Supplementary Material

1 LLM Prompting Templates

This section provides comprehensive details of the prompting strategies employed across different phases of the algorithmic discovery process. The templates are designed to guide language models through systematic reasoning while incorporating domain-specific knowledge and maintaining consistency across evolutionary operations.

System Context and Task Definition. All interactions with the LLM ensemble begin with a standardized system prompt that establishes the expert role and problem context:

```
You are an expert in gravitational wave signal detection algorithms. Your task is to design
 heuristics that can effectively solve optimization problems. The task involves
constructing a pipeline for gravitational wave signal detection. This pipeline will
encompass data conditioning and time-frequency transformations as part of the signal
processing workflow. The input will consist of raw, finite-length dual-channel
gravitational wave data from the H1 and L1 detectors. The pipeline will be tested on
segmented data spanning several weeks, with each segment having variable length (7000s
-30000s). Each segment's dual-channel data will be directly used as input. The ultimate
goal is to produce a catalog of potential gravitational wave signals, where each trigger
includes information such as GPS time, ranking statistic, and the timing accuracy of the
prediction. This systematic approach is essential for effectively identifying and
cataloging candidate gravitational wave signals.
```

This system prompt serves multiple purposes: (i) establishing domain expertise expectations, (ii) defining the specific optimization context, (iii) specifying input data characteristics, and (iv) clarifying the expected output format and evaluation criteria.

1.1 Initial Algorithm Generation Prompts

Seed Function Template and Analysis Framework. The initial algorithm generation process begins with a structured analysis of the seed function to establish baseline understanding. The seed function analysis template guides the LLM through systematic examination of the foundational algorithm:

```
## Seed Function Analysis Task
Analyze the foundational algorithm's design strategy to establish baseline understanding
for Monte Carlo Tree Search (MCTS) exploration. This first-level analysis will guide
subsequent optimization directions.

## Seed Function Implementation
```python
{prompt_seed_func}
```
```

```
- **Technical implementation details**: {prompt_other_inf}
- **Performance impact rationale**: {prompt_inout_inf}

## Context for Analysis
This initial analysis at MCTS depth first-level should:
- Identify core algorithmic mechanisms
- Extract fundamental processing stages
- Surface high-level optimization opportunities
- Establish baseline for diversity generation
{external_knowledge}

## Analysis Requirements
1. Characterize the seed's core approach in one sentence containing:
    - Primary computational strategy
    - Key transformation stages
    - Fundamental signal processing techniques
    - Overall optimization philosophy

2. Focus on architectural-level characteristics rather than implementation details

3. Description must fit within single braces and avoid:
    - Code references
    - Parameter-level details
    - Performance assessments
    - Comparative statements

## Output Format Rules
- Return optimization strategies within SINGLE BRACE
- Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
```

Seed Algorithm Specification. The seed function implements a three-stage linear signal processing pipeline that serves as the evolutionary starting point:

- Stage 1: Data Conditioning and Whitening

```python
def data_conditioning(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    window_length = 4096
    dt = times[1] - times[0]
    fs = 1.0 / dt

    def whiten_strain(strain):
        strain_zeromean = strain - np.mean(strain)
        freqs, psd = signal.welch(strain_zeromean, fs=fs, nperseg=window_length,
                                  window='hann', noverlap=window_length//2)
        smoothed_psd = np.convolve(psd, np.ones(32) / 32, mode='same')
        smoothed_psd = np.maximum(smoothed_psd, np.finfo(float).tiny)
        white_fft = np.fft.rfft(strain_zeromean) / np.sqrt(np.interp(np.fft.rfftfreq(len(strain_zeromean), d=dt), freqs, smoothed_psd))
        return np.fft.irfft(white_fft)

    whitened_h1 = whiten_strain(strain_h1)
    whitened_l1 = whiten_strain(strain_l1)

    return whitened_h1, whitened_l1, times
```

- Stage 2: Time-Frequency Decomposition

```python
def compute_metric_series(h1_data: np.ndarray, l1_data: np.ndarray, time_series: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    fs = 1 / (time_series[1] - time_series[0])
    f_h1, t_h1, Sxx_h1 = signal.spectrogram(h1_data, fs=fs, nperseg=256, noverlap=128, mode='magnitude', detrend=False)
    f_l1, t_l1, Sxx_l1 = signal.spectrogram(l1_data, fs=fs, nperseg=256, noverlap=128, mode='magnitude', detrend=False)
    tf_metric = np.mean((Sxx_h1**2 + Sxx_l1**2) / 2, axis=0)
    gps_mid_time = time_series[0] + (time_series[-1] - time_series[0]) / 2
    metric_times = gps_mid_time + (t_h1 - t_h1[-1] / 2)
```

```
    return tf_metric, metric_times
```

- Stage 3: Peak Detection and Trigger Generation

```
def calculate_statistics(tf_metric, t_h1):
    background_level = np.median(tf_metric)
    peaks, _ = signal.find_peaks(tf_metric, height=background_level * 1.0, distance=2, prominence=background_level * 0.3)
    peak_times = t_h1[peaks]
    peak_heights = tf_metric[peaks]
    peak_deltat = np.full(len(peak_times), 10.0)  # Fixed uncertainty value
    return peak_times, peak_heights, peak_deltat
```

Template Variables and Customization. The prompting template incorporates several customizable variables that enable systematic variation generation:

- `prompt_seed_func`: Complete seed function implementation
- `prompt_other_inf`: Technical implementation details including sampling rates, window parameters, and algorithmic constraints
- `prompt_inout_inf`: Performance impact rationale explaining the relationship between input characteristics and expected output quality
- `external_knowledge`: Domain-specific knowledge injection including gravitational wave physics, detector characteristics, and signal morphology constraints

Output Format Requirements. All generated algorithms must conform to the standardized interface:

```
def pipeline_v{N}(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) -> tuple[np.ndarray, np.ndarray,
↪ np.ndarray]:
    # Algorithm implementation for seed function
    # ...
    # Stage 1: Data Conditioning and Whitening
    whitened_h1, whitened_l1, data_times = data_conditioning(strain_h1, strain_l1, times)
    # Stage 2: Time-Frequency Decomposition
    tf_metric, metric_times = compute_metric_series(whitened_h1, whitened_l1, data_times)
    # Stage 3: Peak Detection and Trigger Generation
    peak_times, peak_heights, peak_deltat = calculate_statistics(tf_metric, metric_times)
    return peak_times, peak_heights, peak_deltat
```

This interface consistency ensures that all generated algorithms can be evaluated within the same framework while enabling diverse internal implementations.

1.2 Parent Crossover Implementation

The Parent Crossover (PC) operation represents a sophisticated genetic operation that combines algorithmic components from two reference implementations at different levels of the MCTS hierarchy. This operation is designed to preserve successful characteristics from both parent algorithms while introducing novel enhancements that exceed simple interpolation.

Template Structure and Crossover Strategy. The PC operation employs a structured template that guides the LLM through systematic analysis and synthesis of two parent algorithms:

```
## Task Overview
Develop a novel algorithm that strategically combines components from two reference
implementations while introducing innovative enhancements. The solution must demonstrate
measurable improvements beyond simple interpolation of existing approaches.
Current Depth Level: [Level {depth}]

## Implementation Analysis
### Code Comparison
1. VERSION A (Baseline Implementation):
```python
{worse_code}
```

2. VERSION B (Enhanced Implementation):
```python
{better_code}
```

Strengths to Combine
```text
{reflection}
```

Key Synthesis Requirements:
- Preserve 2 distinct advantages from Version A
- Incorporate 3 critical enhancements from Version B
- Identify 1 synergistic improvement opportunity

Architecture Strategy
{external_knowledge}

Depth-Specific Synthesis Guidelines (Depth={depth})
1. Structural Synthesis (Depth 1-2):
    - Create hybrid control flow combining best elements from both versions
    - Example: "Combine Version A's iteration structure with Version B's termination
    conditions"
    - Forbid direct replication of either version's architecture

2. Implementation Fusion (Depth 3-4):
    - Develop novel parameter hybridization techniques
    - Example: "Blend Version A's exploration mechanism with Version B's exploitation
    strategy"
    - Require at least one innovative combination per functional module

3. Mathematical Innovation (Depth 5+):
    - Derive new computational operators through version synthesis
    - Example: "Fuse Version A's approximation method with Version B's error correction"
    - Mandate 10-20% computational complexity reduction
```

This template structure ensures that the crossover operation is not merely concatenative but involves intelligent analysis and strategic combination of algorithmic strengths.

Depth-Adaptive Synthesis Guidelines. The PC operation implements depth-specific strategies that adapt the crossover complexity based on the current position in the MCTS tree:

- Structural Synthesis (Depth 1-2):
  - Focuses on combining high-level architectural elements from both parent algorithms
  - Creates hybrid control flow structures that merge the best organizational patterns

  - Example directive: "Combine Version A's iteration structure with Version B's termination conditions"
  - Explicitly forbids direct replication of either parent's complete architecture

- Implementation Fusion (Depth 3-4):
  - Emphasizes parameter hybridization and functional module integration
  - Develops novel approaches to blend algorithmic strategies
  - Example directive: "Blend Version A's exploration mechanism with Version B's exploitation strategy"
  - Requires at least one innovative combination per functional module

- Mathematical Innovation (Depth 5+):
  - Derives new computational operators through sophisticated version synthesis
  - Focuses on mathematical justification for algorithmic improvements
  - Example directive: "Fuse Version A's approximation method with Version B's error correction"
  - Mandates 10-20% computational complexity reduction alongside performance gains

Innovation Requirements and Quality Assurance. The PC operation enforces strict innovation standards to ensure that generated algorithms represent genuine improvements:

- Core Innovation Targets:
  - Synthesize 3+ novel elements not present in either parent version
  - Resolve 2 fundamental limitations identified through comparative analysis
  - Introduce 1 breakthrough enhancement with rigorous mathematical justification
  - Demonstrate non-trivial performance gains over both parent algorithms
  - Prohibit direct replication of complete code blocks from either parent

Reflection Generation Process. Before conducting the crossover synthesis, the system generates analytical insights through a depth-adaptive reflection template:

````
## Task Objective
Analyze optimization patterns across algorithm versions and generate depth-specific
improvement strategies. Current MCTS Depth: depth/max_depth={depth}/{max_depth}

## Depth-Specific Focus
- Shallow (Depth 1-2): Structural patterns & control flow
- Medium (Depth 3-4): Implementation techniques & parameterization
- Deep (Depth 5+): Mathematical formulations & computational primitives

## Algorithm Comparison
- Original (Suboptimal)
```python
{code_worse}
```
````

```
- Improved (Optimized)
```python
{code_better}
```

Depth-Adaptive Analysis
1. Core Pattern Extraction
For {depth}-level analysis:
- Shallow: Compare control structures/algorithmic paradigms
- Medium: Analyze parameter configurations/function compositions
- Deep: Examine mathematical operators/numerical methods

2. Optimization Principle Generation
Generate 3-5 transferable rules that:
- Directly address {depth}-specific limitations
- Contain concrete parameter values from improved version
- Maintain functional equivalence

Output Format Rules
- Return optimization strategies within SINGLE BRACE
- Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
```

This reflection generation produces the `reflection` variable used in the main crossover template.

Reflection-Guided Analysis. The crossover process incorporates a reflection component that analyzes the strengths and weaknesses of both parent algorithms:

```
## Requirements
1. Core Innovation Targets:
    - Synthesize 3+ novel elements not present in either version
    - Resolve 2 fundamental limitations identified in analysis
    - Introduce 1 breakthrough enhancement with mathematical justification
    - Demonstrate non-trivial performance gain over both versions
    - Prohibit direct replication of complete code blocks
```

This reflection analysis is generated through the `deepseek-r1-250120` model and provides crucial insights that guide the synthesis process. The reflection identifies:

- Computational advantages in each parent algorithm
- Structural design patterns that contribute to performance
- Potential synergistic combinations that could yield emergent benefits
- Limitation patterns that should be addressed in the offspring

Output Format and Validation. The PC operation enforces a standardized output format that ensures both human readability and automated processing:

```
2. Output Format:
- Place the core design idea in a sentence within a brace BEFORE the function definition
- For the core design idea format: \\{{A hybrid gravitational wave detection pipeline...}}
- Implement as Python function: {func_name}
- Inputs: {input_count} parameter(s) ({joined_inputs})
- Outputs: {output_count} return value(s) ({joined_outputs})
- Follow: {inout_inf}
- Constraints: {other_inf}
- IMPORTANT: All output code MUST be valid Python syntax. Do not place description text
inside curly braces within the function body.
- Example of correct format:
    \\{{Core design description here}}
    ```python
```
```

```
    def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) ->
    tuple[np.ndarray, np.ndarray, np.ndarray]:
        """Core design description can alternatively be placed here as a docstring"""
        # Function implementation...
    ```
```

## 1.3 Sibling Crossover Implementation

The Sibling Crossover (SC) operation implements a sophisticated two-phase approach that leverages peer algorithm insights to generate improved offspring. Unlike Parent Crossover, which combines algorithms from different hierarchical levels, SC focuses on horizontal knowledge transfer between algorithms at the same MCTS depth, promoting diversity while maintaining comparable complexity levels.

**Two-Phase Architecture.** The SC operation employs a unique two-stage process: first generating optimization hints through multi-level reflection analysis (Phase 1), then implementing concrete algorithmic improvements based on these insights (Phase 2). This separation enables more targeted optimization by allowing the system to first identify improvement opportunities before implementing solutions.

**Phase 1: Multi-Level Reflection Analysis.** The first phase generates depth-specific optimization hints by synthesizing insights from sibling algorithms and parent-level analysis:

```
Task Overview
Generate depth-specific optimization hints for gravitational wave detection algorithms by
synthesizing multi-level reflections.
Current Optimization Depth: {parent_depth}/{max_depth} (shallow: structural patterns,
medium: implementation techniques, deep: mathematical details)

Contextual Insights
1. Peer Algorithm Reflections (Depth {parent_depth}):
    - Formatted as performance-annotated entries: [No.N Brother Reflection Score: X]<
    reflection>
    - Time-ordered weighting (newest=highest priority) with objective score-based ranking
    - Includes full technical post-mortems from immediate ancestors
{parent_reflections}

2. Father Algorithm Analysis (Depth {father_depth}):
{father_reflection}

Hint Generation Requirements
1. Produce 3-5 executable optimization directives that:
    - Integrate cross-depth insights from peer implementations
    - Target {parent_depth}-level (shallow: structural patterns, medium: implementation
    techniques, deep: mathematical details) components for improvement
    - Formulate mathematically sound enhancements
    - Align with gravitational wave data processing objectives

2. Output Format Rules
    - Return optimization strategies within SINGLE BRACE
    - Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
    - Focus on {parent_depth}-appropriate modifications
    - Emphasize time-domain processing optimizations

Critical Constraints
- Each directive must correspond to concrete code changes
- Explicitly connect to reflection insights where applicable
- Maintain strict {parent_depth}-level focus in all suggestions
```
```

```
- Exclude explanatory text within the hint brace
- Prioritize modifications matching current depth's optimization type
```

Sibling Selection and Weighting Strategy. The SC operation employs a sophisticated parent selection mechanism that prioritizes high-performing sibling algorithms:

```python
# Select parents based on objective value weights
other = [ind for ind in pop if ind['code'] != father['code']]
weights = [1.0 / (-ind['fitness'] + 1e-10) for ind in other]  # Lower objective = higher weight
normalized_weights = [w / sum(weights) for w in weights]
parents = random.choices(other, weights=normalized_weights, k=min(self.m, len(other)))
```

This weighting strategy ensures that successful algorithmic patterns from high-performing siblings are more likely to influence the offspring generation process.

Depth-Adaptive Optimization Focus. The first phase implements depth-specific optimization strategies that adapt to the current position in the MCTS tree:

- Shallow Depth (1-2): Focuses on structural patterns and control flow restructuring
- Medium Depth (3-4): Emphasizes implementation techniques and numerical optimizations
- Deep Depth (5+): Concentrates on mathematical details and advanced computational methods

Phase 2: Concrete Algorithm Implementation. The second phase transforms the optimization hints into executable algorithms:

```
## Algorithm Optimization Task
Develop an enhanced gravitational signal processing algorithm for interferometer data
analysis by implementing concrete improvements from multi-level code analysis.

## Technical Context
1. Optimization Depth Specifications:
- Current Focus Level: {depth} (max_depth={max_depth})
    (1-2: Control flow restructuring, 3-4: Numerical computation optimizations, 5+:
    Advanced linear algebra methods)
- Code Analysis Insights from Prior Level:
```text
{reflection}
```

2. Base Implementation Details:
[Functional Purpose] {algorithm_description}
[Core Implementation]
```python
{algorithm_code}
```

Implementation Directives (Depth {depth}):
- Shallow (1-2): Restructure control flow using reflection suggestion (e.g., split data
conditioning/analysis phases)
- Medium (3-4): Apply numerical optimizations from reflection (e.g., FFT window size
optimization)
- Deep (5+): Implement matrix computation improvements from reflection (e.g., regularized
inverse covariance)

{external_knowledge}
```

```
## Output Format
- Place the core design idea in a sentence within a brace BEFORE the function definition
- For the core design idea format: \\{{A hybrid gravitational wave detection pipeline...}}
- Implement as Python function: {func_name}
- Inputs: {input_count} parameter(s) ({joined_inputs})
- Outputs: {output_count} return value(s) ({joined_outputs})
- Follow: {inout_inf}
- Constraints: {other_inf}
- IMPORTANT: All output code MUST be valid Python syntax. Do not place description text
inside curly braces within the function body.
- Example of correct format:
    \\{{Core design description here}}
    ```python
    def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) ->
    tuple[np.ndarray, np.ndarray, np.ndarray]:
        """Core design description can alternatively be placed here as a docstring"""
        # Function implementation...
    ```

Important Notes
- Focus on algorithmic improvements rather than code style changes
- Ensure the new implementation directly addresses the reflection insights
```

Reflection Processing and Template Variables. The SC operation processes multiple sources of algorithmic insight through structured template variables:

- `parent_reflections`: Performance-annotated reflections from peer algorithms, formatted as ranked entries with objective scores
- `father_reflection`: Analysis from the immediate parent algorithm at depth-1
- `reflection`: Synthesized optimization hints generated in Phase 1
- `algorithm_description`: Functional description of the base algorithm
- `algorithm_code`: Complete implementation of the parent algorithm

Quality Assurance and Validation. The two-phase approach enables comprehensive quality control:

- Phase 1 Validation:
  - Ensures reflection insights are depth-appropriate
  - Validates mathematical soundness of optimization suggestions
  - Confirms alignment with gravitational wave processing objectives
- Phase 2 Validation:
  - Verifies syntactic correctness of generated code
  - Confirms interface compliance with standardized function signatures
  - Tests algorithmic improvements against reflection insights
  - Validates computational efficiency claims

Temporal Weighting and Performance Ranking. The SC operation implements sophisticated temporal weighting that prioritizes recent algorithmic discoveries while maintaining objective score-based ranking:

- Time-ordered weighting: Newer algorithms receive higher priority in the reflection synthesis

- Performance-based ranking: Algorithms with better objective scores contribute more heavily to the optimization hints
- Cross-depth integration: Insights from both peer algorithms and parent-level analysis are systematically combined

This comprehensive approach ensures that SC operations generate algorithms that not only improve upon their immediate ancestors but also incorporate the collective intelligence of high-performing siblings, leading to more robust and efficient gravitational wave detection strategies.

1.4 Point Mutation Implementation

Point Mutation (PM) operations introduce targeted modifications to individual algorithms based on performance analysis, implementing two distinct approaches that offer different levels of sophistication and computational investment. The framework provides both single-stage direct improvement and two-stage reflection-driven enhancement strategies.

Single-Stage Point Mutation: Direct Algorithm Improvement. The operation implements a straightforward approach that directly compares an original algorithm with a high-performing elite algorithm to generate improvements. This method prioritizes computational efficiency while maintaining effective algorithmic enhancement.

Template Structure:

````
## Task Overview
You will analyze an original algorithm, an improved version of it, and create a new
enhanced algorithm. Below are the key components:

## Algorithm Details
1. ORIGINAL ALGORITHM:
    - Description: {original_algorithm_description}
    - Code:
```python
{original_algorithm_code}
```
    - **Objective Value**: {original_objective_value}

2. BETTER ALGORITHM (Reference Implementation):
    - Description: {better_algorithm_description}
    - Code:
```python
{better_algorithm_code}
```
    - **Objective Value**: {better_objective_value}
    - Improvement Insights:
```text
{better_algorithm_reflection}
```

## Implementation Requirements
1. Analyze the differences between the original and better algorithms
2. Create a new algorithm that:
    - Incorporates successful elements from the better algorithm
    - Addresses limitations revealed in the improvement insights
    - Produces better results than the original algorithm
3. Output format requirements:
    - Place the core design idea in a sentence within a brace BEFORE the function
    definition
````

```
    - For the core design idea format: \\{{A hybrid gravitational wave detection pipeline
    ...}}
    - Implement as Python function: {func_name}
    - Inputs: {input_count} parameter(s) ({joined_inputs})
    - Outputs: {output_count} return value(s) ({joined_outputs})
    - Follow: {inout_inf}
    - Constraints: {other_inf}
    - IMPORTANT: All output code MUST be valid Python syntax. Do not place description text
     inside curly braces within the function body.
    - Example of correct format:
        \\{{Core design description here}}
        ```python
        def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) ->
         tuple[np.ndarray, np.ndarray, np.ndarray]:
            """Core design description can alternatively be placed here as a docstring"""
            # Function implementation...
        ```
{external_knowledge}

Important Notes
- Focus on algorithmic improvements rather than code style changes
- Ensure the new implementation directly addresses the reflection insights
```

Two-Stage Point Mutation: Reflection-Driven Enhancement. The operation implements a sophisticated two-phase approach that mirrors the sibling crossover methodology but focuses on individual algorithm improvement rather than horizontal knowledge transfer.

Phase 1: Strategic Reflection Generation. The first phase synthesizes insights from multiple sources to generate comprehensive optimization guidelines:

```
## Task Overview
Generate optimized technical guidelines for gravitational wave detection algorithms through
 systematic analysis of multi-generational reflection insights. Focus on enhancing data
conditioning pipelines, time-frequency analysis methods, noise suppression techniques, and
H1-L1 detector coherence optimization. Produce executable directives addressing: waveform
recognition precision, computational complexity management, and non-stationary noise
differentiation while maintaining strict API compliance.

## Input Context
1. NEW INSIGHTS FROM RECENT ITERATIONS:
    - Formatted as performance-annotated entries: [Parent N Reflection  Score: X]<
    reflection>
    - Time-ordered weighting (newest=highest priority) with objective score-based ranking
    - Includes full technical post-mortems from immediate ancestors
{parent_reflections}

2. LONG-TERM REFLECTION REPOSITORY:
    - Contains battle-tested insights from top 1% performers
    - 3x weighting factor for architectural-level insights
    - Curated through 3-stage filtration:
        1. Statistical significance validation
        2. Cross-generational effectiveness verification
        3. Compatibility check with current detector configurations
{elite_reflection}

## Implementation Requirements
1. Perform weighted synthesis of reflections
2. Generate 3-5 technically-grounded optimization directives
3. Prioritize:
    - Mitigation of historical implementation flaws
    - Amplification of proven effective patterns
    - Weighted integration of multi-generational insights

## Output Format
```

```
- Return all guidelines within SINGLE BRACE
- Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
- Concrete technical directives only
- No explanatory text or formatting
```

Phase 2: Concrete Algorithm Implementation. The second phase transforms the strategic insights into executable algorithmic improvements:

```
## Task Overview
Leverage insights from prior strategic reflection to architecturally enhance the
gravitational wave detection algorithm. Develop improvements that directly address
identified limitations in CRITICAL REFLECTION INSIGHTS while preserving core functionality
through:

1. Stage-level architectural modifications informed by reflection analysis
2. Reflection-driven noise reduction and coherence enhancement strategies
3. Time-frequency analysis variations targeting specific weaknesses identified
4. H1-L1 synthesis improvements based on cross-detector insights

Generate architecturally distinct variants that implement reflection-derived concepts
through fundamental structural changes.

## Input Context
1. CRITICAL REFLECTION INSIGHTS (Improvement Basis):
```text
{reflection}
```

2. REFERENCE IMPLEMENTATION:
[Description] {elite_algorithm_description}
[Baseline Code]
```python
{elite_algorithm_code}
```

Implementation Requirements
1. Execute reflection-guided analysis:
    - Map reflection insights to specific code components
    - Identify 2-3 architectural limitations in current implementation
2. Propose improvements that directly convert reflection insights into:
    - Enhanced signal path architecture
    - Novel noise handling structures
    - Optimized computational patterns
    - Advanced detector synergy mechanisms
3. Maintain strict interface compatibility with existing system integration

{external_knowledge}

Output Format
- Place the core design idea in a sentence within a brace BEFORE the function definition
- For the core design idea format: \\{{A hybrid gravitational wave detection pipeline...}}
- Implement as Python function: {func_name}
- Inputs: {input_count} parameter(s) ({joined_inputs})
- Outputs: {output_count} return value(s) ({joined_outputs})
- Follow: {inout_inf}
- Constraints: {other_inf}
- IMPORTANT: All output code MUST be valid Python syntax. Do not place description text
inside curly braces within the function body.
- Example of correct format:
    \\{{Core design description here}}
    ```python
    def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) ->
    tuple[np.ndarray, np.ndarray, np.ndarray]:
        """Core design description can alternatively be placed here as a docstring"""
        # Function implementation...
    ```

Important Notes
```

```
- Focus on algorithmic improvements rather than code style changes
- Ensure the new implementation directly addresses the reflection insights
```

Selection Strategies and Elite Integration. Both PM operations leverage the elite offspring as a performance benchmark and source of successful algorithmic patterns. The key distinction lies in their selection strategies:

- Single-Stage Selection Strategy:
  - Selects a single parent algorithm from the population (excluding the elite)
  - Directly compares parent performance with elite offspring
  - Implements immediate improvement through direct analysis
- Two-Stage Selection Strategy:
  - Selects multiple parent algorithms for comprehensive reflection analysis
  - Incorporates both recent algorithmic insights and long-term elite patterns
  - Implements sophisticated multi-generational knowledge synthesis

Computational Efficiency Considerations. The two PM approaches offer different computational trade-offs:

- Single-Stage Advantages:
  - Single-stage processing reduces computational overhead
  - Direct comparison enables rapid algorithm improvement
  - Simplified prompting reduces LLM token consumption
- Two-Stage Advantages:
  - Two-stage processing enables more sophisticated optimization
  - Multi-generational insight integration leads to more robust improvements
  - Reflection-driven approach produces more interpretable algorithmic modifications

Template Variable Integration. Both PM operations incorporate comprehensive template variables that enable systematic algorithmic improvement:

- Common Variables:
  - `func_name`, `input_count`, `output_count`: Interface specification
  - `joined_inputs`, `joined_outputs`: Parameter documentation
  - `better_algorithm_description`, `better_algorithm_code`: Elite algorithm details
  - `original_objective_value`, `better_objective_value`: Performance metrics
  - `better_algorithm_reflection`: Elite algorithm insights
- Two-Stage Variables:
  - `parent_reflections`: Multi-parent reflection synthesis
  - `elite_reflection`: Long-term elite insights

– `reflection`: Generated optimization guidelines

Quality Assurance and Validation. Both PM operations implement rigorous validation procedures:

- Single-Stage Validation:
  - Direct performance comparison with both parent and elite algorithms
  - Verification of improvement insight integration
  - Confirmation of interface compliance
- Two-Stage Validation:
  - Two-stage validation covering both reflection generation and implementation
  - Cross-generational consistency checking
  - Architectural improvement verification

The dual PM approach provides flexibility in algorithmic improvement strategies, enabling the framework to adapt to different optimization scenarios while maintaining consistent quality standards and interface compliance.

1.5 Path-wise Crossover Implementation

Path-wise Crossover (PWC) operations synthesize information along complete root-to-leaf trajectories in the MCTS tree, capturing long-range dependencies and enabling global optimization strategies. The framework implements two distinct PWC approaches that differ in their analytical methodologies: reflection-based synthesis and comprehensive algorithm analysis.

Reflection-Based Path-wise Crossover: Multi-Algorithm Insight Synthesis. The operation implements a two-stage process that analyzes reflection patterns across multiple algorithms in a complete MCTS path to identify generalizable optimization principles.

Phase 1: Cross-Algorithm Pattern Analysis. The first phase extracts recurring technical strategies from multiple algorithm reflections:

```
## Task Overview
Analyze and synthesize technical reflections from multiple algorithm iterations to identify
 cross-algorithm optimization patterns and guide next-generation algorithm design.
Prioritize extraction of generalizable technical principles over implementation-specific
details.
Current Optimization Depth: depth/max_depth={depth}/{max_depth} (shallow: structural
patterns, medium: implementation techniques, deep: mathematical details)

## Input Context
Analyzing {num_algorithms} algorithm reflections from MCTS exploration trajectories.
Technical reflections follow depth-specific analysis requirements. Structural format: [No.N
 algorithm's reflection (depth: X)]<reflection>
{algorithm_reflections}

## Reflection Requirements
1. **Pattern Identification** (Key Observed Patterns):
    - Extract 2-3 recurring technical strategies (e.g. "Multi-scale wavelet decomposition"
    not "used Morlet wavelet")
    - Categorize by analysis level:
        * Structural: Component architecture (e.g. "Parallel filter banks")
        * Implementation: Algorithmic choices (e.g. "Adaptive thresholding")
```

```
       * Mathematical: Core transforms (e.g. "Orthogonal matching pursuit")

2. **Technical Pathway Analysis** (Promising Technical Pathways):
    - Identify under-utilized but theoretically sound approaches (e.g. "Sparse
    representation in frequency domain")
    - Specify required technical components without code details (e.g. "Requires:
    Overcomplete basis construction")

3. **Optimization Principles** (Strategic Optimization Principles):
    - Formulate depth-specific guidelines (e.g. "At mathematical level: Maximize time-
    frequency product $\leq$ 0.5")
    - Relate physical constraints to algorithmic parameters (e.g. "Wavelet duration should
    match typical glitch durations")

4. **Specificity Balance**:
    - Technical specificity: Name mathematical concepts (e.g. "Gabor uncertainty") and
    signal processing domains
    - Implementation avoidance: Omit code structures (e.g. "Avoid: 'Use 3 nested loops'")

## Output Format Rules
- Return optimization strategies within SINGLE BRACE
- Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
- Do not include markdown formatting or additional explanations
```

Phase 2: Algorithm Implementation. The second phase transforms the synthesized insights into concrete algorithmic improvements:

```
## Task Overview
Develop an enhanced gravitational wave detection algorithm through targeted modifications
addressing specific technical shortcomings identified in the reflection analysis.

## Input Context
[Critical Reflection Insights]
```text
{reflection}
```

[Baseline Implementation]
[Functional Description] {algorithm_description}
[Current Codebase]
```python
{algorithm_code}
```

{external_knowledge}

Output Format
- Place the core design idea in a sentence within a brace BEFORE the function definition
- For the core design idea format: \\{{A hybrid gravitational wave detection pipeline...}}
- Implement as Python function: {func_name}
- Inputs: {input_count} parameter(s) ({joined_inputs})
- Outputs: {output_count} return value(s) ({joined_outputs})
- Follow: {inout_inf}
- Constraints: {other_inf}
- IMPORTANT: All output code MUST be valid Python syntax. Do not place description text
inside curly braces within the function body.
- Example of correct format:
  \\{{Core design description here}}
  ```python
  def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) -> tuple
  [np.ndarray, np.ndarray, np.ndarray]:
      """Core design description can alternatively be placed here as a docstring"""
      # Function implementation...
  ```

Important Notes
- Focus on algorithmic improvements rather than code style changes
- Ensure the new implementation directly addresses the reflection insights
```

Comprehensive Algorithm Analysis Path-wise Crossover: Multi-Level Technical Synthesis. The operation implements a more sophisticated analytical approach that examines complete algorithm implementations across different depth levels.

Phase 1: Multi-Level Technical Analysis. The first phase conducts comprehensive analysis of algorithms along the complete MCTS path:

```
## Task Objective
Synthesize technical insights from algorithm evolution MCTS path to guide targeted
improvements. Current Analysis Level: depth/max_depth={depth}/{max_depth} (1-2: structural,
 3-4: implementation, 5+: mathematical)

## Depth-Specific Focus
- Shallow (Depth 1-2): Structural patterns & control flow
- Medium (Depth 3-4): Implementation techniques & parameterization
- Deep (Depth 5+): Mathematical formulations & computational primitives

## Input Context
Analyzing {num_algorithms} algorithm reflections from MCTS exploration trajectories.
Technical reflections follow depth-specific analysis requirements. Structural format: [No.N
 algorithm's reflection (depth: X)]<description><objective><code>
{parent_info}

## Synthesis Process
1. Cross-Level Insight Integration:
    - Identify key recurring technical strategies across abstraction levels
    - Note level-specific constraints affecting current implementations

2. Domain Compliance Verification:
    - Validate approaches against gravitational wave signal characteristics
    - Check numerical reliability across different implementation levels

3. Improvement Planning:
    - Structural: Adjust data processing pipelines
    - Implementation: Optimize critical parameter relationships
    - Mathematical: Enhance core transformation components

## Technical Workflow
### 1. Multi-Level Technical Analysis
Structural -> Compare module composition and interaction patterns
Implementation -> Assess parameter sensitivity and adaptation logic
Mathematical -> Examine transformation kernels and precision handling

### 2. Level-Appropriate Optimization
For current depth={depth}:
    - Select 2-4 improvement focus areas with technical rationale
    - Define implementation requirements for each focus area
    - Establish verification criteria with domain constraints

## Output Format Rules
- Return optimization strategies within SINGLE BRACE
- Ensure entire response can be parseable by regex: \\{{(.*?)\\}} with DOTALL flag
- Do not include markdown formatting or additional explanations
```

Phase 2: Algorithm Implementation. The operation shares the same implementation phase as the reflection-based path-wise crossover, utilizing the reflection-based PWC template for consistent output formatting and algorithmic generation.

Methodological Distinctions. The key differences between the reflection-based path-wise crossover and the comprehensive algorithm analysis PWC lie in their analytical strategies:

- Reflection-Based PWC:

  - Focuses on synthesizing existing reflection insights from multiple algorithms
  - Emphasizes pattern recognition across previously analyzed algorithmic behaviors
  - Prioritizes extraction of generalizable technical principles over implementation details
  - Categorizes insights by structural, implementation, and mathematical analysis levels

- Comprehensive Algorithm Analysis PWC:
  - Conducts direct analysis of complete algorithm implementations
  - Examines algorithmic components across multiple depth levels simultaneously
  - Integrates cross-level insights through systematic technical workflow
  - Emphasizes domain compliance verification and improvement planning

Depth-Adaptive Processing. Both PWC operations implement depth-specific analysis strategies that adapt to the current position in the MCTS tree:

- Shallow Depth Focus (1-2):
  - Structural patterns and component architecture analysis
  - Control flow restructuring and module composition optimization
  - Data processing pipeline adjustments

- Medium Depth Focus (3-4):
  - Implementation techniques and algorithmic parameter optimization
  - Critical parameter relationship assessment
  - Numerical computation enhancement strategies

- Deep Depth Focus (5+):
  - Mathematical formulation analysis and computational primitive optimization
  - Transformation kernel examination and precision handling
  - Advanced linear algebra method integration

Path Trajectory Analysis. Both operations process algorithms along complete MCTS paths, with depth tracking that enables comprehensive evolutionary analysis:

- Reflection-Based PWC Path Processing:
  - Analyzes reflection patterns from algorithms at decreasing depth levels
  - Tracks depth-specific insights through structured format annotations
  - Synthesizes cross-depth technical strategies for optimization guidance

- Comprehensive Algorithm Analysis PWC Path Processing:

- Examines complete algorithm implementations with performance metrics
- Integrates algorithmic descriptions, objective values, and code analysis
- Conducts multi-level technical synthesis across the entire path trajectory

Template Variable Integration. Both PWC operations incorporate sophisticated template variables that enable comprehensive path analysis:

- Common Variables:
  - `depth`, `max_depth`: Depth-specific processing parameters
  - `num_algorithms`: Path length and analysis scope
  - `func_name`, `input_count`, `output_count`: Interface specifications
  - `external_knowledge`: Domain knowledge integration
- Reflection-Based PWC Variables:
  - `algorithm_reflections`: Multi-algorithm reflection synthesis
  - `reflection`: Generated optimization insights
- Comprehensive Algorithm Analysis PWC Variables:
  - `parent_info`: Complete algorithm implementation details
  - `current_algorithm_description`, `current_algorithm_code`: Baseline algorithm specifications
  - `current_objective_value`: Performance reference metrics

Quality Assurance and Validation. Both PWC operations implement comprehensive validation procedures:

- Reflection-Based PWC Validation:
  - Pattern identification verification across multiple algorithm reflections
  - Technical pathway analysis consistency checking
  - Optimization principle formulation validation
- Comprehensive Algorithm Analysis PWC Validation:
  - Multi-level technical analysis coherence verification
  - Domain compliance checking across different implementation levels
  - Cross-level insight integration validation

The dual PWC approach provides complementary strategies for capturing long-range dependencies in the MCTS tree, enabling the framework to synthesize insights across complete evolutionary trajectories while maintaining depth-specific optimization focus and domain knowledge integration.

1.6 Domain Knowledge Integration

Domain knowledge integration serves as a critical component that ensures generated algorithms remain grounded in gravitational wave detection principles while encouraging exploration beyond traditional linear processing methods. The framework incorporates specialized domain expertise through structured

knowledge templates that guide algorithmic development toward physically meaningful and computationally efficient solutions.

External Knowledge Template Structure. The domain knowledge integration employs a comprehensive template that emphasizes non-linear processing approaches and adaptive algorithmic strategies:

```
### External Knowledge Integration
1. **Non-linear** Processing Core Concepts:
    - Signal Transformation:
        * Non-linear vs linear decomposition
        * Adaptive threshold mechanisms
        * Multi-scale analysis

    - Feature Extraction:
        * Phase space reconstruction
        * Topological data analysis
        * Wavelet-based detection

    - Statistical Analysis:
        * Robust estimators
        * Non-Gaussian processes
        * Higher-order statistics

2. Implementation Principles:
    - Prioritize adaptive over fixed parameters
    - Consider local vs global characteristics
    - Balance computational cost with accuracy
```

Non-linear Processing Emphasis. The domain knowledge framework explicitly prioritizes non-linear algorithmic approaches over traditional linear methods, recognizing that gravitational wave signals exhibit complex, transient characteristics that require sophisticated analysis techniques. This emphasis addresses fundamental limitations in conventional matched filtering approaches that rely heavily on linear processing assumptions.

Signal Transformation Guidance. The domain knowledge provides specific guidance on signal transformation strategies that leverage advanced signal processing concepts:

- Non-linear vs Linear Decomposition: The framework encourages exploration of non-linear decomposition methods that can capture complex signal morphologies beyond the capabilities of traditional Fourier-based approaches. This includes techniques such as empirical mode decomposition, intrinsic mode functions, and adaptive basis construction.
- Adaptive Threshold Mechanisms: Rather than employing fixed threshold values, the domain knowledge promotes adaptive thresholding strategies that respond to local signal characteristics and noise conditions. This approach enables more robust detection performance across diverse observational scenarios.
- Multi-scale Analysis: The framework emphasizes multi-scale signal analysis techniques that can simultaneously capture both short-duration transient signals and longer-duration continuous wave sources. This includes wavelet-based methods, time-frequency analysis, and hierarchical decomposition strategies.

- Feature Extraction Methodologies. The domain knowledge incorporates advanced feature extraction approaches that extend beyond traditional signal processing paradigms:
  - Phase Space Reconstruction: The framework encourages exploration of phase space reconstruction techniques that can reveal hidden dynamical structures in gravitational wave data. This includes embedding dimension analysis, recurrence analysis, and attractor reconstruction methods.
  - Topological Data Analysis: The domain knowledge promotes topological data analysis approaches that can identify persistent features and structural patterns in high-dimensional gravitational wave data. This includes persistent homology, Mapper algorithms, and topological feature extraction.
  - Wavelet-based Detection: The framework emphasizes wavelet-based detection strategies that can provide optimal time-frequency resolution for transient signal analysis. This includes continuous wavelet transforms, discrete wavelet decomposition, and wavelet packet analysis.
- Statistical Analysis Enhancement. The domain knowledge integrates sophisticated statistical analysis techniques that account for the complex noise characteristics of gravitational wave detectors:
  - Robust Estimators: The framework promotes robust statistical estimators that can maintain performance in the presence of outliers and non-Gaussian noise distributions. This includes median-based estimators, M-estimators, and trimmed mean approaches.
  - Non-Gaussian Processes: The domain knowledge emphasizes analysis techniques that can handle non-Gaussian noise processes commonly encountered in gravitational wave data. This includes heavy-tailed distributions, skewed probability models, and non-stationary noise characterization.
  - Higher-order Statistics: The framework encourages exploration of higher-order statistical moments and cumulants that can capture subtle signal characteristics beyond second-order analysis. This includes bispectrum analysis, higher-order moment estimation, and polyspectral techniques.
- Implementation Principles and Constraints. The domain knowledge provides specific implementation principles that guide algorithmic development toward practical and efficient solutions:
  - Adaptive Parameter Prioritization: The framework emphasizes adaptive parameter selection over fixed parameter values, enabling algorithms to respond dynamically to changing signal and noise conditions. This principle encourages exploration of learning-based parameter adjustment, feedback control mechanisms, and online adaptation strategies.
  - Local vs Global Characteristics: The domain knowledge promotes consideration of both local signal characteristics and global data patterns,

enabling algorithms to balance fine-grained analysis with comprehensive signal understanding. This includes local stationarity analysis, global trend estimation, and multi-resolution processing approaches.
- Computational Cost-Accuracy Balance: The framework provides guidance on balancing computational efficiency with detection accuracy, ensuring that generated algorithms remain practical for real-time implementation while maintaining scientific rigor. This includes complexity analysis, algorithmic optimization, and performance benchmarking considerations.

Integration Across Evolutionary Operations. The domain knowledge template is systematically integrated across all evolutionary operations (PC, SC, PM, PWC) through the external_knowledge template variable. This ensures consistent application of gravitational wave detection principles regardless of the specific evolutionary strategy employed.

Physical Validity Assurance. The domain knowledge template ensures that all generated algorithms respect fundamental physical constraints related to gravitational wave signal characteristics, detector limitations, and noise properties.

Computational Feasibility. The implementation principles guide algorithmic development toward computationally feasible solutions that can be practically implemented within the constraints of current computational resources and real-time processing requirements.

This comprehensive domain knowledge integration creates a robust framework for scientifically grounded algorithmic discovery, ensuring that the evolutionary process generates algorithms that are both innovative and practically applicable to gravitational wave detection challenges.

1.7 Error Handling and Iterative Refinement

The Evo-MCTS framework incorporates a robust error handling mechanism that enables iterative refinement of generated algorithms through automated debugging and correction processes. When generated code encounters execution errors, fails to detect signals, or exceeds computational time limits, the system employs a rechat strategy using advanced reasoning models to diagnose and resolve issues.

Error Detection and Classification. The framework monitors three primary failure modes during algorithm execution: (i) runtime exceptions and syntax errors that prevent code execution, (ii) algorithmic failures where no gravitational wave signals are detected despite their presence in the data, and (iii) computational timeout scenarios where algorithms exceed predefined execution limits. Each failure mode triggers specific diagnostic protocols tailored to the underlying issue type.

Iterative Refinement Protocol. Upon error detection, the system implements a structured refinement process through a carefully designed prompt content structure. The system constructs conversation messages in a specific format to facilitate effective error correction:

Initially, when no system content is provided, the framework creates a message list containing a single user role entry with the original prompt content (`prompt_content`): `messages = [{"role": "user", "content": prompt_content}]`

When a rechat response is available (indicating a previous failed attempt), the system extends the conversation by first appending the assistant's previous response (`rechat_response`): `messages.append({"role": "assistant", "content": rechat_response})`

Subsequently, the system adds a new user message that explicitly requests debugging and issue resolution (`prompt_content`): `messages.append({"role": "user", "content": "Your previous code had execution errors, couldn't find signals, or timed out. Please debug and fix the issues:\n\n" + prompt_content})`

This structured approach maintains the conversational context while providing clear guidance for error correction, ensuring that the assistant understands both the original requirements and the specific issues that need to be addressed.

Automated Debugging Integration. The error handling system leverages advanced reasoning capabilities to analyze failed algorithms and propose targeted corrections. This approach maintains the evolutionary optimization trajectory while addressing immediate technical obstacles that could otherwise terminate the search process. The iterative refinement ensures that promising algorithmic concepts are not discarded due to implementation errors, instead receiving corrective guidance to achieve functional implementations.

1.8 Post-Generation Analysis and Knowledge Extraction

The post-generation analysis phase extracts interpretable insights from evolved algorithms through automated knowledge distillation. This process transforms the raw algorithmic implementations into concise, human-readable descriptions that capture the essential design principles and operational characteristics of discovered solutions.

Algorithm Description Generation. The framework employs a structured prompt template to generate concise algorithm descriptions that highlight critical design decisions and implementation strategies. The prompt construction follows a systematic format:

```
Following is the Design Idea of a heuristic algorithm for the problem and the code with
function name 'pipeline_v2' for implementing the heuristic algorithm.
{prompt_inout_inf} {prompt_other_inf}
Design Idea:
{algorithm}

Code:
```python
{code}
```
The content of the Design Idea idea cannot fully represent what the algorithm has done
informative. So, now you should re-describe the algorithm using less than 3 sentences.
```

```
Hint: You should reference the given Design Idea and highlight the most critical design
ideas of the code. You can analyse the code to describe which variables are given higher
priorities and which variables are given lower priorities, the parameters and the structure
 of the code.
```

This template systematically combines the original design concept with the implemented code, requesting a refined description that captures the algorithm's core operational principles. The analysis focuses on parameter prioritization, structural characteristics, and critical design decisions that distinguish the evolved solution.

Knowledge Extraction Protocol. The post-generation analysis captures key design principles and compresses algorithmic representations into human-readable summaries. This reflection process identifies algorithmic innovations, signal processing techniques, and computational characteristics while reducing token consumption to prevent context window overflow in subsequent LLM interactions.

Interpretability Enhancement. The generated descriptions provide concise algorithmic summaries that enable efficient reference to previous discoveries without overwhelming the LLM context, facilitating continued exploration while maintaining algorithmic memory across generations.

1.9 Code Examples and Case Studies

This section presents a detailed examination of the highest-performing algorithm discovered during the Evo-MCTS optimization process, corresponding to node 486 (as shown in Figure 5a in the main text) which achieved the maximum fitness score of 5,241.37 units. The algorithm demonstrates sophisticated multi-stage signal processing techniques that emerged through evolutionary optimization.

Algorithm Overview. The evolved algorithm implements a four-stage pipeline combining robust baseline detrending, adaptive whitening with enhanced power spectral density (PSD) smoothing, coherent time-frequency analysis with frequency-conditioned regularization, and multi-resolution thresholding with octave-spaced dyadic wavelet validation. This architecture represents a novel synthesis of classical signal processing techniques with adaptive parameter selection mechanisms.

Stage 1: Robust Baseline Detrending. The algorithm initiates with median filtering-based detrending to remove long-term instrumental drifts and environmental variations. The median filter kernel size of 101 samples provides robust trend removal while preserving transient gravitational wave signatures. This preprocessing stage establishes a stable baseline for subsequent whitening operations.

Stage 2: Adaptive Whitening with Enhanced PSD Smoothing. The core innovation lies in the adaptive whitening mechanism that dynamically adjusts window parameters based on data characteristics. The algorithm implements Tukey windowing with 75% overlap and adaptive segment lengths constrained between 5-30 seconds, optimizing spectral estimation for varying noise conditions. The PSD smoothing employs exponential filtering with

stationarity-dependent coefficients (0.75-0.85 range), while Tikhonov regularization provides frequency-dependent gain control. Savitzky-Golay filtering generates causal-like gradients, and sigmoid-based nonlinear scaling enhances spectral features through adaptive gain factors.

Stage 3: Coherent Time-Frequency Analysis. The algorithm computes complex spectrograms preserving phase information across both detectors, enabling coherent analysis of gravitational wave signatures. Phase difference calculations and coherence estimation provide cross-detector validation, while frequency-conditioned regularization balances phase alignment with noise characteristics. The integration of axial curvature estimates through second derivatives and nonlinear activation functions (tanh-based boost) enhances signal discrimination capabilities.

Stage 4: Multi-Resolution Validation. The final stage implements sophisticated peak detection using robust statistical measures (median absolute deviation) combined with octave-spaced dyadic wavelet validation. Continuous wavelet transform coefficients across scales 1-8 provide multi-resolution signal verification, while Gaussian-weighted uncertainty estimation quantifies detection confidence intervals.

```python
import numpy as np
import scipy.signal as signal
from scipy.signal.windows import tukey
from scipy.signal import savgol_filter

def pipeline_v2(strain_h1: np.ndarray, strain_l1: np.ndarray, times: np.ndarray) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    """
    The pipeline function processes gravitational wave data from the H1 and L1 detectors to identify potential gravitational wave signals.
    It takes strain_h1 and strain_l1 numpy arrays containing detector data, and times array with corresponding time points.
    The function returns a tuple of three numpy arrays: peak_times containing GPS times of identified events,
    peak_heights with significance values of each peak, and peak_deltat showing time window uncertainty for each peak.
    """
    eps = np.finfo(float).tiny
    dt = times[1] - times[0]
    fs = 1.0 / dt
    # Base spectrogram parameters
    base_nperseg = 256
    base_noverlap = base_nperseg // 2
    medfilt_kernel = 101       # odd kernel size for robust detrending
    uncertainty_window = 5     # half-window for local timing uncertainty

    # -------------------- Stage 1: Robust Baseline Detrending --------------------
    # Remove long-term trends using a median filter for each channel.
    detrended_h1 = strain_h1 - signal.medfilt(strain_h1, kernel_size=medfilt_kernel)
    detrended_l1 = strain_l1 - signal.medfilt(strain_l1, kernel_size=medfilt_kernel)

    # -------------------- Stage 2: Adaptive Whitening with Enhanced PSD Smoothing --------------------
    def adaptive_whitening(strain: np.ndarray) -> np.ndarray:
        # Center the signal.
        centered = strain - np.mean(strain)
        n_samples = len(centered)
        # Adaptive window length: between 5 and 30 seconds
        win_length_sec = np.clip(n_samples / fs / 20, 5, 30)
        nperseg_adapt = int(win_length_sec * fs)
        nperseg_adapt = max(10, min(nperseg_adapt, n_samples))

        # Create a Tukey window with 75% overlap.
        tukey_alpha = 0.25
        win = tukey(nperseg_adapt, alpha=tukey_alpha)
        noverlap_adapt = int(nperseg_adapt * 0.75)
        if noverlap_adapt >= nperseg_adapt:
            noverlap_adapt = nperseg_adapt - 1

        # Estimate the power spectral density (PSD) using Welch's method.
```

```
        freqs, psd = signal.welch(centered, fs=fs, nperseg=nperseg_adapt,
                                  noverlap=noverlap_adapt, window=win, detrend='constant')
        psd = np.maximum(psd, eps)

        # Compute relative differences for PSD stationarity measure.
        diff_arr = np.abs(np.diff(psd)) / (psd[:-1] + eps)
        # Smooth the derivative with a moving average.
        if len(diff_arr) >= 3:
            smooth_diff = np.convolve(diff_arr, np.ones(3)/3, mode='same')
        else:
            smooth_diff = diff_arr

        # Exponential smoothing (Kalman-like) with adaptive alpha using PSD stationarity.
        smoothed_psd = np.copy(psd)
        for i in range(1, len(psd)):
            # Adaptive smoothing coefficient: base 0.8 modified by local stationarity (±0.05)
            local_alpha = np.clip(0.8 - 0.05 * smooth_diff[min(i-1, len(smooth_diff)-1)], 0.75, 0.85)
            smoothed_psd[i] = local_alpha * smoothed_psd[i-1] + (1 - local_alpha) * psd[i]

        # Compute Tikhonov regularization gain based on deviation from median PSD.
        noise_baseline = np.median(smoothed_psd)
        raw_gain = (smoothed_psd / (noise_baseline + eps)) - 1.0

        # Compute a causal-like gradient using the Savitzky-Golay filter.
        win_len = 11 if len(smoothed_psd) >= 11 else ((len(smoothed_psd)//2)*2+1)
        polyorder = 2 if win_len > 2 else 1
        delta_freq = np.mean(np.diff(freqs))
        grad_psd = savgol_filter(smoothed_psd, win_len, polyorder, deriv=1, delta=delta_freq, mode='interp')

        # Nonlinear scaling via sigmoid to enhance gradient differences.
        sigmoid = lambda x: 1.0 / (1.0 + np.exp(-x))
        scaling_factor = 1.0 + 2.0 * sigmoid(np.abs(grad_psd) / (np.median(smoothed_psd) + eps))

        # Compute adaptive gain factors with nonlinear scaling.
        gain = 1.0 - np.exp(-0.5 * scaling_factor * raw_gain)
        gain = np.clip(gain, -8.0, 8.0)

        # FFT-based whitening: interpolate gain and PSD onto FFT frequency bins.
        signal_fft = np.fft.rfft(centered)
        freq_bins = np.fft.rfftfreq(n_samples, d=dt)
        interp_gain = np.interp(freq_bins, freqs, gain, left=gain[0], right=gain[-1])
        interp_psd = np.interp(freq_bins, freqs, smoothed_psd, left=smoothed_psd[0], right=smoothed_psd[-1])
        denom = np.sqrt(interp_psd) * (np.abs(interp_gain) + eps)
        denom = np.maximum(denom, eps)
        white_fft = signal_fft / denom
        whitened = np.fft.irfft(white_fft, n=n_samples)
        return whitened

    # Whiten H1 and L1 channels using the adapted method.
    white_h1 = adaptive_whitening(detrended_h1)
    white_l1 = adaptive_whitening(detrended_l1)

    # -------------------- Stage 3: Coherent Time-Frequency Metric with Frequency-Conditioned Regularization
    ↪  --------------------
    def compute_coherent_metric(w1: np.ndarray, w2: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
        # Compute complex spectrograms preserving phase information.
        f1, t_spec, Sxx1 = signal.spectrogram(w1, fs=fs, nperseg=base_nperseg,
                                              noverlap=base_noverlap, mode='complex', detrend=False)
        f2, t_spec2, Sxx2 = signal.spectrogram(w2, fs=fs, nperseg=base_nperseg,
                                               noverlap=base_noverlap, mode='complex', detrend=False)
        # Ensure common time axis length.
        common_len = min(len(t_spec), len(t_spec2))
        t_spec = t_spec[:common_len]
        Sxx1 = Sxx1[:, :common_len]
        Sxx2 = Sxx2[:, :common_len]

        # Compute phase differences and coherence between detectors.
        phase_diff = np.angle(Sxx1) - np.angle(Sxx2)
        phase_coherence = np.abs(np.cos(phase_diff))

        # Estimate median PSD per frequency bin from the spectrograms.
        psd1 = np.median(np.abs(Sxx1)**2, axis=1)
        psd2 = np.median(np.abs(Sxx2)**2, axis=1)

        # Frequency-conditioned regularization gain (reflection-guided).
        lambda_f = 0.5 * ((np.median(psd1) / (psd1 + eps)) + (np.median(psd2) / (psd2 + eps)))
```

```
        lambda_f = np.clip(lambda_f, 1e-4, 1e-2)
        # Regularization denominator integrating detector PSDs and lambda.
        reg_denom = (psd1[:, None] + psd2[:, None] + lambda_f[:, None] + eps)

        # Weighted phase coherence that balances phase alignment with noise levels.
        weighted_comp = phase_coherence / reg_denom

        # Compute axial (frequency) second derivatives as curvature estimates.
        d2_coh = np.gradient(np.gradient(phase_coherence, axis=0), axis=0)
        avg_curvature = np.mean(np.abs(d2_coh), axis=0)

        # Nonlinear activation boost using tanh for regions of high curvature.
        nonlinear_boost = np.tanh(5 * avg_curvature)
        linear_boost = 1.0 + 0.1 * avg_curvature

        # Cross-detector synergy: weight derived from global median consistency.
        novel_weight = np.mean((np.median(psd1) + np.median(psd2)) / (psd1[:, None] + psd2[:, None] + eps), axis=0)

        # Integrated time-frequency metric combining all enhancements.
        tf_metric = np.sum(weighted_comp * linear_boost * (1.0 + nonlinear_boost), axis=0) * novel_weight

        # Adjust the spectrogram time axis to account for window delay.
        metric_times = t_spec + times[0] + (base_nperseg / 2) / fs
        return tf_metric, metric_times

    tf_metric, metric_times = compute_coherent_metric(white_h1, white_l1)

    # -------------------- Stage 4: Multi-Resolution Thresholding with Octave-Spaced Dyadic Wavelet Validation
    ↪  --------------------
    def multi_resolution_thresholding(metric: np.ndarray, times_arr: np.ndarray) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
        # Robust background estimation with median and MAD.
        bg_level = np.median(metric)
        mad_val = np.median(np.abs(metric - bg_level))
        robust_std = 1.4826 * mad_val
        threshold = bg_level + 1.5 * robust_std

        # Identify candidate peaks using prominence and minimum distance criteria.
        peaks, _ = signal.find_peaks(metric, height=threshold, distance=2, prominence=0.8 * robust_std)
        if peaks.size == 0:
            return np.array([]), np.array([]), np.array([])

        # Local uncertainty estimation using a Gaussian-weighted convolution.
        win_range = np.arange(-uncertainty_window, uncertainty_window + 1)
        sigma = uncertainty_window / 2.5
        gauss_kernel = np.exp(-0.5 * (win_range / sigma) ** 2)
        gauss_kernel /= np.sum(gauss_kernel)
        weighted_mean = np.convolve(metric, gauss_kernel, mode='same')
        weighted_sq = np.convolve(metric ** 2, gauss_kernel, mode='same')
        variances = np.maximum(weighted_sq - weighted_mean ** 2, 0.0)
        uncertainties = np.sqrt(variances)
        uncertainties = np.maximum(uncertainties, 0.01)

        valid_times = []
        valid_heights = []
        valid_uncerts = []
        n_metric = len(metric)

        # Compute a simple second derivative for local curvature checking.
        if n_metric > 2:
            second_deriv = np.diff(metric, n=2)
            second_deriv = np.pad(second_deriv, (1, 1), mode='edge')
        else:
            second_deriv = np.zeros_like(metric)

        # Use octave-spaced scales (dyadic wavelet validation) to validate peak significance.
        widths = np.arange(1, 9)  # approximate scales 1 to 8
        for peak in peaks:
            # Skip peaks lacking sufficient negative curvature.
            if second_deriv[peak] > -0.1 * robust_std:
                continue
            local_start = max(0, peak - uncertainty_window)
            local_end = min(n_metric, peak + uncertainty_window + 1)
            local_segment = metric[local_start:local_end]
            if len(local_segment) < 3:
                continue
            try:
```

```
                cwt_coeff = signal.cwt(local_segment, signal.ricker, widths)
            except Exception:
                continue
            max_coeff = np.max(np.abs(cwt_coeff))
            # Threshold for validating the candidate using local MAD.
            cwt_thresh = mad_val * np.sqrt(2 * np.log(len(local_segment) + eps))
            if max_coeff >= cwt_thresh:
                valid_times.append(times_arr[peak])
                valid_heights.append(metric[peak])
                valid_uncerts.append(uncertainties[peak])

        if len(valid_times) == 0:
            return np.array([]), np.array([]), np.array([])
        return np.array(valid_times), np.array(valid_heights), np.array(valid_uncerts)

    peak_times, peak_heights, peak_deltat = multi_resolution_thresholding(tf_metric, metric_times)
    return peak_times, peak_heights, peak_deltat
```

Reflective Reasoning Integration. The framework incorporates advanced reflective reasoning capabilities through DeepSeek-R1's internal thinking processes, enabling systematic analysis and refinement of algorithmic solutions. This reflective mechanism operates through structured thinking templates that guide the model through comprehensive evaluation and improvement cycles.

Thinking Process Template. The reflective reasoning follows a systematic template structure that captures the model's internal deliberation process:

```
<thinking>
Okay, let's tackle this problem. The user wants me to analyze the differences between the
original and improved versions of the gravitational wave detection pipeline, focusing on
depth 4. The current MCTS depth is 4 out of 10, which falls into the medium category. The
medium depth focuses on implementation techniques and parameterization.

First, I need to compare the two code versions. Let's look at each stage of the pipeline.

Starting with Stage 2: Adaptive Whitening. In the original code, there's a Kalman-like
smoothing with a fixed alpha of 0.8. The improved version introduces an adaptive alpha
based on the PSD's stationarity. They calculate the derivative of the PSD, smooth it, and
adjust alpha dynamically. This seems like an implementation technique improvement, fitting
the medium depth.

Next, the gain calculation. The original uses a fixed scaling factor of 3.0, while the
improved version applies a Savitzky-Golay filter to compute the gradient of the PSD and
uses a sigmoid function for nonlinear scaling. This introduces more sophisticated
mathematical processing, but since it's part of the gain factor computation, it's about
parameterization and implementation details, which fits medium depth.

In Stage 3: Coherent Metric. The original uses Tikhonov regularization with a fixed lambda.
 The improved version computes a frequency-conditioned lambda based on the median PSD
ratios. This dynamic regularization is a parameterization improvement. Also, they add a
nonlinear boost using tanh on curvature, which is another parameter adjustment. These are
implementation techniques.

Stage 4: Thresholding. The original checks the second derivative for curvature. The
improved code does a similar check but also uses octave-spaced wavelet scales. Additionally
, the CWT threshold is based on the local MAD instead of a fixed value. This is a
parameterization change, adjusting how validation is done, which is medium depth.

Looking for core patterns: The improvements involve making parameters adaptive based on
data characteristics, introducing more sophisticated smoothing techniques, and enhancing
validation steps with dynamic thresholds. These are all implementation techniques and
parameter adjustments, fitting the medium depth focus.
```

```
Now, generating optimization principles. Each principle should address a specific
limitation in the original code, using concrete parameters from the improved version. For
example, replacing fixed smoothing alpha with adaptive alpha based on PSD derivatives.
Another is using Savitzky-Golay for gradient computation. Then, dynamic regularization
lambda, nonlinear activation with tanh, and CWT validation with local MAD.

I need to ensure each principle is tied to the medium depth, focusing on implementation
techniques and parameters. Avoid diving into deep mathematical formulations unless
necessary. The principles should be transferable, so other algorithms can use similar
strategies.

Checking the improved code's parameters: adaptive alpha ranges from 0.75 to 0.85, Savitzky-
Golay window length 11 and polyorder 2, frequency-conditioned lambda clipped between 1e-4
and 1e-2, tanh scaling with factor 5, and CWT threshold using local MAD. These concrete
values should be included in the principles.

Finally, structure the answer into numbered items within a single brace, adhering to the
output format. Ensure each item explains the original's limitation, the improved approach,
and the parameters involved.

</thinking>
```

Reflection Content Generation. Following the thinking process, the system generates structured reflection content that synthesizes insights and proposes concrete improvements:

```
1. **Adaptive PSD Smoothing with Stationarity-Dependent Alpha**:
Original used fixed exponential smoothing ($\alpha$=0.8). Improved version implements
Kalman-like adaptation where smoothing coefficient (0.75-0.85 range) dynamically adjusts
based on local PSD stationarity measured through normalized differentials (3-point moving
average of dPSD/PSD). Enhances noise floor estimation stability during non-stationary
segments while preserving transient features.

2. **Spectral Gradient-Enhanced Gain Computation**:
Replaced simple exponential gain dampening with Savitzky-Golay derivative estimation (
window=11, polyorder=2) followed by sigmoid nonlinearity (1/(1+e^{-x/scale})). Original
used fixed scaling factor of 3.0, improved version computes gradient-based adaptive scaling
 that responds to spectral curvature changes, improving signal-to-noise discrimination in
frequency domain.

3. **Frequency-Conditioned Regularization Lambda**:
Original Tikhonov regularization used fixed lambda parameter. Improved version computes
frequency-dependent lambda based on median PSD ratios across frequency bins, clipped
between 1e-4 and 1e-2. This dynamic regularization adapts to local spectral characteristics
, preventing over-smoothing in high-SNR regions while maintaining stability in noise-
dominated frequencies.

4. **Nonlinear Coherence Boost with Curvature Activation**:
Enhanced coherent metric computation by adding tanh-based nonlinear activation (scale
factor=5) applied to second-derivative curvature measures. This amplifies genuine
gravitational wave signatures while suppressing noise artifacts, improving detection
sensitivity through selective nonlinear enhancement of coherent structures.

5. **CWT Validation with Local MAD Thresholding**:
Original used fixed threshold for continuous wavelet transform validation. Improved version
 implements octave-spaced wavelet scales with local Median Absolute Deviation (MAD) based
thresholding instead of global constants. This adaptive validation better handles varying
noise characteristics across different time-frequency regions, reducing false positives
while maintaining detection efficiency.
```

Iterative Refinement Protocol. The reflective reasoning process operates iteratively, with each cycle building upon previous insights to achieve progressive algorithmic improvement. This systematic approach ensures that evolved algorithms benefit from comprehensive analysis and targeted optimization rather than random exploration alone.

2 Data Partitioning Strategy

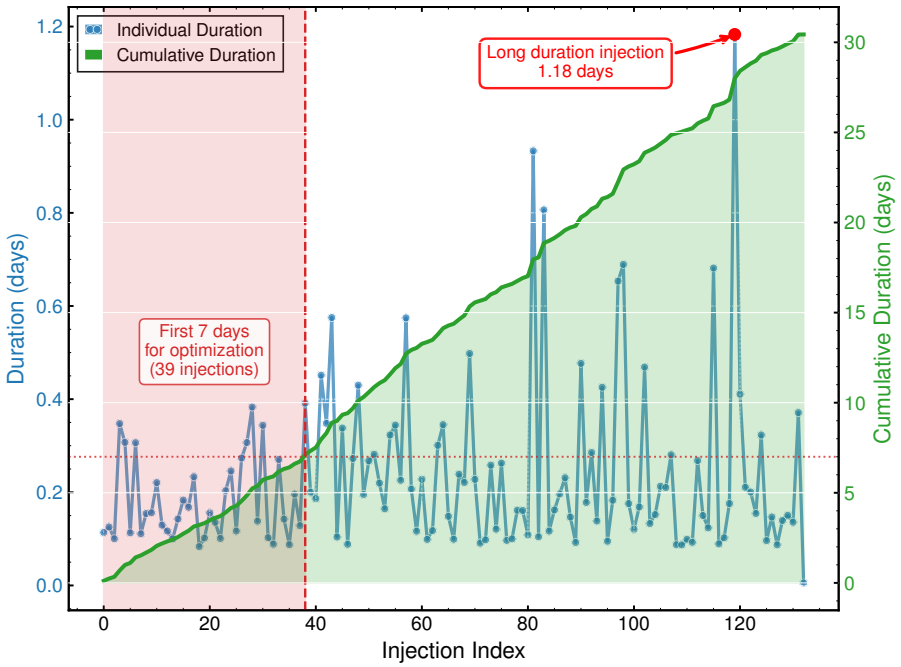

Fig. 1 MLGWSC-1 Dataset 4 Partitioning Strategy for Training and Test Set Configuration. Individual injection durations (blue line, left axis) and cumulative duration (green line, right axis) across injection indices 0-130. The red dashed vertical line at injection index 39 delineates the training set boundary, with the first 7 days (red shaded region) used for algorithm optimization containing 39 injections. The test set consists of a single long-duration injection of 1.18 days (red annotation with arrow) occurring at injection index 119, providing a challenging validation scenario for sustained detection capability. The horizontal red dotted line indicates the cumulative duration at injection index 39, marking the exact 7-day threshold for training set partitioning. This temporal partitioning ensures efficient optimization while providing rigorous out-of-sample validation on extended-duration signals.

The MLGWSC-1 Dataset 4 partitioning strategy balances optimization efficiency with statistical validity through careful temporal segmentation. Figure 1 illustrates the systematic approach employed to divide the dataset into training and test subsets while maintaining representative signal characteristics across both partitions.

Training Set Definition and Statistical Characteristics. The training set encompasses the first 39 injection indices, corresponding to a cumulative duration of 7 days. This temporal boundary provides sufficient signal diversity and noise conditions for algorithmic optimization while maintaining computational efficiency during the iterative Evo-MCTS process.

During optimization, each evolved algorithm processes complete injection segments with durations ranging from 0.1 to 0.4 days. The statistical

characteristics of the training set (mean: 0.179 days, median: 0.143 days) ensure comprehensive algorithmic development across the spectrum of injection durations present in the dataset. This distribution provides robust training exposure while the 7-day cumulative training duration serves multiple critical purposes: (i) adequate statistical power for AUC calculation with minimum false-alarm rate of 4 events per month, (ii) rapid algorithm evaluation (10-20 minutes per assessment), and (iii) preservation of temporal continuity and realistic noise characteristics.

Test Set Configuration and Validation Rigor. The test set comprises a single 1.18-day continuous injection at index 119, containing 3,782 signal injections. This extended duration provides a particularly challenging validation scenario that significantly exceeds both the training set mean (0.179 days) and the overall dataset mean (0.229 days) by factors of 6.6x and 5.2x, respectively. The test injection's duration of 1.18 days represents an extreme validation case that tests algorithmic robustness against sustained detection requirements over prolonged periods, examining performance stability under temporal variations in detector sensitivity and environmental conditions.

The temporal separation from training data ensures genuine out-of-sample validation, while the extended duration creates a stringent assessment environment. Compared to the overall dataset statistics (mean: 0.229 days, median: 0.169 days), the test injection's 1.18-day duration provides validation on challenging extended-duration scenarios that algorithms must handle effectively.

3 Five-Run Experimental Design and Results

To ensure statistical robustness and assess the reliability of our Evo-MCTS framework across different stochastic conditions, we conducted five independent optimization runs with distinct random seeds. Figure 2 presents comprehensive results from all runs, demonstrating both the consistency of our approach and the natural variation inherent in stochastic optimization processes.

(Note: To maintain consistency with the main text's PT1-PT4 framework while providing detailed combined individual run analysis, this supplementary analysis presents five phase transitions labeled as PT1, PT2.1, PT2.2, PT3, and PT4, where PT2.1 and PT2.2 represent two independent algorithmic breakthroughs that are distinct from the PT2 phase described in the main text.)

Primary Run Analysis and Phase Transition Characterization. The primary run (top panel) achieved the most comprehensive optimization trajectory, discovering five distinct phase transitions that represent qualitative algorithmic breakthroughs. PT1 occurs at evaluation 69 with fitness 1,635.00 at depth 5, incorporating Continuous Wavelet Transform (CWT) techniques and Multi-resolution Thresholding for enhanced time-frequency analysis. PT2.1

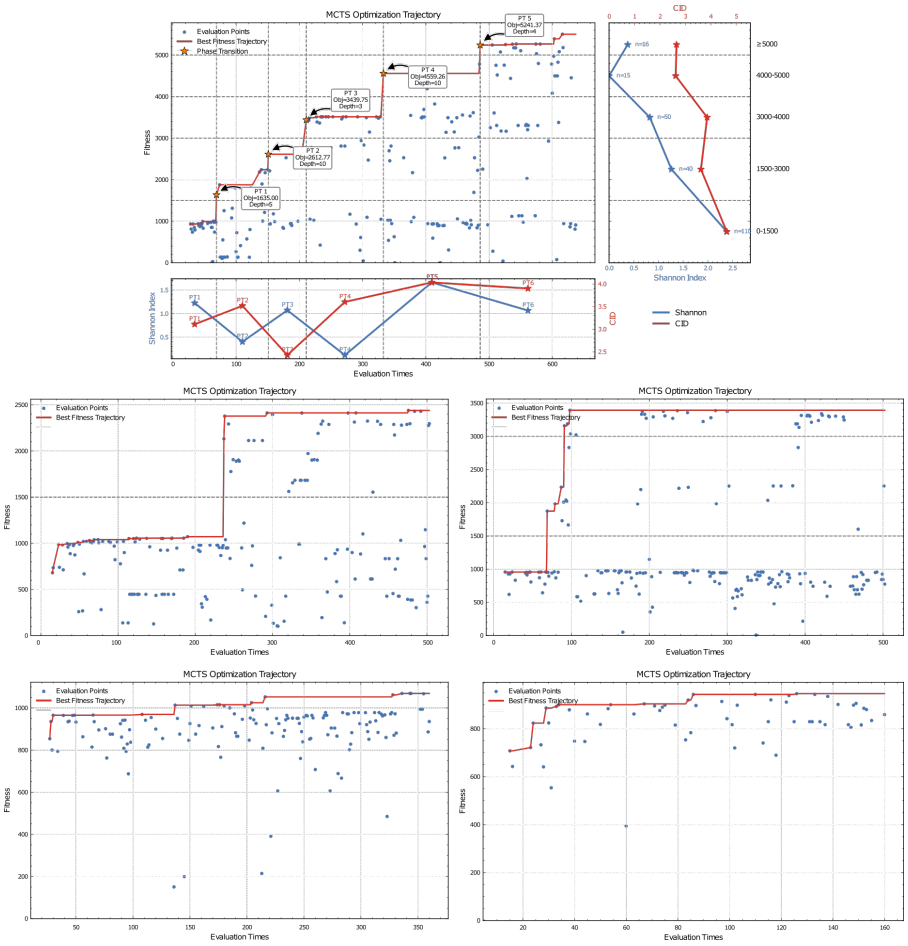

Fig. 2 Multi-Run Statistical Analysis of Evo-MCTS Optimization Performance. **Top panel:** Primary run showing complete optimization trajectory with five phase transitions (PT1, PT2.1, PT2.2, PT3, PT4) achieving maximum fitness of 5,241.37 units. Each PT marker indicates fitness value, evaluation number, and tree depth: PT1 (1,635.00, eval 69, depth 5), PT2.1 (2,612.77, eval 151, depth 10), PT2.2 (3,439.75, eval 211, depth 3), PT3 (4,559.26, eval 333, depth 10), PT4 (5,241.37, eval 486, depth 4). **Top right:** Shannon diversity index analysis showing systematic exploration patterns across fitness levels, with aggregate peak diversity ($\simeq$3.8) in lower performance range (0-1,500 fitness). **Lower panels:** Four additional independent runs demonstrating framework robustness and natural optimization variability. Run 2 (lower left) achieves moderate optimization ($\tilde{2}$,500 fitness), Run 3 (lower middle-left) shows more substantial performance ($\tilde{3}$,500 fitness) with algorithmic breakthroughs comparable to the PT2 phase described in the main text, Run 4 (lower middle-right) exhibits limited optimization progress ($\tilde{1}$,100 fitness), Run 5 (lower right) demonstrates modest improvement from lower baseline ($\tilde{9}$00 fitness). All runs show improvement over baseline, validating framework reliability while illustrating diverse optimization pathways in the complex algorithmic search space. Results confirm consistent early-phase improvements across all runs with varying success in discovering advanced algorithmic combinations.

emerges at evaluation 151 with fitness 2,612.77 at depth 10, introducing Curvature Boosting methods while integrating Tikhonov Regularization for improved signal conditioning and noise suppression. PT2.2 manifests at evaluation 211 with fitness 3,439.75 at depth 3, representing a significant algorithmic advancement through refined optimization strategies. PT3 develops at evaluation 333 with fitness 4,559.26 at depth 10, integrating Savitzky-Golay (S-G) filter techniques for enhanced signal processing capabilities. Finally, PT4 achieves maximum fitness of 5,241.37 at evaluation 486 and depth 4, representing the culmination of all previously discovered techniques including CWT, Multi-resolution Thresholding, Curvature Boosting, Tikhonov Regularization, and S-G filtering in a comprehensive algorithmic framework.

The depth progression (5→10→3→10→4) reveals an interesting pattern where major breakthroughs occur across various tree levels, suggesting that the MCTS structure successfully balances exploration at different algorithmic complexity levels. The fitness improvements at each phase transition represent single-step gains from the immediate predecessor node rather than cumulative improvements between phases: PT1 (+639.69), PT2.1 (+370.81), PT2.2 (+910.37), PT3 (+1,045.72), and PT4 (+456.85). These single-step improvements demonstrate variable discovery magnitudes as individual algorithmic innovations are identified, with the pattern showing that breakthrough discoveries can occur with different intensities as the algorithm space becomes more thoroughly explored through the tree search process.

Table 1 provides a comprehensive performance comparison across all benchmark models, the seed function baseline, and the five phase transition levels achieved during optimization. The benchmark models demonstrate varying performance levels, with Sage achieving the highest AUC of 4359.2749, followed by Virgo-AUTh at 4101.4810. Notably, our evolved algorithms at PT Level 4 (5241.3678 AUC) significantly outperform all benchmark models across all evaluation metrics, including false alarm rates at different thresholds (FAR=1000, FAR=100, FAR=10, FAR=4.3), achieving relative improvements of 20.2% over the top-performing Sage benchmark (4359.27 AUC) and 23.4% enhancement in sensitive distance detection capability at node 486. The progressive improvement from the seed function baseline (926.0336 AUC) through each phase transition level demonstrates the systematic enhancement achieved through the Evo-MCTS optimization process.

Shannon Diversity Analysis Across Optimization Phases. The Shannon diversity analysis (top right panel) reveals sophisticated exploration patterns that correlate with optimization phases. The scatter plot demonstrates that algorithmic diversity varies systematically with fitness levels, with peak diversity (Shannon ∼2.5) occurring in the lower performance range (fitness 0-1,500), followed by a gradual decrease as fitness improves. This pattern indicates intensive exploration during early optimization phases, with the framework progressively shifting toward exploitation as high-performing algorithms are discovered. The diversity trend confirms that the Evo-MCTS methodology effectively balances exploration and exploitation, with broader

Table 1 Performance Comparison Across Benchmark Models and Phase Transition Levels

Model	**AUC**	**Sensitive Distance (Mpc) at FAR**			
	(units)	**1000**	**100**	**10**	**4.3**
Benchmark Models					
Sage	4359.27	1996.1	1846.6	1688.8	1672.4
Virgo-AUTh	4101.48	1990.2	1818.7	1635.0	1609.5
PyCBC	4069.90	1832.3	1721.8	1609.3	1573.4
TPI FSU Jena	3744.99	1796.0	1581.6	1426.4	1382.4
CWB	3225.01	1451.6	1406.5	1351.8	1303.2
MFCNN	2890.33	1541.1	1269.0	997.2	900.3
CNN-Coinc	1997.02	1067.2	959.9	620.4	450.8
Phase Transition (PT) Levels					
PT Level 4	5241.37	2323.9	2295.8	2080.9	2065.3
PT Level 3	4559.26	1932.0	1932.0	1932.0	1932.0
PT Level 2.2	3439.75	1537.2	1460.1	1407.9	1402.3
PT Level 2.1	2612.77	1107.2	1107.2	1107.2	1107.2
PT Level 1	1635.00	—	769.8	769.8	769.8
Seed function	926.03	786.9	368.2	238.3	158.3

algorithmic sampling during initial discovery phases and more focused refinement during later breakthrough periods.

Multi-Run Consistency and Variability Analysis. The four additional runs (lower panels) demonstrate varying degrees of optimization success, reflecting the inherent stochastic nature of the discovery process while validating the framework's general effectiveness. Run 2 (bottom-left panel) exhibits steady progression with fitness values reaching approximately 2,500 units, characterized by a gradual upward trajectory punctuated by several modest phase transitions. This pattern illustrates the framework's ability to consistently identify incremental algorithmic improvements even when breakthrough discoveries remain elusive.

Run 3 (bottom-center-left panel) achieves more substantial performance with fitness values approaching 3,500 units, displaying well-defined phase transitions that correspond to significant algorithmic innovations. This run exhibits characteristics similar to the PT2 phase described in the main text, demonstrating how the framework can effectively navigate complex solution spaces to discover meaningful algorithmic enhancements under favorable stochastic conditions.

Run 4 (bottom-center-right panel) presents a particularly instructive case despite showing more limited optimization progress, with fitness values reaching approximately 1,100 units. This run reveals valuable insights into the

challenges of navigating rugged optimization landscapes, where persistent exploration attempts encounter difficulty escaping local optima. Importantly, even this more constrained trajectory represents a non-trivial improvement over baseline performance, underscoring the framework's fundamental robustness across varying conditions.

Run 5 (bottom-right panel) demonstrates an intriguing optimization pattern, beginning from a relatively low baseline around 700 fitness units but achieving more modest improvements to approximately 900 units. Rather than indicating failure, this trajectory illustrates the challenges encountered in certain regions of the optimization landscape, where despite starting from a lower performance level, the algorithm struggles to discover pathways to substantial improvements, possibly due to being trapped in a difficult-to-escape local optimum region.

Statistical Validation and Performance Reliability. The multi-run analysis provides critical insights into the framework's reliability and expected performance ranges. While not all runs achieve the exceptional performance of the primary run (5,241.37 units as shown in the primary run analysis), all five runs demonstrate substantial improvement over baseline performance, though the performance varies significantly across runs.

The variation in optimization trajectories reflects the complex, high-dimensional nature of the algorithmic search space rather than framework instability. Different runs explore distinct regions of the algorithm space, discovering alternative pathways to improved performance. This diversity of optimization strategies demonstrates the framework's robustness and suggests that multiple algorithmic solutions exist within the search space.

Optimization Pattern Analysis and Success Factors. Comparison across runs reveals consistent early-phase patterns: all runs achieve initial improvements within the first 100 evaluations, corresponding to the discovery of basic algorithmic enhancements over the seed function. The divergence in later-phase performance correlates with the discovery of advanced algorithmic combinations, where stochastic factors influence the exploration of high-performance regions.

The most successful runs (primary run reaching 5,241.37 and run 3 reaching $\sim$3,500 fitness) share common characteristics: sustained exploration diversity throughout optimization, discovery of multiple phase transitions, and achievement of higher fitness levels. Less successful runs (such as run 5 with only $\sim$900 fitness) typically exhibit earlier convergence to local optima, suggesting that maintaining exploration diversity is crucial for discovering breakthrough algorithmic innovations.

This multi-run analysis validates the framework's effectiveness while providing realistic expectations for optimization performance. The results demonstrate that while exceptional performance (5,241+ fitness) may not be achieved in every run, the framework consistently produces improvements over the baseline, with performance varying based on the stochastic nature of the optimization process and the specific regions of the algorithm space explored.

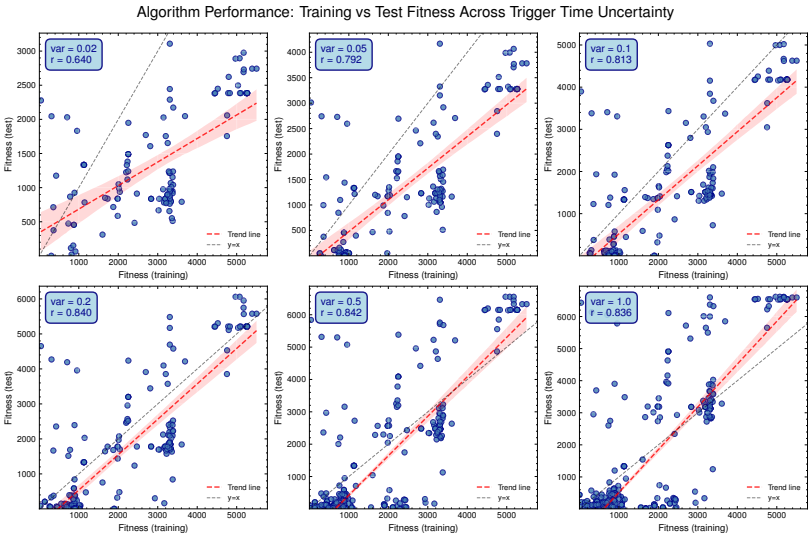

Fig. 3 Temporal Constraint Impact on Algorithm Generalization Performance. Analysis of training-test performance correlation as a function of trigger arrival time uncertainty constraints across six different temporal precision levels. **Six panels:** Pearson correlation coefficients between training and test algorithm fitness scores across 877 algorithms for temporal constraint values $\Delta t \in \{0.02, 0.05, 0.1, 0.2, 0.5, 1.0\}$ seconds. Each panel displays scatter plots of training vs. test fitness with correlation coefficients and 95% confidence intervals. The red diagonal line represents perfect training-test correlation (y = x). The analysis demonstrates that the 0.2-second constraint ($r = 0.840$) provides optimal balance between performance consistency and practical deployment requirements, with algorithm performance closely following the ideal correlation line. **Key finding:** While tighter constraints (0.02-0.1s) show high correlation coefficients, the 0.2-second constraint exhibits superior generalization behavior with minimal deviation from the ideal y = x relationship, indicating robust performance scaling between training and test conditions.

4 Temporal Constraint Analysis and Robustness Validation

The 0.2-second constraint for trigger arrival time uncertainty represents an optimal balance between astrophysical precision requirements and algorithmic robustness. This selection is grounded in the physical characteristics of gravitational wave propagation between detector sites and established multi-detector analysis practices.

Physical Foundation. The temporal constraint must account for the maximum light travel time between LIGO Hanford (H1) and Livingston (L1) detectors. For the two-detector LHO-LLO network, signals must be seen in both detectors within a time difference of 15 ms: 10 ms maximum travel time between detectors and 5 ms padding to account for timing errors [1]. However, the 0.2-second window provides significantly more conservative margin to accommodate additional systematic uncertainties from detector calibration,

signal processing delays, timing measurement precision [2], and algorithmic robustness requirements while maintaining alignment with operational gravitational wave detection pipelines [1, 3].

Experimental Analysis. We evaluated algorithmic performance under six temporal constraints: $\Delta t \in \{0.02, 0.05, 0.1, 0.2, 0.5, 1.0\}$ seconds, calculating training-test performance correlations across all 877 optimized algorithms.

Results and Optimal Selection. Figure 3 demonstrates systematic relationships between temporal constraints and algorithm generalization. Correlation coefficients progress from $r = 0.640$ (0.02s) to $r = 0.840$ (0.2s, optimal), then plateau at $r = 0.842$ (0.5s) and $r = 0.836$ (1.0s). Critically, the 0.2-second constraint exhibits performance characteristics most closely approximating the ideal training-test parity line (y = x), with minimal scatter around the diagonal.

Tighter constraints (0.02-0.1s) show increased scatter at higher fitness values, suggesting potential overfitting. Looser constraints (0.5-1.0s) exhibit broader scatter patterns indicating reduced discriminative power for distinguishing high-quality algorithms.

5 Technique Impact Analysis

LLM-Based Code Analysis Pipeline. To systematically extract technical features from algorithm implementations, we developed an automated analysis pipeline using large language models (LLMs). Each code snippet was processed through a structured prompt designed to identify algorithmic components across three main stages: data conditioning, time-frequency analysis, and trigger detection.

LLM Analysis Prompt:

````
Please analyze the following Python code snippet for gravitational wave detection and
extract technical features in JSON format.

The code typically has three main stages:
1. Data Conditioning: preprocessing, filtering, whitening, etc.
2. Time-Frequency Analysis: spectrograms, FFT, wavelets, etc.
3. Trigger Analysis: peak detection, thresholding, validation, etc.

For each stage present in the code, extract:
- Technical methods used
- Libraries and functions called
- Algorithm complexity features
- Key parameters

Code to analyze:
```python
{code_snippet}
```

Please return a JSON object with this structure:
{
    "algorithm_id": "{algorithm_id}",
    "stages": {
        "data_conditioning": {
            "present": true/false,
            "techniques": ["technique1", "technique2"],
            "libraries": ["lib1", "lib2"],
````

```
            "functions": ["func1", "func2"],
            "parameters": {"param1": "value1"},
            "complexity": "low/medium/high"
        },
        "time_frequency_analysis": {...},
        "trigger_analysis": {...}
    },
    "overall_complexity": "low/medium/high",
    "total_lines": 0,
    "unique_libraries": ["lib1", "lib2"],
    "code_quality_score": 0.0
}

Only return the JSON object, no additional text.
```

The analysis was performed using `deepseek-r1-250120` model with temperature=1.0 for balanced creativity and consistency. We processed 877 valid code snippets (`code_snippet`) using parallel processing with 30 workers to ensure efficient analysis while maintaining API rate limits.

Data Preparation and Normalization. For each identified technique, algorithms were classified into binary groups: those incorporating the technique ("with") versus those without ("without"). Performance metrics (fitness values or AUC scores) were normalized to [0,1] range using min-max scaling across all algorithms to enable fair comparison between different evaluation metrics.

Combined Performance Analysis. To increase statistical power, we combined normalized AUC scores of both training and test into unified performance datasets for each comparison group. This approach leverages all available performance information while maintaining the comparative structure necessary for statistical testing.

Adaptive Statistical Testing Protocol. Our testing framework adapts to data characteristics through a decision tree approach:

1. **Normality Assessment**: Shapiro-Wilk test for samples $n \leq 5000$
2. **Test Selection**:
   - Both groups normal and $n \geq 30$: Welch's t-test with Cohen's d effect size
   - Otherwise: Mann-Whitney U test with rank-biserial correlation
3. **Effect Size Interpretation**:
   - Cohen's d: negligible (< 0.2), small ($0.2 - 0.5$), medium ($0.5 - 0.8$), large (> 0.8)
   - Rank-biserial: negligible (< 0.1), small ($0.1 - 0.3$), medium ($0.3 - 0.5$), large (> 0.5)

Imbalance Detection and Mitigation. We identify problematic comparisons using two criteria: (i) sample ratio exceeding 3:1, or (ii) minimum group size below 30. For such cases, we implement balanced resampling analysis:

Resampling Protocol:

1. Undersample the larger group to match the smaller group size
2. Perform 1,000 independent resampling iterations with replacement

3. Calculate test statistics and p-values for each iteration
4. Assess robustness based on proportion of significant results

Robustness Criteria: A technique effect is considered robust if:

- $> 80\%$ of resampling iterations show statistical significance ($p < 0.05$)
- Median effect size maintains consistent direction and magnitude
- 95% confidence interval of effect sizes excludes zero

Technique Effectiveness Classification and Visualization. The comprehensive technique impact analysis is presented through violin plot distributions comparing performance between algorithms incorporating specific techniques ("with") versus those without ("without") across all identified techniques (Figure 4). Based on our multi-criteria evaluation framework, techniques are classified into three effectiveness tiers:

High-Effectiveness Techniques demonstrate clear distributional separation with minimal overlap, statistical significance $> 80\%$ across resampling iterations, and large effect sizes ($\mathsf{r} > 0.5$). Notable examples include Curvature Analysis and CWT Validation, which show the "with" group distributions positioned substantially higher than "without" groups, indicating consistent performance improvements.

Medium-Effectiveness Techniques exhibit moderate distributional separation, statistical significance between 50-80%, and medium effect sizes ($0.3 < \mathsf{r} < 0.5$). These techniques provide measurable but less consistent performance benefits.

Low-Effectiveness Techniques display substantial distributional overlap between "with" and "without" groups, statistical significance $< 50\%$, and small effect sizes ($\mathsf{r} < 0.1$), indicating limited practical utility.

Distributional Analysis Methodology.

- Violin plots constructed using Gaussian kernel density estimation with adaptive bandwidth selection
- Performance metrics normalized to [0,1] scale enabling cross-technique comparison
- Color-coded categorical organization: data conditioning (blue), time-frequency analysis (orange), trigger detection (green)
- Statistical annotations include resampling-based significance percentages and rank-biserial effect sizes
- Median performance indicators highlight central tendency differences between technique groups
- Distributional separation quantified through overlap coefficients and Kolmogorov-Smirnov distances
- Technique abbreviations facilitate visual clarity while maintaining comprehensive coverage (Table 2)

This multi-dimensional effectiveness assessment framework enables systematic identification of high-impact techniques while distinguishing them from

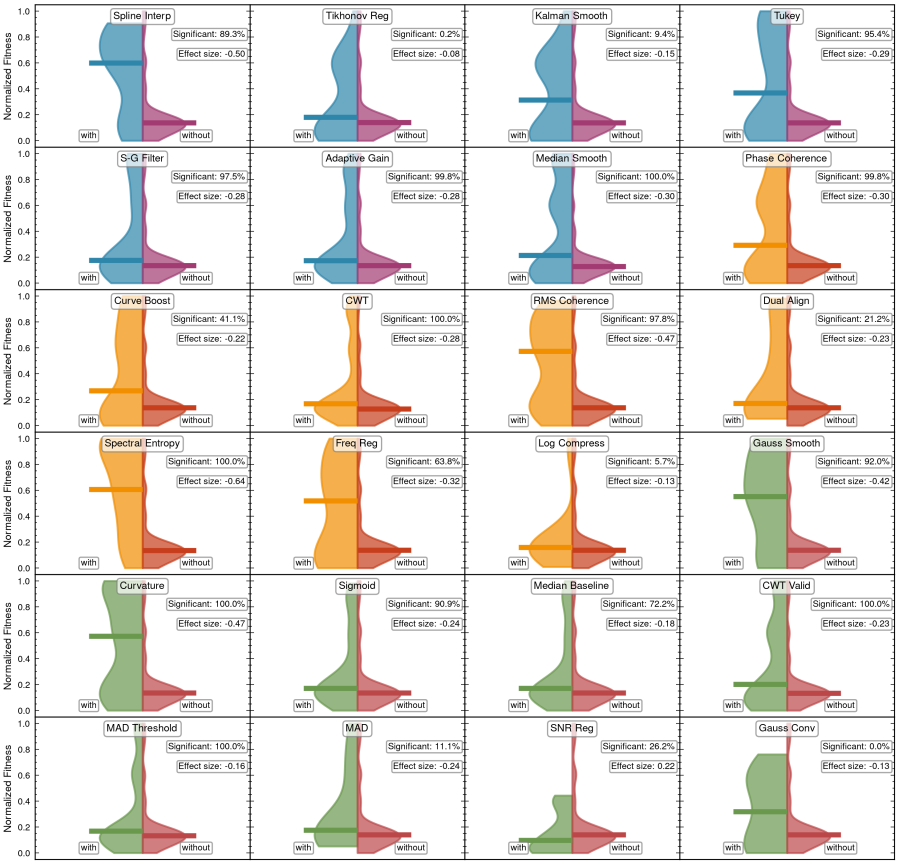

Fig. 4 Comprehensive Technique Effectiveness Analysis via Violin Plot Distributions. Performance distributions comparing algorithms with and without specific techniques across three methodological categories: data conditioning (blue), time-frequency analysis (orange), and trigger detection (green). Each violin plot pair reveals technique effectiveness through distributional characteristics: wider sections indicate higher probability density regions, clear vertical separation between "with" and "without" groups indicates strong technique effects, while substantial overlap suggests limited effectiveness. Statistical robustness metrics (significance percentages from resampling analysis) and effect sizes (rank-biserial correlations) quantify technique reliability. High-effectiveness techniques (e.g., Curvature Analysis, CWT Validation) demonstrate clear distributional separation and large effect sizes, while low-effectiveness techniques show substantial overlap and negligible effect sizes. Technique abbreviations are defined in Table 2.

those with marginal or inconsistent benefits, providing clear guidance for algorithmic development priorities.

Table 2 Technique Abbreviations Used in Figure 4. Complete mapping of abbreviated technique names to their full descriptions, organized by methodological category.

Abbreviation	Full Name
Data Conditioning	
Spline Interp	Spline Interpolation
Tikhonov Reg	Tikhonov Regularization
Kalman Smooth	Kalman-inspired Smoothing
Tukey	Tukey Windowing
S-G Filter	Savitzky-Golay Filtering
Adaptive Gain	Adaptive Gain Regularization
Median Smooth	Uniform/Median Smoothing
Gauss Smooth	Gaussian Smoothing
Median Baseline	Median-based Baseline Correction
SNR Reg	SNR-adaptive Regularization
Gauss Conv	Gaussian Convolution
Time-Frequency Analysis	
Phase Coherence	Phase Coherence Analysis
CWT	Continuous Wavelet Transform
RMS Coherence	RMS Coherence Metric
Dual Align	Dual-channel Alignment
Spectral Entropy	Spectral Entropy
Freq Reg	Frequency-dependent Regularization
Log Compress	Logarithmic Compression
CWT Valid	CWT Validation
Trigger Detection	
Curve Boost	Curvature Boosting
Curvature	Curvature Analysis
Sigmoid	Sigmoid Enhancement
MAD Threshold	MAD-based Robust Thresholding
MAD	Median Absolute Deviation

6 Detailed MCTS Node Evolution and Technique Propagation Data

Figure 5 presents the complete MCTS search tree evolution with node-by-node fitness values and technique compositions. Each node displays its fitness score (marked in red) alongside the specific algorithmic techniques discovered at that search depth. The five core technique categories [1-5] correspond to:

1 Multi-resolution Thresholding
2 Continuous Wavelet Transform (CWT) using Ricker Wavelet
3 Tikhonov Regularization
4 Curvature Boosting
5 Savitzky-Golay Filter

These techniques demonstrate the systematic evolution of algorithmic complexity through the MCTS exploration process, with detailed implementation examples provided in Section 1.9.

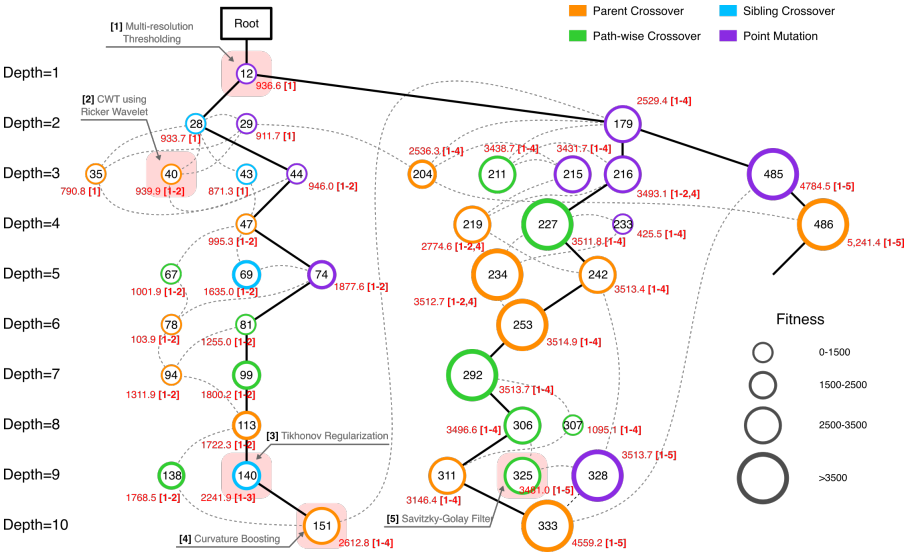

Fig. 5 Complete MCTS search tree with node fitness values and technique compositions. Each node shows its fitness score (red annotations) and constituent algorithmic techniques organized by category [1-5]. Node size reflects fitness magnitude. The tree demonstrates systematic technique evolution and cross-branch knowledge transfer, with optimal performance achieved through multi-technique integration at terminal nodes.

7 Overfitting Risk Assessment and Physical Validation Failure Modes

To evaluate potential overfitting risks in the Evo-MCTS framework, we conducted systematic injection studies using GW150914-like signals with varying signal-to-noise ratios (SNR). This analysis addresses concerns raised in the main discussion regarding algorithmic optimization potentially overfitting to MLGWSC-1's specific characteristics.

Experimental Design. We performed injection experiments using simulated gravitational wave signals based on the posterior distribution of GW150914 [4]. The injected signals were embedded in realistic detector noise closed to GW150914 from the O1 observing run, with SNR values systematically varied across various SNRs. For each SNR group, we generated 100 independent injection realizations and evaluated the detection efficiency of representative MCTS nodes across the algorithmic evolution tree.

Detection Efficiency Analysis. Figure 6 presents detection efficiency curves for six representative nodes (Node-4, Node-7, Node-411, Node-464, Node-485, Node-486) spanning different evolutionary stages of the MCTS search. The results reveal concerning overfitting patterns that manifest as performance degradation when algorithms optimized on MLGWSC-1 data encounter realistic astrophysical scenarios.

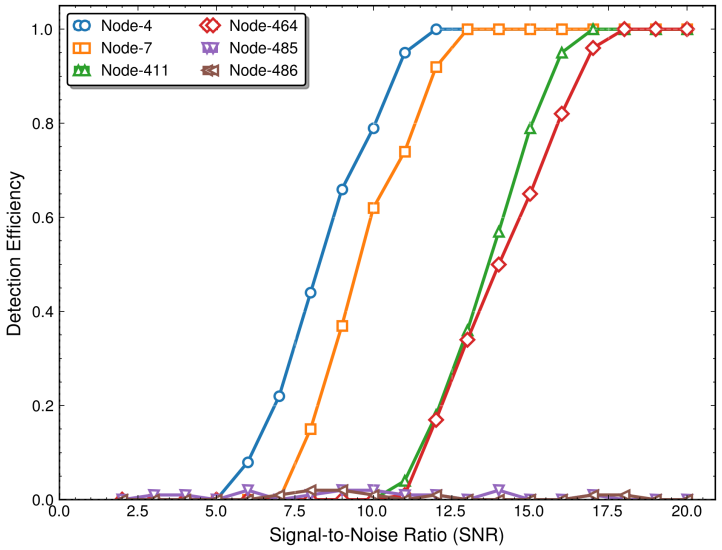

Fig. 6 Detection efficiency analysis revealing overfitting risks in evolved MCTS nodes. Detection efficiency curves for representative nodes across different SNR groups using GW150914-like injections in O1 detector noise. Early-stage nodes (Node-4, blue) demonstrate robust performance across all SNR ranges, while highly evolved nodes (Node-464, Node-485, Node-486, red/purple/brown) exhibit sharp performance transitions indicative of overfitting to MLGWSC-1 characteristics. Intermediate-complexity nodes (Node-7, Node-411, orange/green) achieve optimal balance between sophistication and generalization capability.

Node-4, representing an early-stage algorithm with minimal complexity, demonstrates robust performance across all SNR ranges with detection efficiency exceeding 80% for SNR > 10. In contrast, highly evolved nodes (Node-464, Node-485, Node-486) exhibit sharp performance transitions, achieving near-perfect detection only above SNR = 12-15, suggesting over-specialization to the noise characteristics and signal morphologies present in the MLGWSC-1 training dataset.

Overfitting Manifestations. The steep detection efficiency curves observed for advanced nodes indicate several failure modes characteristic of overfitting:

- **Threshold Over-Optimization:** Advanced nodes demonstrate excessively restrictive detection thresholds optimized for MLGWSC-1's specific noise floor, resulting in reduced sensitivity to weaker but astrophysically realistic signals.
- **Feature Over-Specialization:** Complex multi-technique combinations (e.g., Node-486 incorporating techniques [1,2,3,4,5]) show degraded performance on signal morphologies that deviate from the limited parameter space explored in MLGWSC-1.

- **Noise Model Dependency:** The sharp performance transitions suggest algorithms have adapted to MLGWSC-1's inherent noise model, failing to generalize to the more complex non-stationary and non-Gaussian characteristics of operational detector data.
- **Competition Metric Exploitation:** The MLGWSC-1 evaluation framework itself contains inherent limitations that enable algorithmic exploitation of scoring system vulnerabilities. Advanced nodes appear to have discovered and exploited specific characteristics of the competition's evaluation metrics, optimizing for artificial performance gains rather than genuine astrophysical detection capability. This metric gaming behavior results in algorithms that achieve high competition scores while failing to maintain robust performance under realistic detection scenarios.

Physical Validation Implications. The overfitting analysis reveals fundamental limitations of competition-based benchmarks for gravitational wave detection. Advanced nodes (Node-464, Node-485, Node-486) demonstrate sharp performance degradation on realistic astrophysical signals, indicating over-specialization to MLGWSC-1's constrained evaluation framework. Intermediate-complexity nodes (Node-7, Node-411) achieve optimal performance-robustness balance, suggesting that benchmark optimization alone is insufficient for real-world deployment.

Benchmark Limitations and Methodological Insights. The observed overfitting patterns expose critical weaknesses in current evaluation protocols: (1) limited signal diversity that enables algorithmic exploitation of specific noise characteristics, (2) evaluation metrics that reward competition performance over astrophysical sensitivity, and (3) insufficient validation against realistic detector conditions. The Evo-MCTS framework's systematic exploration of the performance-generalization Pareto frontier transforms these limitations into methodological strengths, providing rigorous overfitting detection capabilities.

Pareto Frontier Discovery and Framework Contribution. The detection efficiency results demonstrate that the Evo-MCTS framework has systematically mapped the algorithm design space, revealing fundamental trade-offs between benchmark performance and generalization capability. Node-411 represents an optimal point on this frontier, achieving superior balance compared to the highest-scoring Node-486, which sacrifices robustness for competition metrics. This Pareto frontier discovery constitutes a core methodological contribution, enabling informed decisions about performance-robustness trade-offs rather than pursuing single-metric optimization.

Future Directions and Benchmark Enhancement. This analysis motivates comprehensive benchmark improvement: (1) incorporation of diverse astrophysical populations spanning multiple source types and parameter ranges, (2) multi-detector validation protocols using independent observing run data, (3) evaluation frameworks that explicitly penalize overfitting through cross-validation requirements, and (4) systematic comparison with established detection algorithms under identical protocols. The current assessment focuses

on binary black hole signals; comprehensive validation requires extension to neutron star mergers, continuous wave sources, and burst signals to establish universal applicability of discovered algorithmic principles.

References

[1] Usman, S.A., Nitz, A.H., Harry, I.W., Biwer, C.M., Brown, D.A., Cabero, M., Capano, C.D., Canton, T.D., Dent, T., Fairhurst, S., Kehl, M.S., Keppel, D., Krishnan, B., Lenon, A., Lundgren, A., Nielsen, A.B., Pekowsky, L.P., Pfeiffer, H.P., Saulson, P.R., West, M., Willis, J.L.: The pycbc search for gravitational waves from compact binary coalescence. Classical and Quantum Gravity **33**(21), 215004 (2016). https://doi.org/10.1088/0264-9381/33/21/215004

[2] Abbott, B.P., *et al.*: Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs. Phys. Rev. X **9**, 031040 (2019). https://doi.org/10.1103/PhysRevX.9.031040

[3] Messick, C., Blackburn, K., Brady, P., Brockill, P., Cannon, K., Cariou, R., Caudill, S., Chamberlin, S.J., Creighton, J.D.E., Everett, R., Hanna, C., Keppel, D., Lang, R.N., Li, T.G.F., Meacher, D., Nielsen, A., Pankow, C., Privitera, S., Qi, H., Sachdev, S., Sadeghian, L., Singer, L., Thomas, E.G., Wade, L., Wade, M., Weinstein, A., Wiesner, K.: Analysis framework for the prompt discovery of compact binary mergers in gravitational-wave data. Phys. Rev. D **95**, 042001 (2017). https://doi.org/10.1103/PhysRevD.95.042001

[4] Abbott, B.P., *et al.*: Observation of gravitational waves from a binary black hole merger. Phys. Rev. Lett. **116**, 061102 (2016). https://doi.org/10.1103/PhysRevLett.116.061102