks
```

Where: - `V_cohesion` = Violations indicating logical inconsistency or instruction forgetting - `N_total_checks` = Total consistency checks performed

---

Implementation Requirements

Core Team Composition

- **LLM Engineers**: Prompt engineering and model interaction
- **Software Architects**: Pattern definition and adherence checker development
- **Data Scientists**: Metrics design and analysis
- **UX/UI Designers**: Benchmark platform interface (if public)

Technical Infrastructure

- **LLM API Access**: Multiple model platforms (Gemini, Claude, open-source)
- **Prompt Management System**: Sequential scenario design and execution
- **Automated Code Analysis Engine**: AST-based structural pattern validation
- **Results Database**: Output storage with metadata tracking

- **Analysis Platform**: Visualization and reporting tools
- **Version Control**: Framework and test case management

Test Case Library

- Sequential coding tasks ranging from simple to complex
- Architectural challenge scenarios
- Context memory stress tests
- Cross-language and framework support

Advanced Features

Diagnostic Capabilities

- **Interactive Degradation Debugging**: Automated explanation of PA drops
- **Rule-Specific Violation Tracking**: Granular adherence monitoring
- **Architectural Tool-Use Evaluation**: External specification consultation testing

Adaptive Management

- **Context Refresh Prediction**: Algorithmic determination of optimal context management
- **Compression Strategy Testing**: Context summarization effectiveness
- **Cost-Benefit Analysis**: Token cost vs. adherence maintenance

Community Integration

- **Benchmark Standardization**: Public submission platform
- **Multi-Language Support**: Cross-platform pattern validation
- **Economic Impact Analysis**: Development cost optimization

Research Landscape Analysis

Current State

Existing Research Areas: - LLM code generation from architectural specifications - Context window degradation studies (LongCodeBench) - Automated consistency analysis - "Needle-in-a-

Haystack" long-context evaluation

Novel Contributions

Unique Framework Elements: - Comprehensive pattern-based structure enforcement - Quantifiable architectural adherence metrics - Sequential interdependent command evaluation - Structural quality as primary performance indicator

Academic Positioning

Competitive Advantages: - Bridges conversational consistency and software engineering demands - Addresses professional development context degradation - Provides granular architectural understanding measurement - Offers real-world applicability beyond functional correctness

---

Strategic Recommendations

For Graduate School Applications

- Leverage UW CS degree credibility
- Demonstrate self-directed research initiative
- Target faculty with aligned research interests
- Develop robust proof-of-concept implementation

For Intellectual Property

- Prioritize research paper publication
- Consider open-source POC release
- Establish prior art through publication
- Consult patent attorney for cost-benefit analysis

For Career Development

- Build GitHub portfolio with implementation
- Network with relevant research communities
- Present findings at conferences/workshops
- Establish thought leadership in LLM evaluation

Success Metrics

Short-term Objectives

- Functional proof-of-concept development
- Initial pattern definition completion
- Basic adherence checker implementation
- Preliminary degradation measurement

Long-term Goals

- Comprehensive benchmark platform
- Multi-model evaluation dataset
- Community adoption and contribution
- Academic publication and recognition

Conclusion

This framework represents a significant advancement in LLM evaluation methodology, specifically addressing the gap between functional correctness and architectural coherence in code generation. By providing quantifiable metrics for structural adherence and context degradation, it enables more rigorous assessment of LLM capabilities for professional software development contexts.

The combination of pattern-based architecture definition, automated adherence measurement, and sequential task evaluation creates a comprehensive benchmark that addresses real-world development challenges while maintaining scientific rigor and reproducibility.