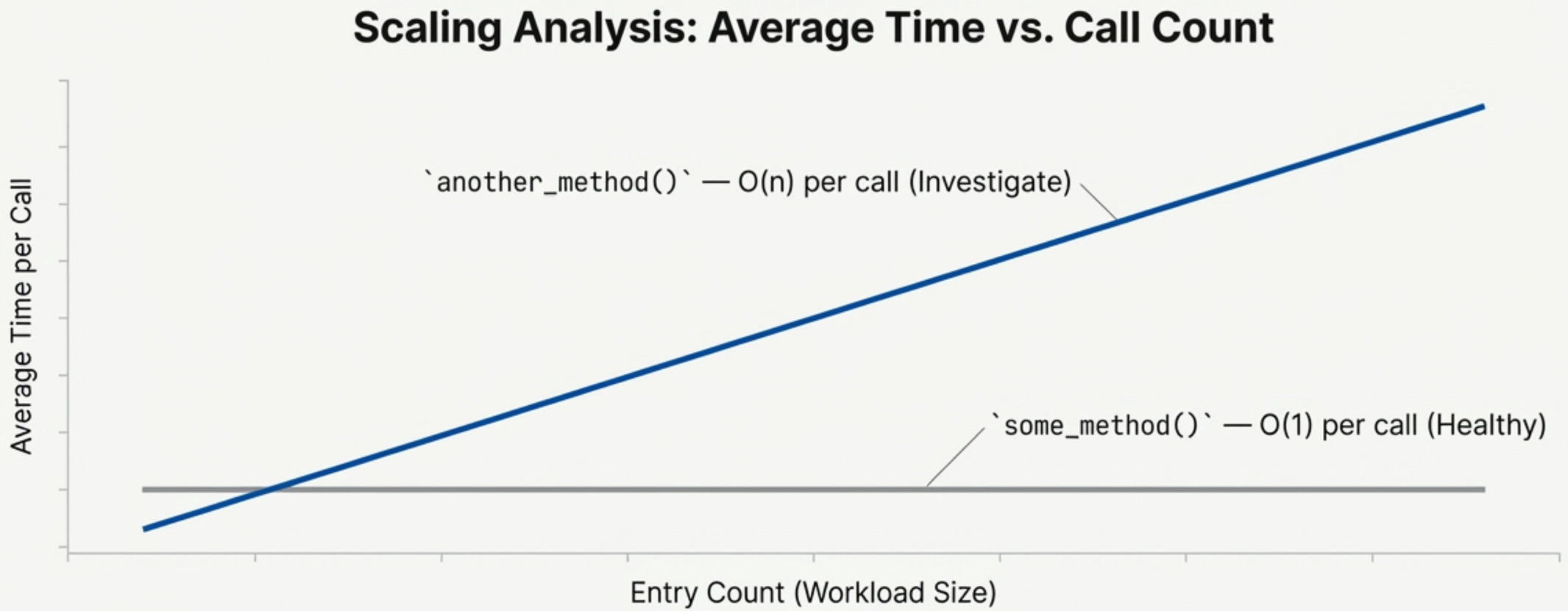
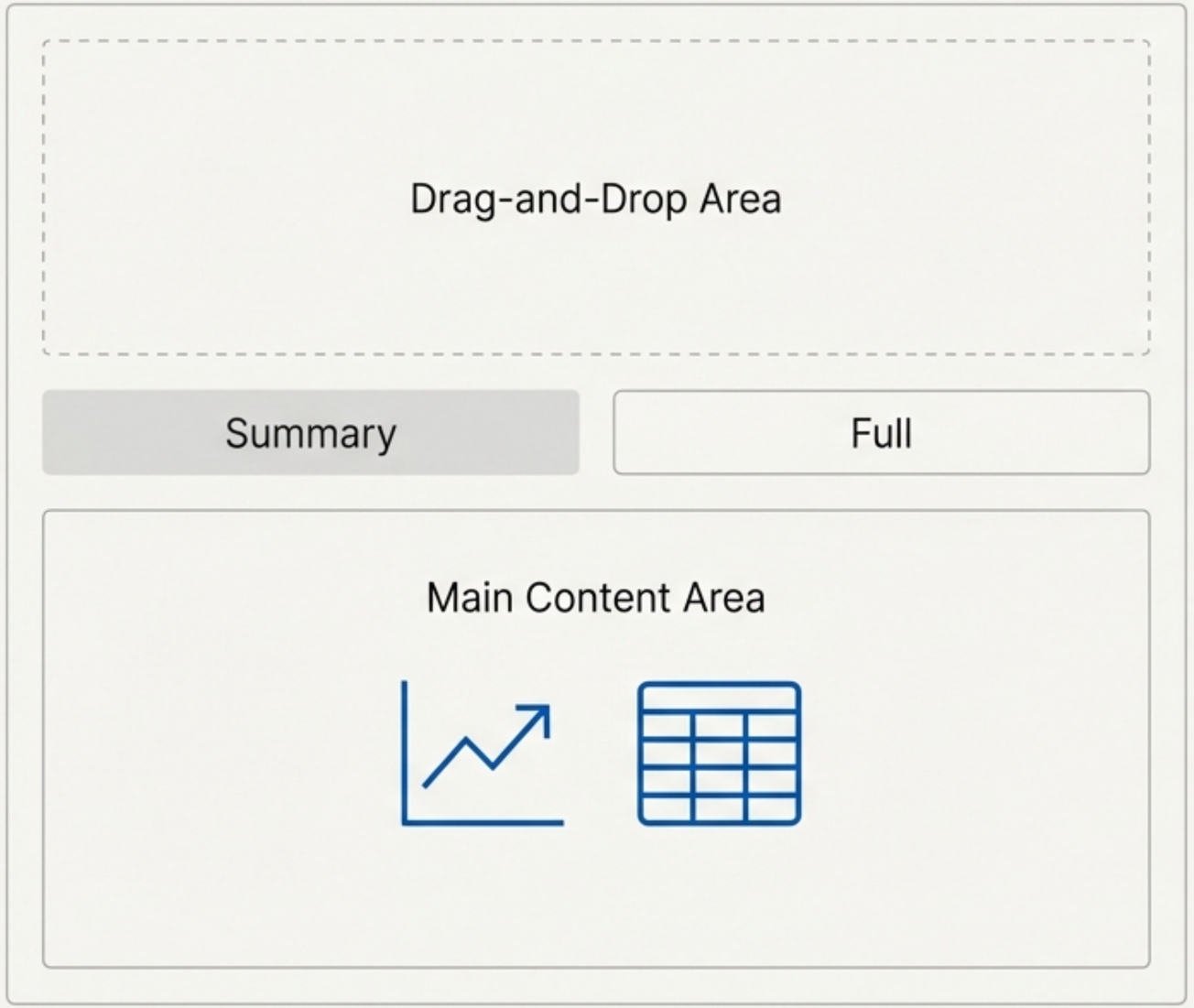
Finding Hidden Algorithmic Complexity in Performance Data



This single visualization reveals algorithmic complexity that aggregate metrics often miss. A flat line indicates healthy, constant-time operations. A rising line signals hidden quadratic behaviour that needs investigation.

The Tool Behind the Insight: The Profile Analyzer



A browser-based interactive visualization tool for performance profiling data.

- **File Loading:** Drag-and-drop loading for JSON profile files.
- **Analysis Modes:** Dual modes for aggregate hotspots (Summary) and per-call traces (Full).
- **Interactive Views:** Bar charts, line charts, comparison tables, and call trees.
- **Exploration:** Click to toggle, hover to highlight, expand to full-screen.
- **Scaling Analysis:** The core diagnostic for detecting O(n) behaviour.

Principles Emerge from Practice, Not Theory

“The most valuable output of this project was not the tool, but the set of user-centric design principles discovered during its development.”

“Each principle is a direct response to a specific user need, transforming direct feedback into a reusable rule for building better data tools.”

Designing for Fluid Interaction

At the moment I need to first figure out which one it is and then go and find the checkbox, where it would be much easier to just click on that line.

Principle 1: Every Visualization Element Should Be Directly Interactive.

Chart lines and table rows are bidirectionally synced. Clicking a line toggles its visibility and updates the corresponding table row.

Users couldn't distinguish between 12+ overlapping lines on a chart.

Principle 10: Hover State Should Clarify, Not Just Decorate.

On hover, the focused line's `strokeWidth` increases to 4, all other lines fade to `opacity 0.25`, and the corresponding table row is highlighted.

Can you make the animation that happens when you reload the data in the graphs to be twice as fast.

Principle 7: Animation Speed Affects Perceived Quality.

Default Recharts animation duration of `1500ms` was reduced to `JetBrains AS 150ms` for a responsive feel. Animations under `200ms` feel instant; over `500ms` feel sluggish.

Building for Cognitive Clarity

“...put that popup window below the graph so that we can still see everything.”

“When we select the line, the UI moves... ideally that hover section... would have the exact same size as when in the selected state.”

“Can you take over the full screen with a small border, making sure that you only use available space.”

Principle 2: Tooltips Must Never Obscure the Data They Describe.

A custom tooltip panel with a fixed position below the chart ensures the user’s focus area is never blocked.

Principle 3: Layout Stability Builds Confidence.

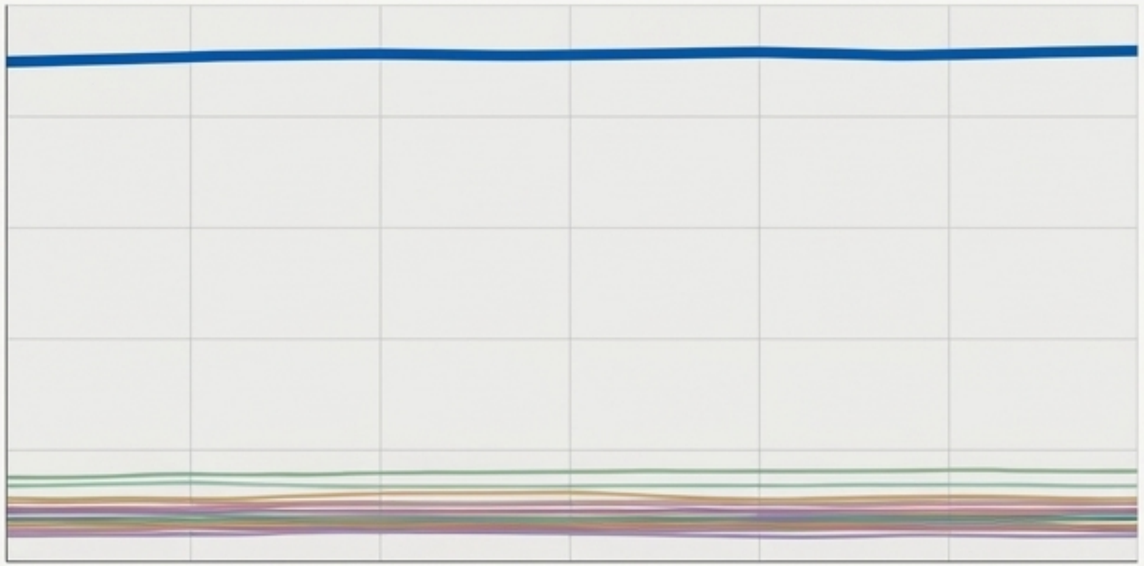
The tooltip container has a fixed height of 72px, with placeholder text when empty. This prevents layout shifts and visual ‘jumping’.

Principle 8: Full-Screen Mode Needs Breathing Room.

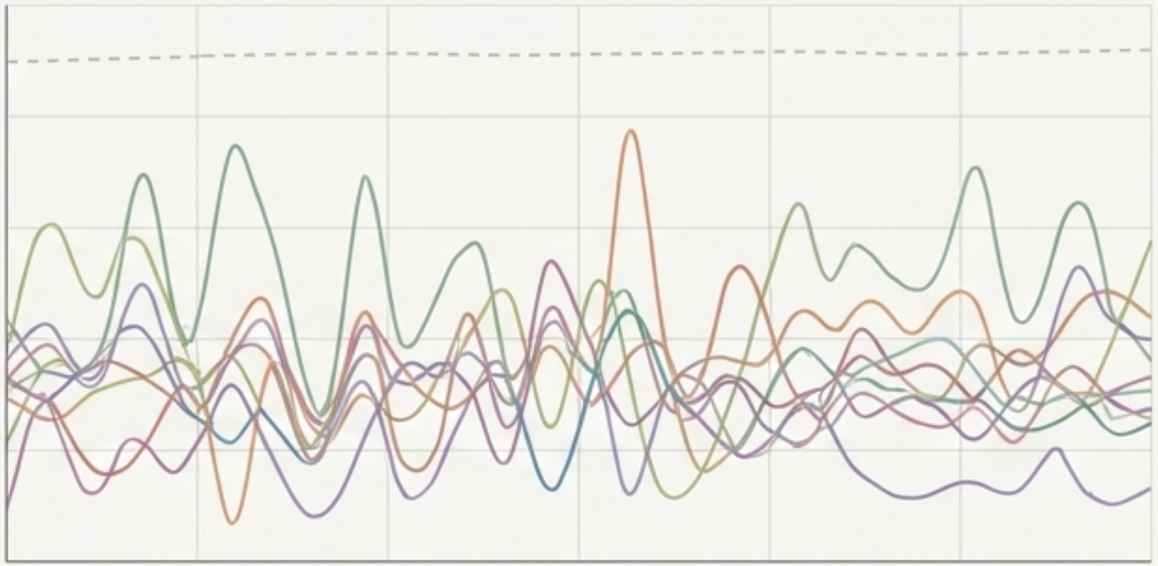
The full-screen overlay uses a fixed position with 24px padding on all edges. Edge-to-edge feels cramped; padding feels intentional.

Empowering Analysis Through Subtraction

Initial View: Signal Obscured



Filtered View: Patterns Revealed



“Root-level pipeline methods dominated scaling charts, compressing all other methods into an unreadable band at the bottom.”

Analysis tools must support iterative hypothesis testing. Let users subtract noise to find the signal.

Principle 6: Dominant Elements Obscure Patterns

Engineering for Real-World Data and Usage

“What about using the entry_count for that comparison?”

“Selection behaviour felt inconsistent between single-item views and multi-comparison views.”

“Files with URL-encoded names (ending in _json) were being silently rejected.”

Principle 4: Use the Data’s Own Metadata.

Switched from parsing fragile filename patterns (e.g., `with_size__30`) to using the `entry_count` field present in all JSON files. Intrinsic metadata is robust; external conventions are not.

Principle 5: Selection Behavior Should Match View Semantics.

The click handler checks the active view to apply the correct logic: ‘radio button’ behaviour for single views, ‘checkbox’ behaviour for comparison views.

Principle 9: File Compatibility Beats Strict Strict Validation.

The file drop handler was updated to accept both `.json` and `_json` extensions. Be liberal in what you accept.

A Simple, Elegant Architecture Makes These Principles Possible

Achieving a highly interactive and responsive user experience does not require a complex build system or framework.

This architecture prioritises simplicity, enabling rapid development and single-file deployment to any static host.

Key Architectural Patterns for Simplicity and Power

Inner Functions for Clean State Access

View components are defined as functions inside the main `App` component. This provides closure access to parent state (e.g., `hoveredMethod`, `colors`) without prop drilling.

```
function App() {
  const [data, setData] = useState(null);
  const [hoveredMethod, setHoveredMethod] =
    useState(null);

  function SummaryView() {
    // Has direct access to 'data' and 'hoveredMethod
    return <Chart data={data} />;
  }

  return <div className="app">{SummaryView()}</div>;
}
```

CSS-in-JS via Template Literal

All styles are embedded in a single <style> tag within the component's JSX. This enables single-file deployment with styles scoped by a simple class naming convention.

```
function ChartSection({ children }) {
  return (
    <div className="chart-section">
      <style>{`
        .chart-section {
          border: 1px solid #333;
          padding: 16px;
        }
      `}</style>
      {children}
    </div>
  );
}
```

Practical and Reusable Recharts Patterns

Custom Tooltip for Layout Stability (Principle 2 & 3)

Disable the default tooltip and render a custom, fixed-height panel below the chart based on hover state.

```
<LineChart onMouseMove={handleMouseMove}>
  <Tooltip content={<div />} cursor={false} />
  {/* ... lines ... */}
</LineChart>
<CustomTooltipPayload payload={hoverPayload} />
```

Fast Animations for Responsiveness (Principle 7)

Override the default animation duration on chart elements for a snappier feel.

```
<Line type="monotone" dataKey="pv" animationDuration={150} />
<Bar dataKey="uv" animationDuration={150} />
```

Interactive Line Styling (Principle 1 & 10)

Use state to dynamically control the `strokeWidth` and `opacity` of lines based on user interaction (hover or click).

```
<Line
  strokeWidth={hoveredMethod === name ? 4 : 2}
  opacity={hoveredMethod && hoveredMethod !== name ? 0.25 : 1}
/>
```

The User-Centric Visualization Playbook

1. Every Visualization Element Should Be Directly Interactive.

2. Tooltips Must Never Obscure the Data They Describe.

3. Layout Stability Builds Confidence.

4. Use the Data's Own Metadata.

5. Selection Behavior Should Match View Semantics.

6. Dominant Elements Obscure Patterns.

7. Animation Speed Affects Perceived Quality.

8. Full-Screen Mode Needs Breathing Room.

9. File Compatibility Beats Strict Validation.

10. Hover State Should Clarify, Not Just Decorate.

Applying the Playbook to Your Own Data

1. **Identify Data Structure**
   Map the fields in your data source to potential visualization needs.

2. **Define Views & Modes**
   Determine the key questions each visualization will answer for the user.

3. **Adopt the Layout**
   Start with a proven structure: header, sidebar, and a main content area with tabs.

4. **Adapt Data Processing**
   Write new extraction and aggregation functions specific to your data format.

5. **Customise Charts**
   Apply the Recharts patterns, mapping them to your processed data.

6. **Preserve Interaction Patterns**
   Implement the core interactions: drag-and-drop, hover, click-to-toggle, and expand-to-fullscreen.

The Foundation: A Minimalist, Modern Stack

Library	Version	Purpose
React	18.x	UI framework
ReactDOM	18.x	DOM rendering
Recharts	2.10.x	Charting library
Babel Standalone	Latest	In-browser JSX transformation

Font	Usage
Space Grotesk	UI text, labels, headers
JetBrains Mono	Data values, code, method names

This entire application is served from a single HTML file with CDN dependencies. No build step required. The focus is on the user experience, not the tooling.