Complete DeepWeb Development Roadmap - Claude Code Prompts

Overview

This roadmap provides a comprehensive set of prompts for Claude Code to transform the DeepWeb Firefox extension from its current state into a production-ready, market-leading AI assistant. Each prompt is designed to be complete, actionable, and includes all necessary context.

Execution Strategy

Phase Execution Order

1. **Phase 1**: Code Refactoring & Architecture (Weeks 1-2)
2. **Phase 2**: Core Feature Implementation (Weeks 3-4)
3. **Phase 3**: Advanced UI/UX & Customization (Weeks 5-6)
4. **Phase 4**: Intelligence & Context Features (Weeks 7-8)
5. **Phase 5**: Performance & Security (Weeks 9-10)
6. **Phase 6**: Production Release (Weeks 11-12)

Context Management

- Provide current project files with each prompt
- Reference previous phase outputs
- Include specific integration requirements
- Validate each phase before proceeding

PHASE 1: CODE REFACTORING & ARCHITECTURE

1.1 Content Script Modularization

TASK: Refactor the content script architecture to eliminate inline HTML and improve maintainability

CONTEXT:
- Current content-firefox.js contains 200+ lines of inline HTML
- Inline event handlers present security risks
- No modular structure for UI components
- HTML/CSS/JS are mixed in a single function

REQUIREMENTS:
1. **Template System Creation**:
   - Create a proper HTML template system
   - Extract all HTML from JavaScript strings
   - Implement template-based component rendering
   - Add template validation and error handling

2. **Component Architecture**:
   - Create modular UI components (Header, MessageList, InputArea, etc.)
   - Implement component lifecycle management
   - Add proper event delegation instead of inline handlers
   - Create component communication system

3. **Security Enhancements**:
   - Replace innerHTML usage with safe DOM manipulation
   - Remove all inline event handlers
   - Implement proper input sanitization
   - Add CSP-compliant event handling

4. **File Structure**:

```
content/
├── content-firefox.js (main coordinator)
├── templates/
│   ├── chat-container.html
│   ├── header.html
│   ├── message-list.html
│   └── input-area.html
├── components/
│   ├── ChatContainer.js
│   ├── Header.js
│   ├── MessageList.js
│   └── InputArea.js
└── utils/
├── template-loader.js
├── dom-utils.js
└── event-manager.js
```

```
5. **Implementation Details**:
- Use DocumentFragment for efficient DOM manipulation
- Implement proper component state management
- Add comprehensive error boundaries
- Include performance optimizations (lazy loading, virtual scrolling)

SUCCESS CRITERIA:
- Zero inline HTML strings in JavaScript
- All event handlers properly delegated
- Components are reusable and testable
- 100% test coverage for new components
- No security vulnerabilities from innerHTML usage
- Performance improvement of 30% in UI creation

DELIVERABLES:
- Refactored content script with modular architecture
- Comprehensive test suite for all components
- Updated build process to handle templates
- Security audit report showing improvements
- Performance benchmarks showing improvements
```

1.2 Configuration System Unification

TASK: Create a unified configuration management system to eliminate duplication

CONTEXT:
- Configuration is duplicated between background-firefox.js and config.js
- No single source of truth for application settings
- Inconsistent validation across different modules
- Hard to maintain and update configurations

REQUIREMENTS:
1. **Unified Configuration Architecture**:
   - Create a single configuration source
   - Implement environment-specific configurations
   - Add runtime configuration validation
   - Create configuration versioning for migrations

2. **Configuration Schema**:
   ```javascript
   {
     api: {
       providers: {
         deepseek: { endpoint, models, pricing },
         openai: { endpoint, models, pricing },
         anthropic: { endpoint, models, pricing }
       },
       timeout: 30000,
       retries: 3,
       rateLimit: { interval: 10000, maxPerHour: 100 }
     },
     ui: {
       themes: { light: {...}, dark: {...} },
       layouts: { corner: {...}, sidebar: {...} },
       animations: { enabled: true, duration: 200 }
     },
     security: {
       apiKeyValidation: { minLength: 20, maxLength: 200 },
       contentSecurity: { maxLength: 1000, allowedTags: [] },
       rateLimiting: { enabled: true, strictMode: false }
     },
     features: {
       streaming: { enabled: true, chunkSize: 1024 },
       conversations: { maxHistory: 100, autoSave: true },
       export: { formats: ['json', 'markdown', 'html'] }
     }
   }
```

3. **Configuration Manager**:
    - Implement singleton pattern for configuration access
    - Add real-time configuration updates
    - Create configuration validation with detailed error messages
    - Implement configuration migration system
4. **Integration Points**:
    - Update background script to use unified config
    - Update content script to use unified config
    - Update popup script to use unified config
    - Add webpack plugin to validate configurations

SUCCESS CRITERIA:

- Single source of truth for all configurations
- Zero configuration duplication across files
- All configurations validated at runtime
- Migration system for configuration updates
- 100% test coverage for configuration system

DELIVERABLES:

- Unified configuration system with proper architecture
- Configuration validation and migration tools
- Updated all modules to use new configuration
- Comprehensive test suite for configuration management
- Documentation for configuration options

```
### 1.3 Error Handling Standardization
```

TASK: Implement comprehensive error handling throughout the application

CONTEXT:

- Inconsistent error handling patterns across modules
- Poor user experience with generic error messages
- No centralized error logging or reporting
- Missing error recovery mechanisms

REQUIREMENTS:

1. **Error Handling Architecture**:
   - Create standardized error classes with proper inheritance
   - Implement error boundaries for UI components
   - Add centralized error logging and reporting
   - Create error recovery strategies

2. **Error Classification System**:

```javascript
class DeepWebError extends Error {
  constructor(message, code, severity, recoverable) {
    super(message);
    this.code = code;
    this.severity = severity; // 'low', 'medium', 'high', 'critical'
    this.recoverable = recoverable;
    this.timestamp = new Date().toISOString();
  }
}

// Specific error types:
// - ApiError (rate limit, authentication, network)
// - ValidationError (input validation, configuration)
// - UIError (component failures, rendering issues)
// - SecurityError (XSS attempts, invalid content)
```

3. **User-Friendly Error Messages**:
   - Create error message templates with actions
   - Implement contextual help for common errors
   - Add error recovery suggestions
   - Create multi-language error message support

4. **Error Reporting & Analytics**:
   - Implement privacy-preserving error reporting
   - Add error trend analysis
   - Create error dashboard for monitoring
   - Implement automatic error recovery where possible

5. **Error Boundaries**:
   - Wrap all UI components with error boundaries
   - Implement graceful degradation for non-critical errors
   - Add retry mechanisms with exponential backoff
   - Create fallback UI for critical component failures

- Create fallback UI for critical component failures

SUCCESS CRITERIA:

- All errors properly classified and handled
- User-friendly error messages with recovery actions
- Centralized error logging and reporting
- 90% reduction in unhandled errors
- Error recovery rate of 80% for recoverable errors

DELIVERABLES:

- Comprehensive error handling system
- User-friendly error UI components
- Error reporting and analytics system
- Updated all modules with proper error handling
- Error handling documentation and examples

```
### 1.4 API Layer Refactoring
```

TASK: Refactor the API layer for better maintainability and extensibility

CONTEXT:

- makeAPIRequest function is too complex and handles too many responsibilities
- No proper retry logic or error handling
- Hard to add new API providers
- No request/response interceptors

REQUIREMENTS:

1. **API Client Architecture**:
   - Create abstract API client with provider-specific implementations
   - Implement request/response interceptors
   - Add comprehensive retry logic with exponential backoff
   - Create request queuing and rate limiting
2. **Provider Abstraction**:

```javascript
class APIProvider {
  constructor(config) { this.config = config; }

  async chat(messages, options) { throw new Error('Not implemented'); }
  async stream(messages, options) { throw new Error('Not implemented'); }
  validateApiKey(key) { throw new Error('Not implemented'); }
  calculateCost(usage) { throw new Error('Not implemented'); }
}

class DeepSeekProvider extends APIProvider {
  // Specific implementation
}
```

3. **Request Management**:
   - Implement request caching for identical queries
   - Add request deduplication
   - Create request priority system
   - Implement request cancellation
4. **Response Handling**:
   - Add response validation and sanitization
   - Implement streaming response handling
   - Create response transformation pipeline
   - Add response caching strategies
5. **Error Handling & Retry Logic**:
   - Implement exponential backoff with jitter
   - Add circuit breaker pattern for failing endpoints
   - Create retry policies per error type
   - Implement fallback provider switching

SUCCESS CRITERIA:

- Clean separation between API providers
- Robust retry and error handling
- 50% reduction in API request failures
- Support for multiple API providers
- Comprehensive request/response logging

DELIVERABLES:

- Refactored API layer with provider abstraction
- Comprehensive retry and error handling
- Request/response interceptor system
- Updated background script to use new API layer
- API layer documentation and examples

```
---

## PHASE 2: CORE FEATURE IMPLEMENTATION

### 2.1 Conversation Management System
```

TASK: Implement comprehensive conversation management with IndexedDB persistence

CONTEXT:

- No conversation persistence across browser sessions
- No way to manage multiple conversations
- No conversation history or search capabilities
- Users lose all context when browser is closed

REQUIREMENTS:

1. **IndexedDB Database Design**:

```javascript
// Database Schema
{
  conversations: {
    id: string (UUID),
    title: string,
    createdAt: Date,
    updatedAt: Date,
    model: string,
    metadata: {
      messageCount: number,
      totalCost: number,
      url: string,
      pageTitle: string
    }
  },
  messages: {
    id: string (UUID),
    conversationId: string,
    role: 'user' | 'assistant' | 'system',
    content: string,
    timestamp: Date,
    metadata: {
      model: string,
      cost: number,
      tokens: number
    }
  },
  settings: {
    key: string,
    value: any,
    updatedAt: Date
  }
}
```

2. **Conversation Management Features**:
   - Create new conversations
   - Switch between conversations
   - Rename conversations
   - Delete conversations
   - Archive/unarchive conversations
   - Duplicate conversations

3. **Message Management**:
    - Add messages to conversations
    - Edit messages (with re-generation)
    - Delete messages
    - Search within conversations
    - Export conversation history
4. **Storage Management**:
    - Implement storage quotas and cleanup
    - Add data migration system
    - Create backup/restore functionality
    - Implement data compression for large conversations
5. **UI Integration**:
    - Add conversation list sidebar
    - Implement conversation switching
    - Add conversation management controls
    - Create conversation search interface

SUCCESS CRITERIA:

- Conversations persist across browser sessions
- Users can manage multiple conversations
- Full-text search within conversations
- Export/import functionality working
- Storage management prevents quota issues

DELIVERABLES:

- Complete IndexedDB implementation
- Conversation management UI components
- Data migration and backup systems
- Comprehensive test suite for data persistence
- User documentation for conversation features

```
### 2.2 Enhanced Message System
```

TASK: Implement advanced message features including editing, deletion, and search

CONTEXT:

- Currently only supports basic message display
- No way to edit or delete messages
- No search functionality within conversations
- Missing message metadata and formatting

REQUIREMENTS:

1. **Message CRUD Operations**:
   - Edit message content with re-generation
   - Delete messages with confirmation
   - Copy messages to clipboard
   - Quote/reply to specific messages
   - Message reactions and ratings

2. **Message Formatting & Display**:
   - Implement markdown rendering with syntax highlighting
   - Add code block formatting with copy buttons
   - Support for tables, lists, and links
   - Add message timestamps and metadata display
   - Implement message threading for complex discussions

3. **Search & Filter System**:

```javascript
// Search capabilities
{
  fullTextSearch: true,
  filters: {
    dateRange: { start: Date, end: Date },
    messageType: ['user', 'assistant', 'system'],
    model: ['deepseek-chat', 'deepseek-reasoner'],
    hasCode: boolean,
    hasLinks: boolean
  },
  sorting: ['relevance', 'date', 'conversation'],
  pagination: { limit: 50, offset: 0 }
}
```

4. **Message Enhancement Features**:
   - Add message bookmarking
   - Implement message tagging system
   - Create message templates and shortcuts
   - Add message export in multiple formats
   - Implement message sharing capabilities

5. **Performance Optimizations**:
   - Virtual scrolling for large conversations
   - Lazy loading of message content

- Efficient search indexing
- Message content compression

SUCCESS CRITERIA:

- Full CRUD operations on messages
- Rich markdown rendering with syntax highlighting
- Fast full-text search across all messages
- Message templates and shortcuts working
- Virtual scrolling handles 1000+ messages smoothly

DELIVERABLES:

- Enhanced message system with full CRUD
- Advanced search and filtering capabilities
- Rich text rendering with markdown support
- Message templates and shortcuts system
- Performance-optimized message display

```
### 2.3 Streaming Response Implementation
```

TASK: Implement real-time streaming responses for better user experience

CONTEXT:

- Current implementation waits for complete responses
- No progress indication during API calls
- Poor user experience for long responses
- Configuration suggests streaming exists but it's not implemented

REQUIREMENTS:

1. **Streaming Architecture**:
   - Implement Server-Sent Events (SSE) handling
   - Add response chunking and progressive display
   - Create cancellation mechanisms for long responses
   - Implement connection management and reconnection

2. **UI Streaming Components**:
   - Add typing indicators during response generation
   - Implement progressive text display
   - Create response progress indicators
   - Add streaming cancellation controls

3. **API Provider Streaming**:

```javascript
class StreamingProvider {
  async *streamChat(messages, options) {
    const response = await fetch(endpoint, {
      method: 'POST',
      headers: { ...headers },
      body: JSON.stringify({ ...payload, stream: true })
    });

    const reader = response.body.getReader();
    let buffer = '';

    while (true) {
      const { done, value } = await reader.read();
      if (done) break;

      buffer += new TextDecoder().decode(value);
      const lines = buffer.split('\n');
      buffer = lines.pop() || '';

      for (const line of lines) {
        if (line.startsWith('data: ')) {
          const data = JSON.parse(line.slice(6));
          yield data;
        }
      }
    }
  }
}
```

4. **Response Management**:
    - Handle partial responses and reassembly
    - Implement response buffering for reliability
    - Add response validation during streaming
    - Create response recovery mechanisms
5. **Performance Optimizations**:
    - Implement efficient DOM updates during streaming
    - Add debouncing for rapid updates
    - Create memory management for long streams
    - Optimize rendering performance

SUCCESS CRITERIA:

- Real-time response streaming working
- Smooth progressive text display
- Reliable connection management
- User can cancel long responses
- 90% reduction in perceived response time

DELIVERABLES:

- Complete streaming implementation
- Streaming UI components with progress indicators
- Connection management and error recovery
- Performance-optimized streaming display
- Comprehensive streaming tests

```
### 2.4 Export/Import System
```

TASK: Implement comprehensive data export and import capabilities

CONTEXT:

- No way to export conversation history
- No backup mechanism for user data
- Can't share conversations with others
- No migration path between installations

REQUIREMENTS:

1. **Export Formats**:
   - JSON (structured data with metadata)
   - Markdown (human-readable format)
   - HTML (styled format for sharing)
   - CSV (for data analysis)
   - PDF (for printing/archiving)

2. **Export Options**:

```javascript
{
  scope: 'conversation' | 'all' | 'selection',
  format: 'json' | 'markdown' | 'html' | 'csv' | 'pdf',
  options: {
    includeMetadata: boolean,
    includeTimestamps: boolean,
    includeSystemMessages: boolean,
    anonymize: boolean,
    compress: boolean
  },
  dateRange: { start: Date, end: Date },
  conversations: string[] // conversation IDs
}
```

3. **Import System**:
   - Support importing JSON exports
   - Handle data validation and sanitization
   - Implement conflict resolution for duplicate data
   - Add progress tracking for large imports
   - Create data migration from other formats

4. **Sharing Features**:
   - Generate shareable conversation links
   - Create conversation snapshots
   - Implement privacy controls for sharing
   - Add conversation embedding for websites

5. **Backup/Restore**:
   - Automatic backup scheduling
   - Cloud storage integration (user's choice)
   - Incremental backup system

- One-click restore functionality

SUCCESS CRITERIA:

- All export formats generate properly formatted output
- Import system handles various data sources
- Backup/restore works reliably
- Sharing features maintain privacy
- Large datasets export/import efficiently

DELIVERABLES:

- Complete export system with multiple formats
- Robust import with validation and conflict resolution
- Automatic backup and restore functionality
- Conversation sharing with privacy controls
- Data migration tools for different formats

PHASE 3: ADVANCED UI/UX & CUSTOMIZATION

3.1 Resizable and Repositionable Interface

TASK: Create a flexible, customizable UI that users can resize and reposition

CONTEXT:

- Current UI is fixed size and position
- No customization options for different screen sizes
- One-size-fits-all approach doesn't work for all users
- Need to support various browsing patterns

REQUIREMENTS:

1. **Resizable Interface**:
   - Implement drag-to-resize functionality
   - Add resize constraints (min/max dimensions)
   - Create responsive breakpoints for different sizes
   - Save user preferences for sizing

2. **Repositioning System**:

```javascript
// Position options
{
  corner: {
    position: 'bottom-right' | 'bottom-left' | 'top-right' | 'top-left',
    offset: { x: number, y: number }
  },
  sidebar: {
    side: 'left' | 'right',
    width: number,
    height: '100%' | number
  },
  floating: {
    x: number,
    y: number,
    draggable: boolean
  },
  fullscreen: {
    overlay: boolean,
    modal: boolean
  }
}
```

3. **Layout Management**:
   - Create layout presets for common use cases
   - Implement layout switching with animations
   - Add layout persistence across sessions
   - Create responsive layouts for mobile devices

4. **Advanced UI Controls**:
   - Add docking/undocking functionality
   - Implement window snapping to screen edges
   - Create transparency/opacity controls
   - Add always-on-top functionality

5. **Accessibility Features**:
    - Keyboard navigation for all resize/move operations
    - Screen reader support for position changes
    - High contrast mode compatibility
    - Reduced motion support

SUCCESS CRITERIA:

- Users can resize interface to any reasonable size
- All position modes work correctly
- UI preferences persist across sessions
- Responsive design works on all screen sizes
- Accessibility features work properly

DELIVERABLES:

- Resizable interface with proper constraints
- Multiple positioning modes (corner, sidebar, floating)
- Layout management system with presets
- Responsive design for mobile devices
- Accessibility-compliant resize/move controls

```
### 3.2 Advanced Theme System
```

TASK: Implement a comprehensive theming system with customization options

CONTEXT:

- Currently only supports basic light/dark themes
- No customization options for colors, fonts, or spacing
- Themes don't adapt to user preferences or website colors
- Missing support for accessibility requirements

REQUIREMENTS:

1. **Theme Architecture**:

```javascript
// Theme structure
{
  name: string,
  version: string,
  colors: {
    primary: string,
    secondary: string,
    background: string,
    surface: string,
    text: {
      primary: string,
      secondary: string,
      disabled: string
    },
    accent: string,
    error: string,
    warning: string,
    success: string
  },
  typography: {
    fontFamily: string,
    fontSize: {
      small: string,
      medium: string,
      large: string
    },
    fontWeight: {
      normal: number,
      medium: number,
      bold: number
    },
    lineHeight: number
  },
  spacing: {
    small: string,
    medium: string,
    large: string
  },
  borderRadius: string,
  shadows: {
    small: string,
    medium: string,
    large: string
```

```
  }
}
```

2. **Built-in Themes**:
   - Light theme (default)
   - Dark theme
   - High contrast theme
   - Sepia theme (reading mode)
   - Auto theme (matches system preference)
   - Website-adaptive theme (extracts colors from current page)
3. **Custom Theme Creation**:
   - Visual theme editor with live preview
   - Color picker with accessibility validation
   - Typography customization
   - Import/export custom themes
   - Theme sharing community features
4. **Adaptive Features**:
   - Automatic theme switching based on time of day
   - Website color extraction and adaptation
   - Accessibility preference detection
   - System theme preference integration
5. **Performance Optimization**:
   - CSS custom properties for theme switching
   - Lazy loading of theme assets
   - Theme caching and preloading
   - Smooth theme transitions

SUCCESS CRITERIA:

- Multiple built-in themes work perfectly
- Custom theme creation is intuitive
- Theme switching is smooth and fast
- Accessibility themes meet WCAG guidelines
- Website-adaptive themes work on major sites

DELIVERABLES:

- Comprehensive theme system with multiple options
- Visual theme editor with live preview
- Website-adaptive theming capability
- Accessibility-compliant theme options
- Theme import/export and sharing features

3.3 Animation and Interaction System

TASK: Implement smooth animations and interactions to enhance user experience

CONTEXT:

- Current interface lacks smooth transitions
- No feedback for user interactions
- Static interface feels unresponsive
- Missing micro-interactions that improve usability

REQUIREMENTS:

1. **Animation Framework**:
   - Create CSS-based animation system
   - Implement JavaScript animation fallbacks
   - Add animation performance monitoring
   - Create animation presets for common patterns

2. **UI Animations**:

```javascript
// Animation types
{
  transitions: {
    slideIn: { duration: 300, easing: 'ease-out' },
    fadeIn: { duration: 200, easing: 'ease-in' },
    scaleIn: { duration: 250, easing: 'ease-in-out' },
    bounceIn: { duration: 400, easing: 'cubic-bezier(0.68, -0.55, 0.265, 1.55)' }
  },
  interactions: {
    hover: { transform: 'scale(1.05)', duration: 150 },
    active: { transform: 'scale(0.95)', duration: 100 },
    focus: { boxShadow: '0 0 0 3px rgba(66, 153, 225, 0.5)' }
  },
  feedback: {
    success: { color: 'green', icon: 'checkmark', duration: 2000 },
    error: { color: 'red', icon: 'warning', duration: 3000 },
    loading: { spinner: true, opacity: 0.7 }
  }
}
```

3. **Micro-interactions**:
   - Button hover and click animations
   - Loading states and progress indicators
   - Message appearance animations
   - Typing indicators and response animations
   - Gesture feedback for touch devices

4. **Performance Considerations**:
   - Use CSS transforms for better performance
   - Implement animation frame throttling
   - Add reduced motion support
   - Optimize animations for mobile devices

5. **User Controls**:
    - Animation speed controls (slow, normal, fast)
    - Disable animations option
    - Custom animation preferences
    - Animation preview system

SUCCESS CRITERIA:

- Smooth 60fps animations throughout the interface
- Reduced motion preferences respected
- Animations enhance rather than distract from usability
- Performance impact is minimal
- User can customize animation preferences

DELIVERABLES:

- Comprehensive animation system
- Smooth transitions for all UI elements
- Micro-interactions that provide feedback
- Performance-optimized animations
- User controls for animation preferences

```
### 3.4 Mobile Responsive Design
```

TASK: Create a fully responsive design that works perfectly on mobile devices

CONTEXT:

- Current design is desktop-focused
- Poor usability on mobile and tablet devices
- No touch-specific interactions
- Missing mobile-specific features

REQUIREMENTS:

1. **Responsive Breakpoints**:

```css
/* Breakpoint system */
@media (max-width: 480px) { /* Mobile portrait */ }
@media (max-width: 768px) { /* Mobile landscape / small tablet */ }
@media (max-width: 1024px) { /* Tablet */ }
@media (min-width: 1025px) { /* Desktop */ }
```

2. **Mobile Layout Adaptations**:
   - Collapsible header with hamburger menu
   - Full-screen conversation view on mobile
   - Swipe gestures for navigation
   - Touch-friendly button sizes (44px minimum)
   - Optimized input methods for mobile keyboards
3. **Touch Interactions**:
   - Swipe to dismiss messages
   - Pull-to-refresh conversations
   - Pinch-to-zoom for text scaling
   - Long-press context menus
   - Touch-specific hover states
4. **Mobile-Specific Features**:
   - Voice input integration
   - Share to other apps
   - Picture-in-picture mode
   - Notification system integration
   - Background sync for conversations
5. **Performance Optimizations**:
   - Lazy loading for mobile networks
   - Reduced bundle size for mobile
   - Touch event optimization
   - Memory management for mobile devices

SUCCESS CRITERIA:

- Perfect usability on all mobile devices
- Touch interactions feel native
- Performance is excellent on mobile networks
- Battery usage is minimized
- Accessibility works on mobile screen readers

DELIVERABLES:

- Fully responsive design for all screen sizes
- Touch-optimized interactions and gestures
- Mobile-specific feature implementations
- Performance optimizations for mobile devices
- Comprehensive mobile testing suite

```
---

## PHASE 4: INTELLIGENCE & CONTEXT FEATURES

### 4.1 Smart Context Extraction
```

TASK: Implement intelligent webpage content analysis and context extraction

CONTEXT:

- Current context extraction is basic (first 500 characters)
- No intelligent content prioritization
- Missing semantic understanding of webpage structure
- No content type detection or optimization

REQUIREMENTS:

1. **Intelligent Content Analysis**:

```javascript
// Content analysis pipeline
{
  extraction: {
    text: extractCleanText(document),
    structure: analyzePageStructure(document),
    metadata: extractMetadata(document),
    media: analyzeMediaContent(document)
  },
  analysis: {
    contentType: detectContentType(content), // article, product, code, etc.
    mainContent: extractMainContent(content),
    keyPoints: extractKeyPoints(content),
    entities: extractNamedEntities(content)
  },
  optimization: {
    relevanceScore: calculateRelevance(content, query),
    contextWindow: optimizeContextWindow(content, limit),
    summaryGeneration: generateSummary(content)
  }
}
```

2. **Content Type Detection**:
   - Article/blog post detection
   - Product page analysis
   - Code repository analysis
   - Social media content
   - News article processing
   - Documentation pages
   - E-commerce pages

3. **Semantic Content Extraction**:
   - Main content identification (removing nav, ads, etc.)
   - Key information extraction (headings, lists, tables)
   - Named entity recognition (people, places, organizations)
   - Sentiment analysis of content
   - Topic classification

4. **Context Optimization**:
   - Intelligent content summarization

- Relevance-based content filtering
- Context window optimization for different models
- Multi-modal content handling (text + images)

5. **Performance Considerations**:
   - Efficient DOM traversal algorithms
   - Background processing for large pages
   - Caching of extracted content
   - Incremental processing for dynamic content

SUCCESS CRITERIA:

- Accurate content type detection (90%+ accuracy)
- Relevant context extraction improves AI responses
- Processing time under 100ms for typical pages
- Main content extraction works on major websites
- Context quality scores consistently high

DELIVERABLES:

- Intelligent content extraction system
- Content type detection and optimization
- Semantic analysis and entity recognition
- Context quality scoring and optimization
- Performance-optimized content processing

```
### 4.2 Prompt Templates and Shortcuts
```

TASK: Create a comprehensive prompt template and shortcuts system

CONTEXT:

- Users often ask similar types of questions
- No way to save frequently used prompts
- Missing quick actions for common tasks
- No prompt optimization for different content types

REQUIREMENTS:

1. **Template System Architecture**:

```javascript
// Template structure
{
  id: string,
  name: string,
  description: string,
  category: string,
  template: string, // with variables like {content}, {url}, {selection}
  variables: {
    content: { required: true, type: 'text', source: 'page' },
    selection: { required: false, type: 'text', source: 'user' },
    url: { required: true, type: 'url', source: 'page' }
  },
  model: string, // preferred model for this template
  settings: {
    temperature: number,
    maxTokens: number
  }
}
```

2. **Built-in Template Categories**:
   - **Summarization**: "Summarize this page", "Key points", "TL;DR"
   - **Analysis**: "Explain this", "Analyze pros/cons", "Find issues"
   - **Coding**: "Review this code", "Explain function", "Find bugs"
   - **Research**: "Find similar content", "Fact-check", "Sources"
   - **Writing**: "Improve text", "Fix grammar", "Rewrite tone"
   - **Learning**: "Explain like I'm 5", "Create quiz", "Study notes"

3. **Quick Action Shortcuts**:
   - Keyboard shortcuts for common templates
   - Context menu integration
   - Smart suggestions based on content type
   - One-click actions for frequent tasks

4. **Custom Template Creation**:
   - Visual template editor
   - Variable insertion helpers
   - Template testing and validation
   - Import/export template collections

- Community template sharing

5. **Intelligent Template Suggestions**:
    - Content-based template recommendations
    - Usage pattern analysis
    - Personalized template ordering
    - Smart template completion

SUCCESS CRITERIA:

- Built-in templates cover 80% of common use cases
- Custom template creation is intuitive
- Template suggestions are relevant and helpful
- Quick actions reduce user effort significantly
- Template system improves response quality

DELIVERABLES:

- Comprehensive template system with categories
- Visual template editor with variable support
- Quick action shortcuts and context integration
- Template suggestion and recommendation engine
- Community template sharing platform

```
### 4.3 Multi-Modal Content Support
```

TASK: Implement support for analyzing images, documents, and other media types

CONTEXT:

- Currently only supports text content
- Many webpages have important visual information
- Users want to analyze images, PDFs, and other media
- Need to prepare for future model capabilities

REQUIREMENTS:

1. **Image Analysis Framework**:

```javascript
// Image analysis capabilities
{
  extraction: {
    text: extractTextFromImage(image), // OCR
    objects: detectObjects(image),
    scenes: analyzeScene(image),
    faces: detectFaces(image), // if enabled
    metadata: extractImageMetadata(image)
  },
  analysis: {
    description: generateDescription(image),
    content: analyzeContent(image),
    context: relateTo(image, pageContent),
    accessibility: generateAltText(image)
  }
}
```

2. **Document Processing**:
   - PDF text extraction and analysis
   - Document structure recognition
   - Table and form extraction
   - Slide presentation analysis
   - Spreadsheet data processing

3. **Media Content Handling**:
   - Image screenshot capabilities
   - Video thumbnail analysis
   - Audio transcription (future)
   - Interactive content analysis

4. **Integration with AI Models**:
   - Vision-capable model integration
   - Multi-modal prompt construction
   - Context combination (text + visual)
   - Response formatting for multi-modal content

5. **Privacy and Security**:
   - Local image processing options
   - Privacy controls for sensitive content

- Secure handling of uploaded files
- User consent for media analysis

SUCCESS CRITERIA:

- Accurate image content analysis
- PDF and document processing works reliably
- Multi-modal responses are coherent
- Privacy controls work as expected
- Performance is acceptable for typical use cases

DELIVERABLES:

- Image analysis and OCR capabilities
- Document processing for PDFs and office files
- Multi-modal AI model integration
- Privacy-preserving media processing
- User controls for media analysis features

```
### 4.4 Advanced Context Management
```

TASK: Implement sophisticated context management for better AI conversations

CONTEXT:

- Current context is limited to current page
- No conversation memory or context building
- Missing cross-page context continuation
- No intelligent context prioritization

REQUIREMENTS:

1. **Context Building System**:

```javascript
// Context management structure
{
  session: {
    pages: PageContext[],
    conversations: ConversationContext[],
    timeline: TimelineEvent[],
    relationships: ContextRelationship[]
  },
  page: {
    url: string,
    title: string,
    content: ProcessedContent,
    metadata: PageMetadata,
    interactions: UserInteraction[]
  },
  conversation: {
    id: string,
    context: ConversationContext,
    memory: MemoryItem[],
    preferences: UserPreferences
  }
}
```

2. **Intelligent Context Selection**:
   - Relevance-based context filtering
   - Context window optimization
   - Multi-page context combination
   - Temporal context considerations

3. **Conversation Memory**:
   - Long-term conversation memory
   - Entity and fact extraction
   - Relationship mapping
   - Preference learning

4. **Cross-Page Context**:
   - Context continuation across pages
   - Site-wide context building
   - Research session management
   - Context similarity detection

- Context similarity detection

5. **Context Optimization**:
   - Token-efficient context encoding
   - Context compression techniques
   - Relevance scoring algorithms
   - Context freshness management

SUCCESS CRITERIA:

- AI maintains context across multiple pages
- Conversation memory improves response quality
- Context selection is efficient and relevant
- Cross-page context works seamlessly
- Memory usage is optimized

DELIVERABLES:

- Advanced context management system
- Intelligent context selection algorithms
- Cross-page context continuation
- Conversation memory and learning
- Context optimization and compression

```
---

## PHASE 5: PERFORMANCE & SECURITY

### 5.1 Comprehensive Security Audit
```

TASK: Conduct thorough security audit and implement all necessary hardening measures

CONTEXT:

- Extension handles sensitive API keys and user data
- Potential for XSS and injection attacks
- Need to meet Mozilla security requirements
- User privacy must be protected

REQUIREMENTS:

1. **Security Assessment Framework**:
   - OWASP Top 10 compliance check
   - Firefox extension security guidelines review
   - Automated security scanning integration
   - Manual penetration testing procedures

2. **Input Validation & Sanitization**:

```javascript
// Comprehensive input validation
{
  userInput: {
    sanitization: sanitizeUserInput(input),
    validation: validateInputFormat(input),
    lengthCheck: enforceInputLimits(input),
    contentFilter: filterMaliciousContent(input)
  },
  apiResponses: {
    validation: validateApiResponse(response),
    sanitization: sanitizeApiContent(response),
    contentFilter: filterUnsafeContent(response)
  }
}
```

3. **XSS Prevention**:
   - Complete elimination of innerHTML usage
   - Proper DOM manipulation with createElement
   - Content Security Policy (CSP) enforcement
   - User-generated content sanitization

4. **API Security**:
   - Secure API key storage and transmission
   - Request signing and validation
   - Rate limiting and abuse prevention
   - API endpoint validation

5. **Data Protection**:
   - Encryption of sensitive data at rest
   - Secure transmission protocols
   - Data minimization principles
   - User consent management

6. **Privacy Enhancements**:
    - No tracking or analytics without consent
    - Local data processing prioritization
    - Clear privacy policy and data handling
    - User control over data sharing

SUCCESS CRITERIA:

- Zero high-severity security vulnerabilities
- OWASP compliance achieved
- Mozilla security requirements met
- Privacy policy compliant with regulations
- Security monitoring and alerting active

DELIVERABLES:

- Complete security audit report
- All security vulnerabilities fixed
- Enhanced input validation and sanitization
- Comprehensive security testing suite
- Privacy-compliant data handling

```
### 5.2 Performance Optimization
```

TASK: Optimize application performance for speed, memory usage, and efficiency

CONTEXT:

- Need to minimize extension impact on browser performance
- Large conversations can cause memory issues
- Bundle size affects loading times
- Mobile performance needs optimization

REQUIREMENTS:

1. **Performance Monitoring**:

```javascript
// Performance metrics tracking
{
  memory: {
    usage: getCurrentMemoryUsage(),
    peak: getpeakMemoryUsage(),
    leaks: detectMemoryLeaks()
  },
  timing: {
    startup: measureStartupTime(),
    responseTime: measureApiResponseTime(),
    renderTime: measureRenderTime()
  },
  bundleSize: {
    total: getTotalBundleSize(),
    byModule: getBundleSizeByModule(),
    compression: getCompressionRatio()
  }
}
```

2. **Bundle Optimization**:
   - Code splitting and lazy loading
   - Tree shaking for unused code
   - Asset compression and minification
   - Dynamic imports for optional features

3. **Memory Management**:
   - Efficient DOM manipulation
   - Proper event listener cleanup
   - Conversation data pagination
   - Virtual scrolling for large lists

4. **Rendering Optimization**:
   - Efficient re-rendering strategies
   - Debounced updates for streaming
   - GPU-accelerated animations
   - Optimized CSS for performance

5. **Network Optimization**:
   - Request caching and deduplication

- Optimized API payloads
- Background data fetching
- Offline capability preparation

SUCCESS CRITERIA:

- Extension startup time under 100ms
- Memory usage under 50MB baseline
- Bundle size under 500KB compressed
- 60fps UI performance maintained
- No memory leaks detected

DELIVERABLES:

- Performance monitoring dashboard
- Optimized bundle with lazy loading
- Memory leak prevention system
- Performance testing automation
- Mobile performance optimizations

```
### 5.3 Error Monitoring and Analytics
```

TASK: Implement comprehensive error monitoring and privacy-preserving analytics

CONTEXT:

- Need to monitor extension health in production
- Error tracking essential for user support
- Analytics needed for feature usage insights
- Must respect user privacy preferences

REQUIREMENTS:

1. **Error Monitoring System**:

```javascript
// Error tracking framework
{
  collection: {
    errors: collectErrors(),
    performance: collectPerformanceMetrics(),
    usage: collectUsageStats(),
    crashes: collectCrashReports()
  },
  analysis: {
    errorCategorization: categorizeErrors(),
    trendAnalysis: analyzeErrorTrends(),
    impactAssessment: assessErrorImpact(),
    rootCauseAnalysis: performRootCauseAnalysis()
  },
  reporting: {
    dashboards: createErrorDashboards(),
    alerts: setupErrorAlerts(),
    reports: generateErrorReports()
  }
}
```

2. **Privacy-Preserving Analytics**:
   - Opt-in analytics with clear consent
   - Data anonymization and aggregation
   - No personally identifiable information
   - User control over data sharing

3. **Performance Monitoring**:
   - Real-time performance metrics
   - User experience monitoring
   - API performance tracking
   - Resource usage monitoring

4. **Health Monitoring**:
   - Extension availability monitoring
   - Feature usage tracking
   - Success rate monitoring
   - User satisfaction metrics

5. **Incident Response**:

- Automated alerting for critical issues
- Incident response procedures
- User notification system
- Emergency rollback capabilities

SUCCESS CRITERIA:

- 99.9% error detection rate
- Mean time to detection under 5 minutes
- Privacy compliance maintained
- Actionable insights from analytics
- Proactive issue identification

DELIVERABLES:

- Comprehensive error monitoring system
- Privacy-preserving analytics platform
- Performance monitoring dashboard
- Incident response automation
- User feedback collection system

```
### 5.4 Testing and Quality Assurance
```

TASK: Implement comprehensive testing suite and quality assurance processes

CONTEXT:

- Need 90%+ test coverage for production readiness
- Multiple browsers and devices need testing
- Automated testing essential for CI/CD
- Quality gates required for releases

REQUIREMENTS:

1. **Test Suite Expansion**:

```javascript
// Comprehensive test coverage
{
  unit: {
    functions: testAllFunctions(),
    components: testAllComponents(),
    utilities: testAllUtilities(),
    coverage: achieveTargetCoverage(90)
  },
  integration: {
    apiIntegration: testApiIntegration(),
    storageIntegration: testStorageIntegration(),
    uiIntegration: testUIIntegration()
  },
  e2e: {
    userJourneys: testUserJourneys(),
    crossBrowser: testCrossBrowser(),
    performance: testPerformance(),
    accessibility: testAccessibility()
  }
}
```

2. **Automated Testing Infrastructure**:
   - CI/CD pipeline integration
   - Cross-browser testing automation
   - Visual regression testing
   - Performance testing automation

3. **Quality Gates**:
   - Test coverage requirements (90%+)
   - Performance benchmarks
   - Security scan requirements
   - Accessibility compliance checks

4. **Manual Testing Procedures**:
   - User acceptance testing protocols
   - Accessibility testing procedures
   - Security testing procedures
   - Usability testing framework

5. **Test Data Management**:

- Test data generation
- Mock data for different scenarios
- Test environment management
- Data cleanup procedures

SUCCESS CRITERIA:

- 90%+ test coverage achieved
- All tests pass in CI/CD pipeline
- Cross-browser compatibility verified
- Performance benchmarks met
- Accessibility compliance confirmed

DELIVERABLES:

- Comprehensive test suite with 90%+ coverage
- Automated testing infrastructure
- Quality gates and compliance checks
- Manual testing procedures and protocols
- Test data management system

PHASE 6: PRODUCTION RELEASE

6.1 Mozilla Add-ons Compliance

TASK: Ensure full compliance with Mozilla Add-ons policies and requirements

CONTEXT:

- Must meet all Mozilla Add-ons requirements
- Need to pass automated and manual review
- Compliance with privacy and security policies
- Preparation for store submission

REQUIREMENTS:

1. **Policy Compliance Checklist**:
   - Content policy compliance
   - Privacy policy requirements
   - Security requirements
   - User interface guidelines
   - Performance requirements

2. **Technical Requirements**:

```javascript
// Mozilla compliance checklist
{
  manifest: {
    validation: validateManifest(),
    permissions: validatePermissions(),
    contentSecurity: validateCSP(),
    icons: validateIcons()
  },
  code: {
    minification: false, // Mozilla requirement
    obfuscation: false, // Mozilla requirement
    externalCode: validateExternalCode(),
    maliciousCode: scanForMaliciousCode()
  },
  privacy: {
    dataCollection: documentDataCollection(),
    userConsent: implementUserConsent(),
    thirdPartyServices: documentThirdPartyServices()
  }
}
```

3. **Documentation Requirements**:
   - Privacy policy creation
   - User documentation
   - Developer documentation
   - Support documentation

4. **Review Preparation**:
   - Source code documentation
   - Build process documentation
   - Testing documentation
   - Security documentation

SUCCESS CRITERIA:

- All Mozilla policies compliance verified
- Automated review passes successfully
- Documentation complete and accurate
- Privacy policy legally compliant
- Ready for store submission

DELIVERABLES:

- Mozilla-compliant extension package
- Complete privacy policy and documentation
- Compliance verification report
- Store submission materials
- Review response procedures

```
### 6.2 User Documentation and Support
```

TASK: Create comprehensive user documentation and support infrastructure

CONTEXT:

- Users need clear guidance on features and usage
- Support system needed for user questions
- Documentation must be accessible and searchable
- Multiple formats needed for different user preferences

REQUIREMENTS:

1. **User Documentation Structure**:

```markdown
# Documentation Structure
- Getting Started Guide
- Feature Documentation
- Troubleshooting Guide
- FAQ
- Video Tutorials
- API Documentation
- Developer Guide
```

2. **Multi-Format Documentation**:
   - Interactive in-app help
   - Online documentation website
   - Video tutorials and demos
   - PDF guides for offline reference
   - Searchable knowledge base

3. **Support Infrastructure**:
   - Help desk system
   - Community forum
   - GitHub issues management
   - User feedback collection
   - Live chat support (optional)

4. **Accessibility Compliance**:
   - Screen reader compatible documentation
   - High contrast documentation themes
   - Keyboard navigation for all help systems
   - Multiple language support preparation

5. **Maintenance and Updates**:
   - Documentation versioning system
   - Automated documentation updates
   - User feedback integration
   - Regular content review and updates

SUCCESS CRITERIA:

- Complete documentation covers all features
- Documentation is accessible to all users
- Support response time under 24 hours
- User satisfaction with documentation above 85%
- Self-service resolution rate above 70%

DELIVERABLES:

- Comprehensive user documentation
- Support infrastructure and procedures
- Accessibility-compliant help system
- User feedback and improvement process
- Documentation maintenance system

```
### 6.3 Launch Strategy and Marketing
```

TASK: Develop and execute comprehensive launch strategy and marketing plan

CONTEXT:

- Need to achieve visibility in crowded extension market
- Target Firefox users specifically
- Differentiate from Chrome-focused alternatives
- Build community and user base

REQUIREMENTS:

1. **Launch Strategy**:

```javascript
// Launch phases
{
  beta: {
    duration: '4 weeks',
    participants: 100,
    feedback: 'intensive',
    features: 'core functionality'
  },
  softLaunch: {
    duration: '2 weeks',
    audience: 'Firefox communities',
    features: 'full feature set',
    monitoring: 'extensive'
  },
  publicLaunch: {
    duration: 'ongoing',
    channels: 'all marketing channels',
    goal: '1000 users in 30 days'
  }
}
```

2. **Marketing Materials**:
   - Extension store listing optimization
   - Website and landing page
   - Video demonstrations
   - Social media content
   - Press release and media kit

3. **Community Engagement**:
   - Firefox community outreach
   - Reddit and forum engagement
   - Developer community engagement
   - User feedback collection
   - Community building initiatives

4. **Content Marketing**:
   - Blog posts about features
   - Comparison content vs. alternatives
   - Tutorial and how-to content

- Case studies and success stories

5. **Performance Tracking**:
   - User acquisition metrics
   - Retention and engagement metrics
   - Conversion funnel analysis
   - User feedback analysis

SUCCESS CRITERIA:

- 1000+ active users within 30 days
- 4.5+ star average rating
- 70%+ user retention after 7 days
- Featured placement consideration
- Positive community feedback

DELIVERABLES:

- Complete launch strategy and timeline
- Marketing materials and content
- Community engagement plan
- Performance tracking dashboard
- Post-launch optimization plan

```
### 6.4 Monitoring and Maintenance
```

TASK: Establish ongoing monitoring and maintenance procedures for production

CONTEXT:

- Need continuous monitoring of extension health
- Regular updates and maintenance required
- User feedback must be incorporated
- Long-term sustainability planning

REQUIREMENTS:

1. **Production Monitoring**:

```javascript
// Monitoring dashboard
{
  health: {
    uptime: monitorUptime(),
    errorRate: monitorErrorRate(),
    performance: monitorPerformance(),
    userSatisfaction: monitorSatisfaction()
  },
  usage: {
    activeUsers: trackActiveUsers(),
    featureUsage: trackFeatureUsage(),
    retention: trackRetention(),
    growth: trackGrowth()
  },
  technical: {
    apiHealth: monitorApiHealth(),
    browserCompatibility: monitorCompatibility(),
    security: monitorSecurity(),
    performance: monitorPerformance()
  }
}
```

2. **Maintenance Procedures**:
   - Regular security updates
   - Bug fix prioritization
   - Feature enhancement planning
   - Performance optimization
   - Compatibility maintenance

3. **User Feedback Integration**:
   - Feedback collection and analysis
   - Feature request management
   - Bug report handling
   - User satisfaction surveys

4. **Release Management**:
   - Release planning and scheduling
   - Automated testing and deployment
   - Rollback procedures

- Communication plans

5. **Long-term Planning**:
    - Roadmap development
    - Technology evolution planning
    - Market analysis and adaptation
    - Resource planning

SUCCESS CRITERIA:

- 99.9% uptime maintained
- Critical bugs fixed within 24 hours
- User satisfaction above 85%
- Regular feature updates delivered
- Sustainable development process

DELIVERABLES:

- Production monitoring system
- Maintenance procedures and automation
- User feedback integration system
- Release management process
- Long-term sustainability plan

EXECUTION GUIDELINES

Context Management for Claude Code

For each prompt, provide:

1. **Current Project State**:
   - Complete file structure
   - Recent changes and updates
   - Current test results
   - Performance metrics

2. **Integration Requirements**:
   - Dependencies on previous phases
   - Integration points with existing code
   - Compatibility requirements
   - Performance constraints

3. **Success Validation**:
   - Specific testing requirements
   - Performance benchmarks
   - User acceptance criteria
   - Quality gates

Prompt Execution Order

Execute prompts in the specified phase order. Do not proceed to the next phase until:
- All deliverables are complete
- All tests pass
- Performance benchmarks are met
- Code review is complete
- Integration testing passes

Quality Assurance

After each phase:
1. Run complete test suite
2. Verify performance benchmarks
3. Check security compliance
4. Validate user experience
5. Review code quality metrics

Risk Mitigation

- Maintain backup branches for each phase
- Implement feature flags for new functionality
- Create rollback procedures for each release
- Monitor user feedback continuously
- Maintain emergency response procedures

This comprehensive roadmap will transform DeepWeb from its current state into a production-ready, market-leading Firefox AI assistant. Each prompt is designed to be complete and actionable, providing Claude Code with all necessary context and requirements for successful implementation.