Helix CLI Specification v3.0 - Complete Technical Blueprint

1. Project Foundation & Philosophy

1.1 Core Mission Statement

Helix CLI is an AI-powered coding agent designed specifically for terminal environments that enables developers to interact with Large Language Models (LLMs) for comprehensive software development workflows. The system bridges the gap between AI capabilities and traditional development practices through a unified, terminal-first interface.

1.2 Design Principles

- **Terminal-First Philosophy**: All features must work seamlessly in terminal environments
- **AI-Augmented Development**: LLMs enhance but don't replace developer decision-making
- **Progressive Enhancement**: Features scale from basic to advanced based on hardware capabilities
- **Zero Configuration Defaults**: Sensible defaults that work out of the box
- **Composable Architecture**: Independent components that can be used separately or together

1.3 Technical Foundation

- **Language**: Go (Golang) for all core components
- **Architecture**: Microservices with bash orchestration
- **Data Storage**: PostgreSQL with SQLCipher encryption
- **Configuration**: JSON-based with environment variable overrides
- **Cross-Platform**: Native support for Linux, macOS, BSDs, Windows

2. Core System Architecture

2.1 Component Architecture

2.1.1 LLM Integration Layer

Core LLM interface with provider implementations for Llama.cpp, Ollama, OpenRouter, DeepSeek, Qwen, Claude, Gemini, Grok, Mistral, HuggingFace, and NVIDIA.

2.1.2 User Interface Layer

Terminal UI, Headless CLI, and REST API server components with WebSocket support for real-time communication.

2.1.3 Processing Engine Layer

Planner, Builder, Refactorer, Tester, Debugger, Designer, Diagram, Deployment, and Porting engines with standardized interfaces.

2.2 Data Architecture

2.2.1 Database Schema

Comprehensive PostgreSQL schema with tables for projects, sessions, requests, models, users, permissions, logs, and configurations.

2.2.2 Configuration Management

Hierarchical configuration system with global, project, module, and session-level settings stored in JSON format.

3. LLM Integration & Management

3.1 Provider Implementation Details

3.1.1 Llama.cpp Integration

- Automatic source download and compilation
- Hardware-specific optimization flags
- Model format detection and conversion
- Context window management
- Streaming response handling
- Memory-mapped model loading
- GPU acceleration detection

3.1.2 Ollama Integration

- Local Ollama instance management
- Model pull and management
- Custom model creation
- Modelfile support
- Streaming generation
- System prompt templates

3.1.3 Remote API Providers

- Rate limiting and throttling
- Request/response logging
- Error handling and retries
- Cost tracking and budgeting
- Concurrent request management

3.2 Model Management System

3.2.1 Hardware Detection & Analysis

- GPU VRAM capacity and type detection
- System RAM availability analysis
- CPU cores and architecture identification
- Storage space and type assessment
- Network bandwidth estimation
- Thermal and power constraints monitoring

3.2.2 Model Selection Algorithm

Multi-stage selection process: hardware filtering, task suitability scoring, user preference application.

3.2.3 Model Installation Process

1. Source selection from multiple repositories
2. Format detection and validation
3. Conversion to required formats
4. Hardware-specific optimization
5. Integrity verification
6. Metadata registration

3.3 Advanced LLM Features

3.3.1 Tool Calling Implementation

File system operations, Git integration, build tools, testing frameworks, database operations, API calls, and shell command execution.

3.3.2 Thinking Process Support

Analysis, research, planning, validation, and optimization thinking steps with structured decision tracking.

4. User Interface Implementation

4.1 Terminal UI Architecture

4.1.1 UI Component Structure

Header, navigation, input, output, status, and help components with event-driven architecture.

4.1.2 Layout Manager

Fixed layout regions: header (top 3 lines), navigation (left 20%), main (center 60%), sidebar (right 20%), status (bottom 2 lines).

4.2 Input System

4.2.1 Command Processing Pipeline

Six-stage processing: input capture, syntax parsing, context resolution, permission checking, execution planning, result handling.

4.2.2 Command Categories

- Project commands: create, switch, list, info
- Session commands: start, join, list, kill
- Mode commands: plan, build, test, refactor, debug
- Model commands: list, install, switch, info
- Configuration commands: get, set, export, import

4.3 Theme System

4.3.1 Theme Definition

Color schemes, style definitions, font settings, and layout configurations in JSON format.

4.3.2 Built-in Themes

Default (green), Warm Red, Blue, Yellow, Gold, Grey, White, Darcula, Dark Blue, Violet, Warm Orange.

5. Development Workflows - Detailed Implementation

5.1 Planning Mode

5.1.1 Planning Process

Seven-step process: project analysis, technology research, architecture design, timeline estimation, resource calculation, risk assessment, documentation generation.

5.1.2 Project Analysis

Codebase structure, technology stack, performance characteristics, security vulnerabilities, code quality metrics, documentation coverage, test coverage analysis.

5.2 Building Mode

5.2.1 Parallel Building Architecture

Coordinator-worker architecture with module-based distribution, dependency-aware scheduling, resource limits, progress tracking, conflict resolution.

5.2.2 Code Generation Process

Requirements analysis, template selection, code generation with validation, formatting, integration testing, documentation generation.

5.3 Testing Mode

5.3.1 Comprehensive Testing Framework

Unit tests, integration tests, E2E tests, automation tests, performance tests, security tests with unified execution framework.

5.3.2 Quality Scanning Integration

SonarQube for code quality, Snyk for security vulnerabilities, custom scanners for performance, architecture, style, and dependency issues.

5.4 Refactoring Mode

5.4.1 Refactoring Process

Extract method/function, inline method/function, move method/class, rename variables/functions, change signature, extract interface, pull up/push down members, replace conditional with polymorphism.

5.4.2 Code Quality Improvements

Performance optimization, memory usage reduction, code complexity reduction, dependency cleanup, documentation enhancement.

5.5 Debugging Mode

5.5.1 Debugging Process

Issue reproduction, root cause analysis, solution generation, validation testing, documentation updates.

5.5.2 Debugging Tools

Log analysis, performance profiling, memory inspection, network monitoring, exception tracking.

5.6 Design Mode

5.6.1 Design System Integration

Figma API integration, Penpot API support, design token management, component library synchronization.

5.6.2 Design Generation

UI component creation, layout design, color scheme generation, typography selection, interaction design.

5.7 Diagram Mode

5.7.1 Diagram Generation

UML diagrams, architecture diagrams, flowcharts, sequence diagrams, entity-relationship diagrams.

5.7.2 Export Formats

Draw.io, Mermaid.js, PNG, JPEG, SVG, PDF with format conversion capabilities.

5.8 Deployment Mode

5.8.1 Deployment Profiles

Environment-specific configurations, cloud provider integrations, deployment scripts, rollback procedures.

5.8.2 Deployment Targets

AWS, Azure, GCP, Docker, Kubernetes, bare metal, virtual machines with provider-specific optimizations.

5.9 Porting Mode

5.9.1 Technology Mapping

Language-to-language conversion, framework migration, library replacement, API adaptation.

5.9.2 Porting Validation

Functional equivalence testing, performance benchmarking, compatibility verification, documentation updates.

6. Session & Collaboration System

6.1 Session Management

6.1.1 Session Lifecycle

Creation, activation, suspension, resumption, termination with state persistence and recovery.

6.1.2 Session Types

Single-user sessions, multi-user collaborative sessions, automated background sessions, detached sessions.

6.2 Multi-User Collaboration

6.2.1 Real-time Synchronization

WebSocket-based state synchronization, conflict resolution, change propagation, presence awareness.

6.2.2 Access Control

Role-based permissions, project-level access, session-level restrictions, operation-level authorization.

6.3 Distributed Computing

6.3.1 Node Management

Worker node registration, capability discovery, load balancing, health monitoring, automatic failover.

6.3.2 Work Distribution

Task partitioning, dependency management, result aggregation, progress synchronization, error handling.

7. Memory & State Management

7.1 Memory Architecture

7.1.1 Multi-layer Storage

Project memory, session memory, request memory, user memory with hierarchical organization.

7.1.2 Context Management

Conversation context, project context, user context, temporal context with intelligent pruning.

7.2 Intelligent Memory Features

7.2.1 Pattern Recognition

Duplicate request detection, similar task identification, context-aware suggestion generation.

7.2.2 Learning System

Progressive improvement from history, user preference adaptation, performance optimization.

8. Security & Access Control

8.1 Security Architecture

8.1.1 Data Protection

End-to-end encryption, secure credential storage, encrypted database, secure communication channels.

8.1.2 Access Security

Multi-factor authentication, session management, API key rotation, audit logging.

8.2 Account Management

8.2.1 User Management

Account creation, profile management, preference storage, activity tracking.

8.2.2 Permission System

Granular permissions, role definitions, inheritance rules, temporary access grants.

9. Performance & Optimization

9.1 Performance Monitoring

9.1.1 Resource Tracking

CPU usage, memory consumption, disk I/O, network bandwidth, GPU utilization.

9.1.2 Performance Metrics

Response times, throughput rates, error rates, resource efficiency, user satisfaction.

9.2 Optimization Strategies

9.2.1 Caching System

Response caching, model caching, template caching, configuration caching with intelligent invalidation.

9.2.2 Resource Management

Dynamic resource allocation, load balancing, connection pooling, garbage collection optimization.

10. Testing & Quality Assurance

10.1 Testing Strategy

10.1.1 Test Coverage

100% code coverage requirement, multiple test types, automated test generation, continuous testing.

10.1.2 Quality Gates

SonarQube quality gates, Snyk security gates, performance thresholds, compatibility requirements.

10.2 Automated Testing

10.2.1 Test Generation

Unit test generation, integration test creation, E2E test automation, performance test setup.

10.2.2 Test Execution

Parallel test execution, distributed testing, continuous integration, result analysis.

11. Documentation System

11.1 Documentation Generation

11.1.1 Automatic Documentation

API documentation, user guides, developer documentation, architecture documentation.

11.1.2 Documentation Formats

Markdown, HTML, PDF, interactive documentation with search and navigation.

11.2 Documentation Management

11.2.1 Version Control

Documentation versioning, change tracking, update propagation, archive management.

11.2.2 Multi-language Support

Internationalization, localization, language detection, automatic translation.

12. Integration & Extensibility

12.1 External Integrations

12.1.1 Version Control

Git integration, branch management, commit automation, merge conflict resolution.

12.1.2 CI/CD Systems

Jenkins, GitHub Actions, GitLab CI, CircleCI, Travis CI integration with pipeline automation.

12.2 Plugin System

12.2.1 Plugin Architecture

Modular plugin system, hot reloading, dependency management, version compatibility.

12.2.2 Plugin Types

Provider plugins, tool plugins, UI plugins, integration plugins with standardized interfaces.

13. Deployment & Distribution

13.1 Packaging & Distribution

13.1.1 Multi-platform Packaging

Native packages for Linux (deb, rpm), macOS (dmg, pkg), Windows (msi, exe), BSD (pkg).

13.1.2 Containerization

Docker images, Kubernetes manifests, Helm charts, container orchestration.

13.2 Installation & Setup

13.2.1 Automated Installation

One-line installation scripts, dependency resolution, configuration wizard, first-time setup.

13.2.2 Update Management

Automatic updates, version checking, migration scripts, rollback capabilities.

14. Monitoring & Analytics

14.1 System Monitoring

14.1.1 Health Monitoring

Service health checks, resource monitoring, performance alerts, automatic recovery.

14.1.2 Usage Analytics

Feature usage tracking, performance metrics, user behavior analysis, improvement suggestions.

14.2 Reporting System

14.2.1 Automated Reports

Daily/weekly/monthly reports, project progress reports, quality metrics, performance analysis.

14.2.2 Custom Reports

Ad-hoc reporting, data export, visualization tools, dashboard creation.

15. Implementation Roadmap

15.1 Phase 1: Core Foundation (Months 1-3)

- Basic LLM integration (Llama.cpp, Ollama)
- Terminal UI framework
- Project configuration system
- Basic building and testing modes
- Simple session management

15.2 Phase 2: Advanced Features (Months 4-6)

- Multi-provider support
- Advanced session management
- Comprehensive testing integration
- Refactoring and debugging modes
- Basic collaboration features

15.3 Phase 3: Collaboration & Scale (Months 7-9)

- Multi-user sessions
- Distributed computing
- Design and diagram modes
- Deployment capabilities
- Advanced security features

15.4 Phase 4: Enterprise Features (Months 10-12)

- Account management
- Advanced analytics
- Performance optimization
- Comprehensive documentation
- Plugin system

16. Success Metrics & KPIs

16.1 Performance Indicators

- **Response Time**: Sub-second command execution (target: <500ms)
- **Resource Efficiency**: Optimal hardware utilization (target: >85%)
- **Test Coverage**: 100% automated test implementation
- **User Satisfaction**: High adoption and retention rates (target: >90%)

16.2 Quality Standards

- **Code Quality**: SonarQube passing scores (target: A rating)
- **Security**: Snyk vulnerability-free status
- **Reliability**: 99.9% uptime for core features
- **Documentation**: Comprehensive and up-to-date (target: 100% coverage)

This specification provides exhaustive technical details for implementing Helix CLI, covering every aspect from low-level implementation to high-level architecture. The modular design allows for incremental development while maintaining a clear path to the complete vision.