🚀 CQL Meta-Prompt Compiler - Proof of Concept Roadmap

Status: In Progress
Current Phase: Phase 1 - Foundation Infrastructure
Next Item: Phase 1.1 - MetaPromptCompiler Foundation Class
Created: September 1, 2025
Last Updated: September 1, 2025

📋 Implementation Plan Overview

Phase 1: Foundation Infrastructure (6 PRs)

Build the core interfaces and local compilation foundation

Phase 2: LLM Integration Core (7 PRs)

Add Anthropic API integration with reliability patterns

Phase 3: Complete Proof of Concept (4 PRs)

Full pipeline with CLI integration and demo

Total PRs: 17
Estimated Timeline: 5-6 weeks
Target Completion: Mid-October 2025

🏗️ Phase 1: Foundation Infrastructure

✅ PR #1: MetaPromptCompiler Foundation *(Status: ✅ COMPLETED)*

Commit Focus: Create base class structure and interfaces

- ☑ Create base `MetaPromptCompiler` namespace and interfaces
- ☑ Add forward declarations for all major components
- ☑ Establish header structure in `include/cql/meta_prompt/`
- ☑ **Deliverable**: Compilable interface definitions
- **Estimated Effort:** 1 day (Actual: < 1 day)
- **Files Created:**
  - `include/cql/meta_prompt/compiler.hpp` ✅
  - `include/cql/meta_prompt/types.hpp` ✅
  - `src/cql/test_meta_prompt_foundation.cpp` ✅ (bonus)
- **Commit:** 16bf0d0

✅ PR #2: Configuration System *(Status: Pending)*

Commit Focus: Add configuration enums and options structures

- ☐ Implement `CompilerFlags` struct with all optimization options
- ☐ Add `CompilationMode` enum (LOCAL_ONLY, CACHED_LLM, FULL_LLM)

- [ ] Create `OptimizationGoal` enum (REDUCE_TOKENS, IMPROVE_ACCURACY, BALANCED)
- [ ] **Deliverable**: Complete configuration system
- **Estimated Effort:** 1 day
- **Files to Create:**
  - `include/cql/meta_prompt/config.hpp`

✅ PR #3: Result Structures *(Status: Pending)*

Commit Focus: Define all result and metrics data structures

- [ ] Implement `CompilationResult` with success/failure, metrics, timing
- [ ] Add `ValidationResult` for semantic equivalence checking
- [ ] Create `CompilationMetrics` for performance tracking
- [ ] **Deliverable**: All result and metrics data structures
- **Estimated Effort:** 1 day
- **Files to Create:**
  - `include/cql/meta_prompt/results.hpp`

✅ PR #4: HybridCompiler Foundation *(Status: Pending)*

Commit Focus: Implement local-only compilation mode

- [ ] Implement `HybridCompiler::compile()` with LOCAL_ONLY mode
- [ ] Integrate with existing `QueryProcessor` as local backend
- [ ] Add basic error handling and result formatting
- [ ] **Deliverable**: Working local-only meta-compilation
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/hybrid_compiler.hpp`
  - `src/cql/meta_prompt/hybrid_compiler.cpp`

✅ PR #5: Basic Caching System *(Status: Pending)*

Commit Focus: Memory-based caching with semantic hashing

- [ ] Implement `IntelligentCache` with memory-based storage
- [ ] Add semantic hash computation for cache keys
- [ ] Create cache hit/miss metrics and basic eviction
- [ ] **Deliverable**: Working cache with 90%+ hit rate on repeated queries
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/intelligent_cache.hpp`
  - `src/cql/meta_prompt/intelligent_cache.cpp`

✅ PR #6: Foundation Tests *(Status: Pending)*

Commit Focus: Complete test coverage for Phase 1 components

- [ ] Create comprehensive unit tests for all Phase 1 components
- [ ] Add performance tests ensuring < 10ms local compilation
- [ ] Implement test fixtures and mocking infrastructure

- ☐ **Deliverable**: 100% test coverage for foundation
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `src/cql/test_meta_prompt_compiler.cpp`

🔌 Phase 2: LLM Integration Core

✅ PR #7: PromptCompiler Core *(Status: Pending)*

Commit Focus: LLM integration foundation

- ☐ Implement `PromptCompiler` class with API client integration
- ☐ Add meta-prompt template system with variable substitution
- ☐ Create basic optimization request/response handling
- ☐ **Deliverable**: Compilable LLM integration foundation
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/prompt_compiler.hpp`
  - `src/cql/meta_prompt/prompt_compiler.cpp`

✅ PR #8: TOKEN_OPTIMIZER Template *(Status: Pending)*

Commit Focus: First working meta-prompt template

- ☐ Implement first meta-prompt template for token reduction
- ☐ Add Anthropic API integration with proper authentication
- ☐ Create JSON response parsing and error handling
- ☐ **Deliverable**: Working token optimization via Claude API
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/templates.hpp`
  - `src/cql/meta_prompt/templates.cpp`

✅ PR #9: Circuit Breaker Pattern *(Status: Pending)*

Commit Focus: API reliability and failure handling

- ☐ Implement `CircuitBreaker` with CLOSED/OPEN/HALF_OPEN states
- ☐ Add exponential backoff retry logic with jitter
- ☐ Create failure threshold and recovery timeout configuration
- ☐ **Deliverable**: Robust API reliability with failure handling
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/circuit_breaker.hpp`
  - `src/cql/meta_prompt/circuit_breaker.cpp`

✅ PR #10: CACHED_LLM Mode *(Status: Pending)*

Commit Focus: High-performance cached compilation

- ☐ Integrate cache lookup before API calls
- ☐ Add intelligent cache storage after successful optimization
- ☐ Implement graceful fallback to local compilation on failures
- ☐ **Deliverable**: Sub-50ms cached compilation performance
- **Estimated Effort:** 2 days

✅ PR #11: Semantic Validation *(Status: Pending)*

Commit Focus: Optimization quality assurance

- ☐ Create `ValidationFramework` with AST comparison
- ☐ Add basic semantic equivalence checking
- ☐ Implement confidence scoring for optimization results
- ☐ **Deliverable**: Validation preventing semantic drift
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/validation_framework.hpp`
  - `src/cql/meta_prompt/validation_framework.cpp`

✅ PR #12: Cost Management *(Status: Pending)*

Commit Focus: Budget control and cost tracking

- ☐ Implement `CostController` with daily budget tracking
- ☐ Add per-compilation cost estimation and logging
- ☐ Create budget enforcement with graceful degradation
- ☐ **Deliverable**: Cost control within daily budgets
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `include/cql/meta_prompt/cost_controller.hpp`
  - `src/cql/meta_prompt/cost_controller.cpp`

✅ PR #13: Integration Tests *(Status: Pending)*

Commit Focus: Live API testing and verification

- ☐ Create live API integration tests (conditional on API key)
- ☐ Add end-to-end compilation pipeline testing
- ☐ Implement performance benchmarking suite
- ☐ **Deliverable**: Verified API integration with real Claude API
- **Estimated Effort:** 2 days
- **Files to Create:**
  - `src/cql/test_meta_prompt_live_integration.cpp`

🎯 Phase 3: Complete Proof of Concept

✅ PR #14: FULL_LLM Mode *(Status: Pending)*

Commit Focus: Complete meta-compilation pipeline

- [ ] Complete the full meta-compilation pipeline
- [ ] Add async compilation support for non-blocking workflows
- [ ] Implement advanced optimization strategies
- [ ] **Deliverable**: Complete LLM-powered compilation pipeline
- **Estimated Effort:** 2 days

✅ PR #15: CLI Integration *(Status: Pending)*

Commit Focus: User interface for meta-compilation

- [ ] Add `--optimize` flag to existing CQL CLI
- [ ] Implement `--mode`, `--goal`, `--domain` options
- [ ] Create optimization result display and metrics output
- [ ] **Deliverable**: Complete CLI experience for meta-compilation
- **Estimated Effort:** 2 days
- **Files to Modify:**
  - `src/cql/application_controller.cpp`
  - `src/cql/command_line_handler.cpp`

✅ PR #16: Proof of Concept Demo *(Status: Pending)*

Commit Focus: Compelling demonstration of capabilities

- [ ] Create comprehensive demo script showing capabilities
- [ ] Add before/after comparison with token reduction metrics
- [ ] Implement quality assessment and validation results
- [ ] **Deliverable**: Impressive demo showing real optimization benefits
- **Estimated Effort:** 1 day
- **Files to Create:**
  - `examples/meta_prompt_demo.cpp`
  - `examples/optimization_examples/`

✅ PR #17: Documentation & Benchmarks *(Status: Pending)*

Commit Focus: Complete proof of concept documentation

- [ ] Add performance benchmarking with detailed metrics
- [ ] Create comprehensive usage documentation
- [ ] Implement optimization results analysis and reporting
- [ ] **Deliverable**: Complete proof of concept documentation
- **Estimated Effort:** 1 day
- **Files to Create:**
  - `docs/META_PROMPT_COMPILER_USER_GUIDE.md`
  - `docs/META_PROMPT_COMPILER_BENCHMARKS.md`

🔍 Verification Criteria for Each PR

Each PR must meet these criteria before proceeding:

✅ Code Quality

- Builds without warnings
- Follows existing CQL coding standards
- Integrates with existing logger and error handling
- Uses modern C++20 features appropriately

✅ Testing Requirements

- Unit tests for all new functionality
- Integration tests where applicable
- Performance tests meeting specified targets
- No regression in existing CQL functionality

✅ Security Standards

- API keys handled via existing SecureString infrastructure
- Input validation using existing CQL patterns
- No sensitive data in logs or error messages
- Proper error context preservation

✅ Documentation

- Doxygen comments for all public APIs
- Code examples in header documentation
- Clear commit messages explaining the change
- Updated architecture diagrams if needed

🎯 Success Metrics for Proof of Concept

By the end of all 17 PRs, you'll have:

📊 Performance Metrics

- ✅ Local compilation: < 10ms
- ✅ Cached LLM compilation: < 50ms
- ✅ Full LLM compilation: < 500ms
- ✅ Cache hit rate: > 80%

💰 Cost Metrics

- ✅ Cost per optimization: $0.005-0.015
- ✅ Token reduction: 15-30% average
- ✅ Daily budget compliance: 100%

🛡️ Reliability Metrics

- ✅ Circuit breaker prevents cascade failures
- ✅ Graceful fallback to local compilation
- ✅ API failure handling: < 1% user impact

🚀 Feature Completeness

- ✅ Complete hybrid compilation pipeline
- ✅ CLI integration with optimization flags
- ✅ Real-time cost and performance monitoring
- ✅ Comprehensive test coverage (85%+)

📝 Progress Tracking

Completed PRs: 1/17

Current Phase: Phase 1 - Foundation Infrastructure

Next Action: Begin PR #2 - Configuration System

Phase 1 Progress: 1/6 PRs completed
Phase 2 Progress: 0/7 PRs completed
Phase 3 Progress: 0/4 PRs completed

Recent Updates:

- **Sept 2, 2025:** PR #1 completed - Foundation types and interfaces implemented ✅
- **Sept 1, 2025:** Initial roadmap created
- **Sept 1, 2025:** Feasibility analysis completed (9.5/10 score)

Blockers: None currently identified

Notes:

- Each PR should be a single focused feature
- All PRs must pass existing test suite
- API key required for Phase 2 integration tests
- Performance benchmarking throughout development

Next Step: Create PR #2 - Configuration System