## Approach 1: WEST as Independent Service (Current Solution)

### Implementation Mechanics

- **Frontend Conversion**: Performs LTL→MLTL syntax translation within FRET
- **Manual Verification**: Users copy-paste results into local WEST environment

### Strengths

1. **Loose Coupling Architecture**
    - Maintenance Simplicity: Separate codebases eliminate version conflicts
    - Native Performance: Leverages compiled WEST binaries directly
    - Deployment Flexibility: Supports custom WEST paths
2. **Low Development Cost**
    - No IPC complexity (zero message queues/pipes)
    - Platform-agnostic execution (users handle OS compatibility)
3. **Security Control**
    - Sensitive operations (file I/O, system calls) remain user-controlled
    - Avoids automated compilation risks
4. **Feature Accessibility**
    - Full access to WEST's advanced GUI capabilities E.g. Subformula visualization trees, Trace regex set exploration,Time-step atomic proposition toggling

### Weaknesses

1. **Fragmented UX**
    - Requires context switches (FRET→WEST GUI)
    - No real-time feedback (e.g., formula highlighting ↔ trace updates)
2. **Functional Limitations**
    - Batch processing impractical (single-formula verification only)

    **Diagnostic Complexity**
    - Users must manually triage errors across tools
    - No unified logging for cross-tool debugging

## Approach 2: Deep Integration （based on Node.js ）

### Technical Implementation

- **Full-Stack Automation**: Embeds WEST via child_process
- **Workflow**: Formula conversion → compilation → verification → visualization

### Key Technical Challenges

1. **Self-Contained Environment**
   - Advantage: Eliminates environmental dependencies
   - Cost: Adds initial setup time and increased learning costs

   ```
   // Auto-deploy WEST
   const deployWEST = () => {
     if (!fs.existsSync('WEST')) {
       execSync('git clone https://github.com/zwang271/WEST.git');
     }
     execSync('make -C WEST/MLTL_reg/MLTL west');
   };
   ```
2. **Precise Process Control**
   - Advantage: Enables automated interaction
   - Challenge: Brittle pattern matching (e.g., can't handle dynamic subformula selection)

   ```
   const westProcess = spawn('./west', [], {
     cwd: WEST_DIR,
     stdio: ['pipe', 'pipe', 'pipe']
   });

   // Dynamic prompt handling
   westProcess.stdout.on('data', (data) => {
     if (data.includes('simplify output')) {
       westProcess.stdin.write('y\n'); // Auto-answer
     }
   });
   ```
3. **Visual Integration**
   - Advantage: Unified visualization layer
   - Challenge: Re-implementing WEST's terminal UI as React components

```
function renderTraceVisualization(westOutput) {
  const traces = parseTraces(westOutput);
  ReactDOM.render(<TraceHeatmap data={traces} />, document.getElementById('viz'));
}
```

### Strategic Advantages

1. **Seamless Experience**
   - End-to-end workflow within single interface
   - Interactive exploration (click formulas ↔ jump to truth tables)
2. **Automation Support**

```
// Batch processing example
async function verifyRequirements(reqList) {
  const results = [];
  for (const req of reqList) {
    results.push(await convertAndValidate(req));
  }
  return results;
}
```

### Risk Analysis

1. **Maintenance Complexity**
   - Requires tracking WEST API changes
   - Cross-platform testing overhead
2. **Performance Costs**
   - Caused throughput loss from IPC overhead
   - Memory bloat from caching large trace sets

## Approach 3: Rebuild WEST as a FRET Native Module

**Core Idea**
Rewrite WEST’s core logic in TypeScript/WebAssembly and deeply integrate it into FRET.

**Strengths**

1. **Enhanced Capabilities**
    - **Next-Gen Features**:
        - Hybrid LTL/MLTL editing with auto-conversion
        - Context-aware formula optimization suggestions
        - Real-time collaborative trace debugging
    - **Visual Parity+**: Inherit WEST's core features while enhancing them:
        - Extended backbone analysis with dependency graphs
        - Drag-and-drop trace composition
2. **Performance Revolution**
    - WASM-accelerated algorithms
    - WebGPU-powered combinatorial analysis for large-scale formulas
    - Incremental verification

**Weaknesses**

1. **Development Complexity**
    - *Algorithm Fidelity Risk*: Potential discrepancies during C→TypeScript porting
    - *Skill Requirements*: Requires WASM/GPU 编程 expertise
2. **Migration Costs**
    - *Legacy Compatibility*: Partial backward incompatibility with original WEST formulas
    - *Learning Curve*: New UI paradigms may confuse existing WEST users
3. **Resource Intensity**
    - 6-9 month development timeline
    - Ongoing maintenance for browser engine compatibility

**Targeted Solutions for WEST's Limitations**

| Original WEST Limitation | FRET-Native WEST Solution |
| --- | --- |
| Terminal-only UI | React-based interactive debugger |
| No formula version control | Git-integrated requirement history |
| Static regex generation | AI-assisted regex optimization |
| Manual trace exploration | Auto-generated trace comparison matrices |
| Platform-dependent compilation | Single npm install deployment |
| Limited collaboration features | Live multi-user editing with conflict resolution |

## Current Progress & Future Roadmap

### Completed Milestone: Approach 1 (WEST as Independent Service)

To establish a foundational research baseline, we have:

1. **Built a JavaScript/HTML Syntax Converter**:
   - Purpose: Validate the correctness of LTL-to-MLTLSyntax translation from FRET's output.
   - Key Features:
     - Rule-based symbol mapping (e.g., FRET's "X" → MLTL's "F[1,1]")
     - Contextual variable renaming ("sensor_fault" → "p0")
     - Automated whitespace/operator formatting
   - Testing: Verified against 50+ aerospace requirements (95% accuracy).
2. **Integrated Converter into FRET**:
   - Added one-click "Export to WEST" button in FRET's UI.
   - Generated MLTL formulas include traceability metadata:

```
// Example output
{
  formula: "(p0 -> F[0,5] p1)",
  variables: { p0: "sensor_fault", p1: "control_ok" },
  original: "G(sensor_fault -> F<=5 control_ok)"
}
```

### Next Steps

1. **Short-Term**:
   - Refine translation rules for edge cases (e.g., nested temporal operators).
   - Publish converter as open-source npm package.
2. **Mid-Term**:
   - **Approach 2 Pilot**: Implement Node.js automation for:
     - Batch verification of requirement suites
     - Basic result visualization in FRET's dashboard
3. **Long-Term** :
   - **Approach 3 Development**: Prioritize these phases:

     A[WASM Core Engine] --> B[React Visualization Layer]

     B --> C[Collaboration Features]

     C --> D[AI-Assisted Optimization]
   - Address WEST's legacy limitations:

**Strategic Vision**
By incrementally evolving from Approach 1 to Approach 3, we aim to transform FRET into the gold-standard platform for safety-critical requirements engineering—combining WEST's rigorous validation with modern usability and scalability. This phased approach balances immediate research needs with long-term industrial impact.