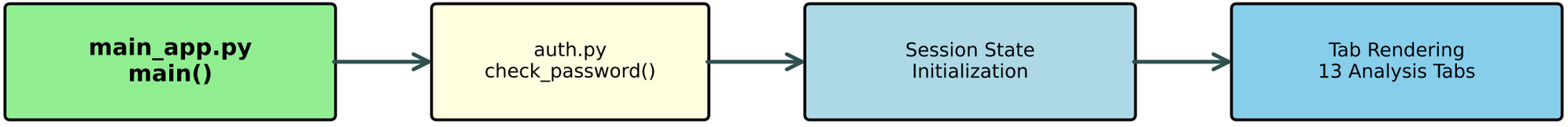
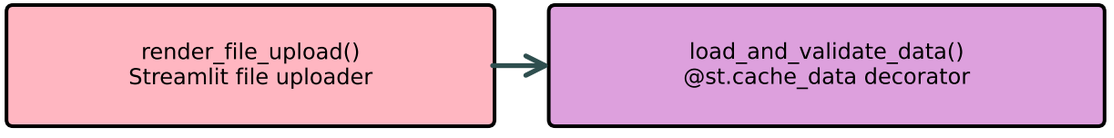
Geotechnical Data Analysis Application

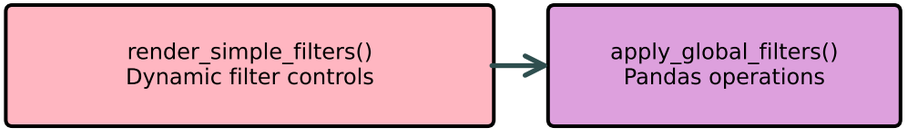
Application Overview & Entry Points



File Upload & Data Loading



Global Data Filtering



Key Application Statistics:
- 13 Analysis Tabs (all rendered simultaneously)
- 18 Utility Modules in utils/ folder
- 21 Plotting Functions in Functions/ folder
- 25+ Parameters in CBR/WPI tab alone
- 2-4 second response time per parameter change
- No parameter change isolation (major bottleneck)

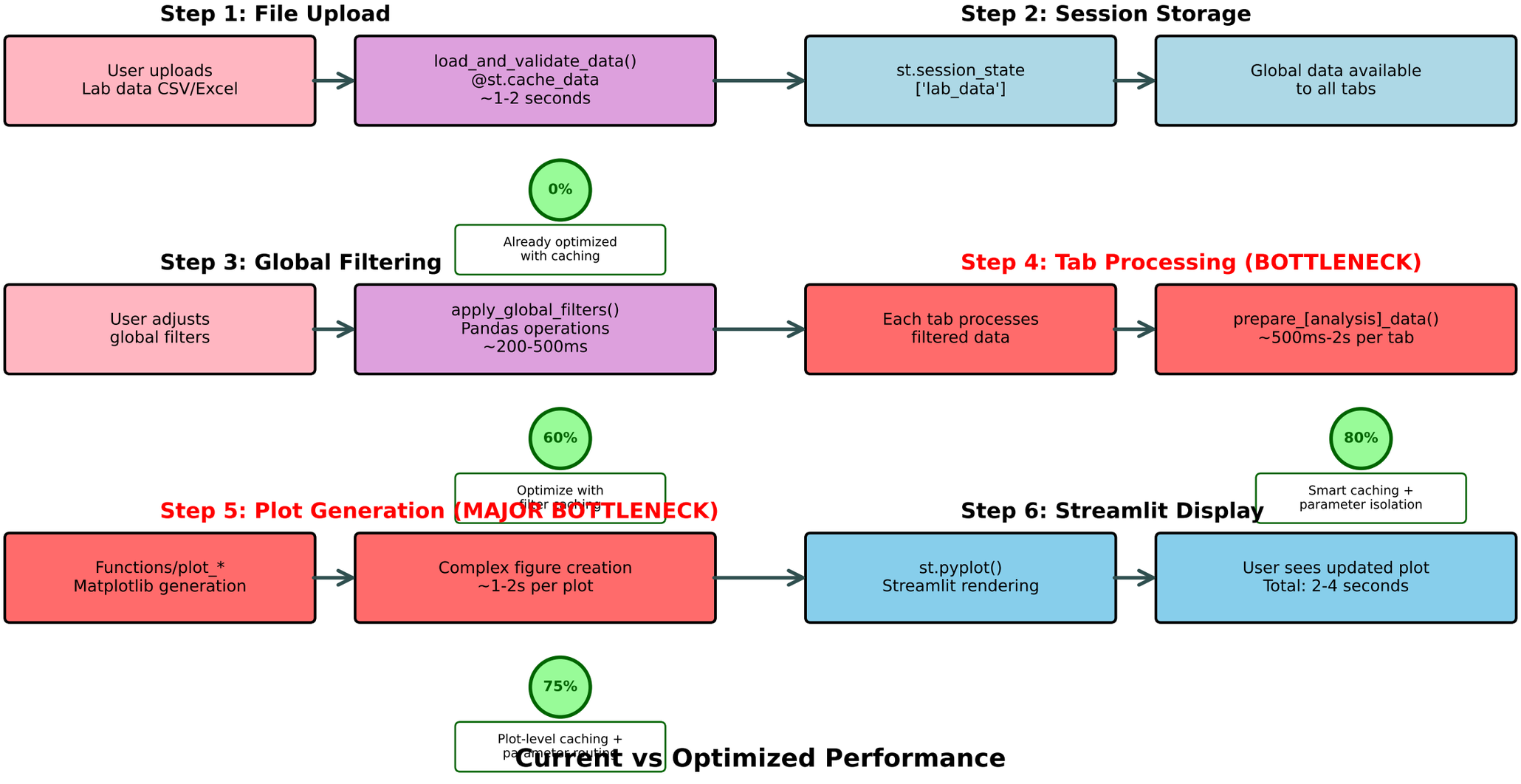
Architecture Highlights

STRENGTHS:
- Modular design
- Cached data loading
- Professional UI
- Golden standard workflows

BOTTLENECKS:
- No parameter isolation
- All tabs render together
- Heavy reprocessing
- Poor responsiveness

OPPORTUNITIES:
- 3-5x performance gain
- Smart caching strategy
- Parameter classification
- Progressive loading

Data Flow & Processing Pipeline

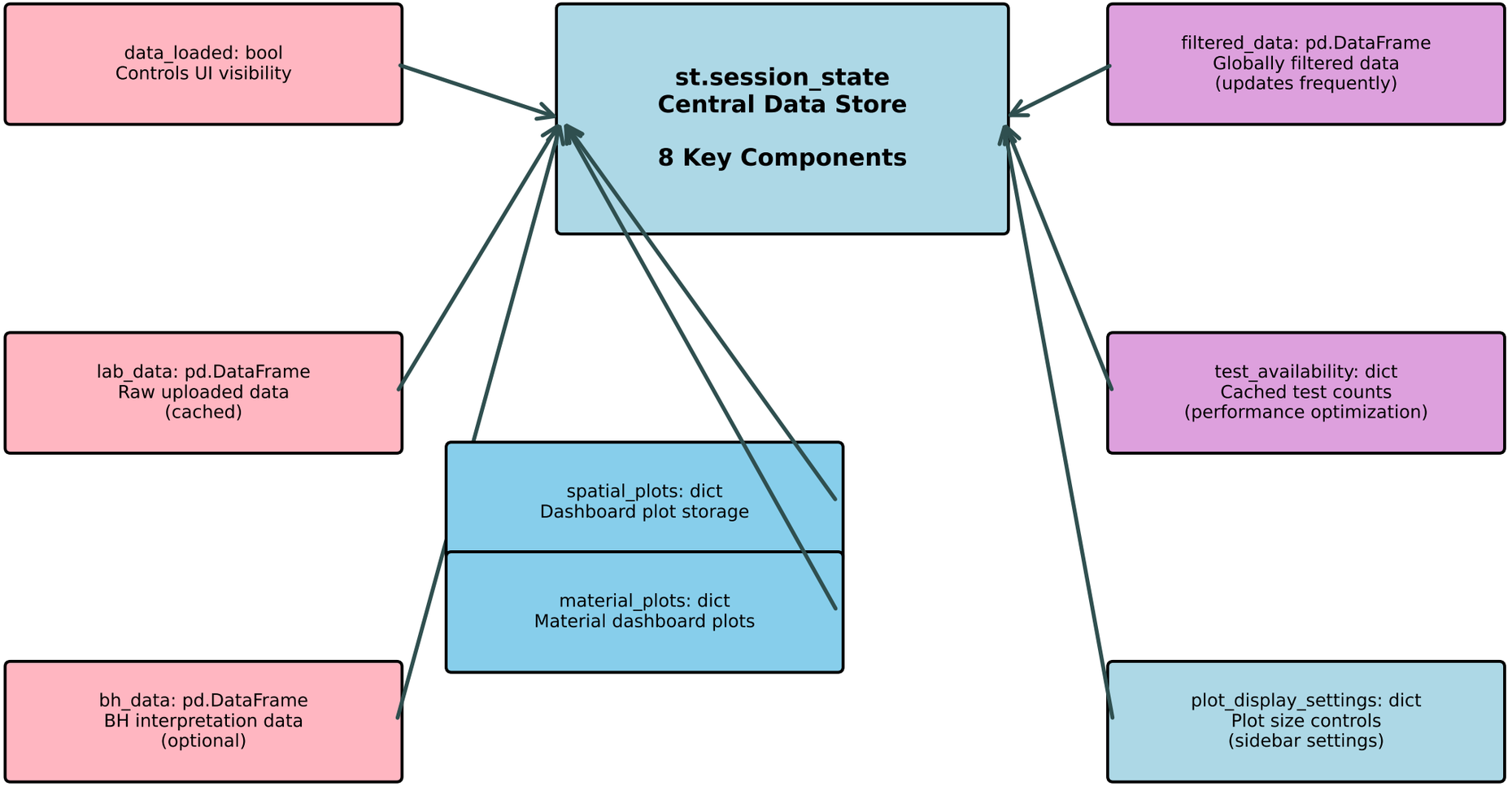


Current vs Optimized Performance

CURRENT WORKFLOW:
ANY parameter change →
Complete reprocessing →
Full plot regeneration →
2-4 second delay

OPTIMIZED WORKFLOW:
Light parameters → cached plot (~200ms)
Medium parameters → filter only (~800ms)
Heavy parameters → smart reprocess (~1s)

Session State Management Architecture



Session State Characteristics

CURRENT BEHAVIOR:
• Global scope affects all tabs
• No tab-specific isolation
• Manual cache invalidation
• Persistent across user interactions
• Grows linearly with data size

OPTIMIZATION OPPORTUNITIES:
• Implement tab-specific namespaces
• Smart garbage collection
• Lazy loading of heavy components
• Memory-efficient data structures
• Automated cache cleanup

Tab Architecture & Rendering Patterns

Data
Overview
Low | 200-500ms

PSD
Analysis
Medium | 500ms-1s

Atterberg
Plasticity
Medium | 300-800ms

SPT
Analysis
Medium | 400ms-1s

Emerson
Classification
Low | 300-600ms

UCS vs Depth
Strength
Medium | 500ms-1s

UCS vs Is50
Correlation
Medium | 400-800ms

Property vs Depth
Depth Analysis
Medium | 600ms-1.2s

Property vs Chainage
Spatial Analysis
Medium | 500ms-1s

Thickness Analysis
Layer Analysis
Medium | 400-900ms

Histograms
General Analysis
High | 1-2s

CBR Swell/WPI
Classification
Very High | 2-4s

Export
Batch Operations
Medium | 500ms-1.5s

80%

Primary optimization target

CURRENT: All 13 tabs render simultaneously → 2-3s tab switch

OPTIMIZED: Lazy loading only active tab renders → <100ms tab switch

Standard Tab Rendering Pattern

1. Parameter Collection
• UI controls with unique keys
• Streamlit widgets in expanders

2. Data Processing (tab-specific)
• prepare_[analysis]_data()
• Complex pandas operations

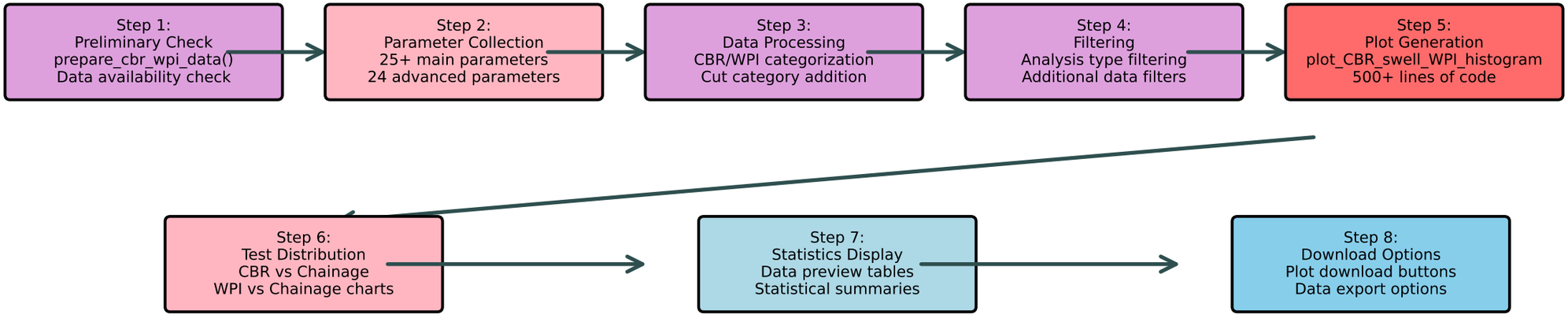
3. Plotting (Functions/ folder)
• plot_[analysis]() functions
• Matplotlib figure generation

4. Display & Download
• st.pyplot() rendering
• Download button generation

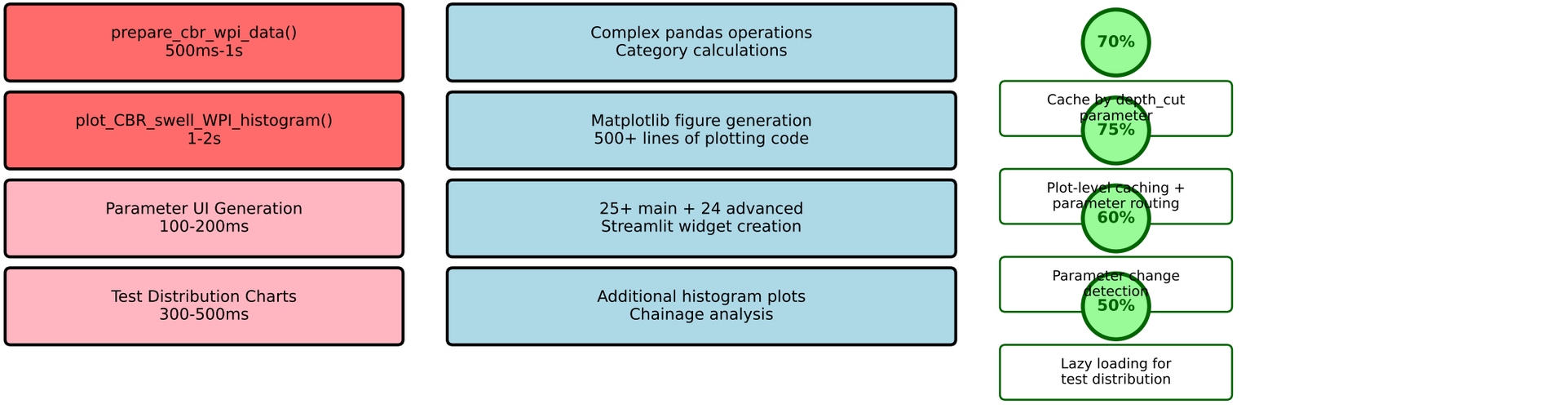
BOTTLENECK: All tabs render simultaneously
OPTIMIZATION: Lazy loading + tab isolation

CBR/WPI Analysis Tab - Deep Dive

Most Complex & Performance-Critical Component



CBR/WPI Performance Analysis



TOTAL CBR/WPI IMPACT: 2-4 seconds per parameter change | OPTIMIZATION POTENTIAL: 3-5x improvement (200ms-1s response)

Parameter Dependencies & Impact Analysis

Smart Parameter Classification for Optimization

LIGHT PARAMETERS

UI-only changes | Expected: ~200ms | Current: 2-4s

- stack_by - Plot grouping
- analysis_type - Data filtering
- cmap_name - Color scheme
- bar_alpha - Transparency
- show_grid - Grid visibility
- show_legend - Legend display
- title - Plot title text
- custom_ylabel - Y-axis label
- xlim, ylim - Axis limits
- figsize - Figure dimensions

85%

Keep processed data
Re-plot only

MEDIUM PARAMETERS

Data filtering | Expected: ~800ms | Current: 2-4s

- filter1_col - First filter column
- filter1_value - First filter value
- filter2_col - Second filter column
- filter2_value - Second filter value
- facet_order - Panel sorting
- category_order - X-axis order

70%

Keep base data
Apply filters only

HEAVY PARAMETERS

Complete reprocessing | Expected: ~1s | Current: 2-4s

- depth_cut - Cut category calculation
(Triggers complete data reprocessing)

Future heavy parameters:
- • New categorization rules
- • Data source changes
- • Algorithm modifications

50%

Smart caching
with invalidation

Current vs Optimized Parameter Handling

CURRENT WORKFLOW (INEFFICIENT):
ANY parameter change →
main() rerun →
prepare_cbr_wpi_data() →
plot_CBR_swell_WPI_histogram() →
Full re-render (2-4 seconds)

PROBLEM: Changing 'alpha' takes same time as 'depth_cut'

OPTIMIZED WORKFLOW (INTELLIGENT):
Parameter change detection →
Route by impact level →

Light: cached_data → re-plot (200ms)
Medium: cached_base → filter → plot (800ms)
Heavy: full reprocessing (1s)

RESULT: 3-5x performance improvement

IMPLEMENTATION STRATEGY: Parameter state tracking + Change detection + Routing by impact + Intelligent caching = 3-5x performance gain

Performance Bottlenecks & Root Cause Analysis

CRITICAL ISSUE	BOTTLENECK #1: Complete App Rerun Every parameter change triggers main() rerun ALL 13 tabs re-render simultaneously Complete session state refresh	COST: 2-3s per interaction	FREQUENCY: Every user interaction	50% 50% improvement possible
CRITICAL ISSUE	BOTTLENECK #2: Expensive Data Processing prepare_cbr_wpi_data() runs on every change Complex category calculations Data concatenation operations	COST: 500ms-1s per execution	FREQUENCY: Every CBR/WPI parameter	60% 60% improvement possible
CRITICAL ISSUE	BOTTLENECK #3: Heavy Plotting Function plot_CBR_swell_WPI_histogram() (500+ lines) Complete matplotlib figure generation Complex styling and data grouping	COST: 1-2s per plot	FREQUENCY: Every CBR/WPI parameter	50% 50% improvement possible
CRITICAL ISSUE	BOTTLENECK #4: No Parameter Isolation No distinction between light vs heavy changes Changing alpha affects same workflow as depth_cut Unnecessary reprocessing	COST: Wasted processing time	FREQUENCY: 80% of parameter changes	70% 70% improvement possible
CRITICAL ISSUE	BOTTLENECK #5: Redundant Session Operations Session state updated unnecessarily Memory operations and serialization No cleanup of intermediate data	COST: 100-200ms overhead	FREQUENCY: Every interaction	70% 70-85% improvement possible

Current Performance Measurements

Light Parameter Change	2-4 seconds	Should be 500ms	75-85% improvement possible
Medium Parameter Change	2-4 seconds	Should be 800ms-1.2s	40-60% improvement possible
Heavy Parameter Change	2-4 seconds	Should be 1s	25% improvement possible
Tab Switching	2-3 seconds	Should be instant	95% improvement possible
File Upload (first time)	3-5 seconds	Acceptable	Already optimized with caching

OVERALL IMPACT: Poor user experience, development inefficiency, reduced adoption potential
OPTIMIZATION POTENTIAL: 3-5x overall performance improvement through intelligent caching and parameter isolation

Optimization Strategies & Implementation Plan

STRATEGY 1: Intelligent Caching System

Data Processing Cache:
• @st.cache_data for prepare_cbr_wpi_data()
• Hash by key parameters (depth_cut, data_hash)
• Automatic invalidation on data changes

Plot Generation Cache:
• Cache matplotlib figures by parameter hash
• Memory-efficient storage
• Cleanup old cached plots

Filter Operations Cache:
• Cache intermediate filtering results
• Smart cache invalidation
• Reduced pandas operations

70%

Expected:
70% improvement

STRATEGY 2: Parameter Change Detection

Smart State Management:
• Track previous parameter values
• Classify changes by impact level
• Route to appropriate processing strategy

Parameter Classification:
• Heavy: depth_cut → full reprocessing
• Medium: filters → filter-only processing
• Light: styling → re-plot only

Processing Strategy Selection:
• Minimize unnecessary operations
• Preserve cached data when possible
• Intelligent workflow routing

60%

Expected:
60% improvement

STRATEGY 3: Progressive Enhancement

Loading States:
• Contextual st.spinner() indicators
• Operation-specific feedback
• Cancellable long operations

User Experience:
• Clear progress indication
• Professional loading states
• Skeleton loading for plots

Performance Feedback:
• Real-time performance metrics
• Cache hit rate display
• Optimization suggestions

0%

Expected:
Better UX

STRATEGY 4: Lazy Tab Loading

Tab State Isolation:
• Only render active tab content
• Separate session state namespaces
• Independent parameter management

Performance Benefits:
• 95% reduction in tab switching time
• Reduced memory usage
• Better responsiveness

95%

Expected:
95% improvement
in tab switching

Implementation Code Examples

```
# Smart caching implementation
@st.cache_data(hash_funcs={pd.DataFrame: lambda df: str(df.shape)})
def prepare_cbr_wpi_data_cached(data_hash, depth_cut):
    return prepare_cbr_wpi_data(filtered_data, depth_cut)

# Parameter change detection
def detect_parameter_changes(current, previous):
    heavy_changed = current['depth_cut'] != previous.get('depth_cut')
    if heavy_changed:
        return 'full_reprocess'
    # ... additional logic

# Progressive loading
with st.spinner("Processing data with new depth cut..."):
    data = prepare_cbr_wpi_data_cached(filtered_data, depth_cut)
```

COMBINED IMPACT: 3-5x overall performance improvement | Light parameters: 2-4s → 200ms | Medium parameters: 2-4s → 800ms | Heavy parameters: 2-4s → 1s

Implementation Roadmap & Success Metrics

PHASE 1: Critical Performance Fixes (Week 1)

HIGH PRIORITY - Immediate Impact:

✅ Task 1.1: Fix depth_cut variable error (COMPLETED)
• Effort: 30 minutes | Impact: Application functionality

🔄 Task 1.2: Add caching to prepare_cbr_wpi_data()
• Effort: 2-3 hours | Impact: 70% improvement

🔄 Task 1.3: Implement parameter change detection
• Effort: 4-6 hours | Impact: 60% reduction in processing

🔄 Task 1.4: Add progressive loading indicators
• Effort: 2 hours | Impact: Better user experience

EXPECTED RESULTS:
• Light parameters: 2-4s → 500ms (75% improvement)
• Heavy parameters: 2-4s → 2s (stable performance)

PHASE 2: Smart Optimization (Week 2)

MEDIUM PRIORITY - Substantial Improvement:

🔄 Task 2.1: Plot-level caching
• Effort: 1-2 days | Impact: 50% improvement

🔄 Task 2.2: Tab state isolation
• Effort: 2-3 days | Impact: Isolate tab parameters

🔄 Task 2.3: Enhanced progressive enhancement
• Effort: 1 day | Impact: Professional UX

🔄 Task 2.4: Memory optimization
• Effort: 1 day | Impact: Reduced memory usage

EXPECTED RESULTS:
• Light parameters: 500ms → 200ms (60% additional)
• Medium parameters: 2-4s → 800ms (70% improvement)

PHASE 3: Advanced Features (Week 3)

LOW PRIORITY - Long-term Enhancement:

⚡ Task 3.1: Async processing
• Effort: 2-3 days | Impact: Non-blocking UI

⚡ Task 3.2: Pre-computation strategy
• Effort: 2 days | Impact: Instant common scenarios

⚡ Task 3.3: Incremental data updates
• Effort: 3-4 days | Impact: Surgical updates

EXPECTED RESULTS:
• Near-instant cached scenarios
• Background processing
• Enterprise-grade performance

Success Metrics & Validation

Performance Targets by Phase:

	Current	Phase 1	Phase 2	Phase 3
Light Parameters	2-4s	500ms	200ms	<100ms
Medium Parameters	2-4s	2s	800ms	400ms
Heavy Parameters	2-4s	2s	1.5s	1s
Tab Switching	2-3s	2s	100ms	<50ms
Overall Rating	Poor	Good	Excellent	Outstanding

Technical Metrics:
• Response time reduction: 3-5x improvement target
• Memory usage optimization: 30-50% reduction
• Cache hit rate: >80% for common operations
• Error rate: <1% for all parameter combinations

User Experience Metrics:
• User satisfaction surveys and feedback
• Task completion time measurements
• Feature adoption rates and usage patterns
• Support ticket reduction and issue resolution

IMMEDIATE NEXT STEPS: 1) Implement Phase 1 caching 2) Add parameter detection 3) Test and validate improvements 4) Apply patterns to other tabs