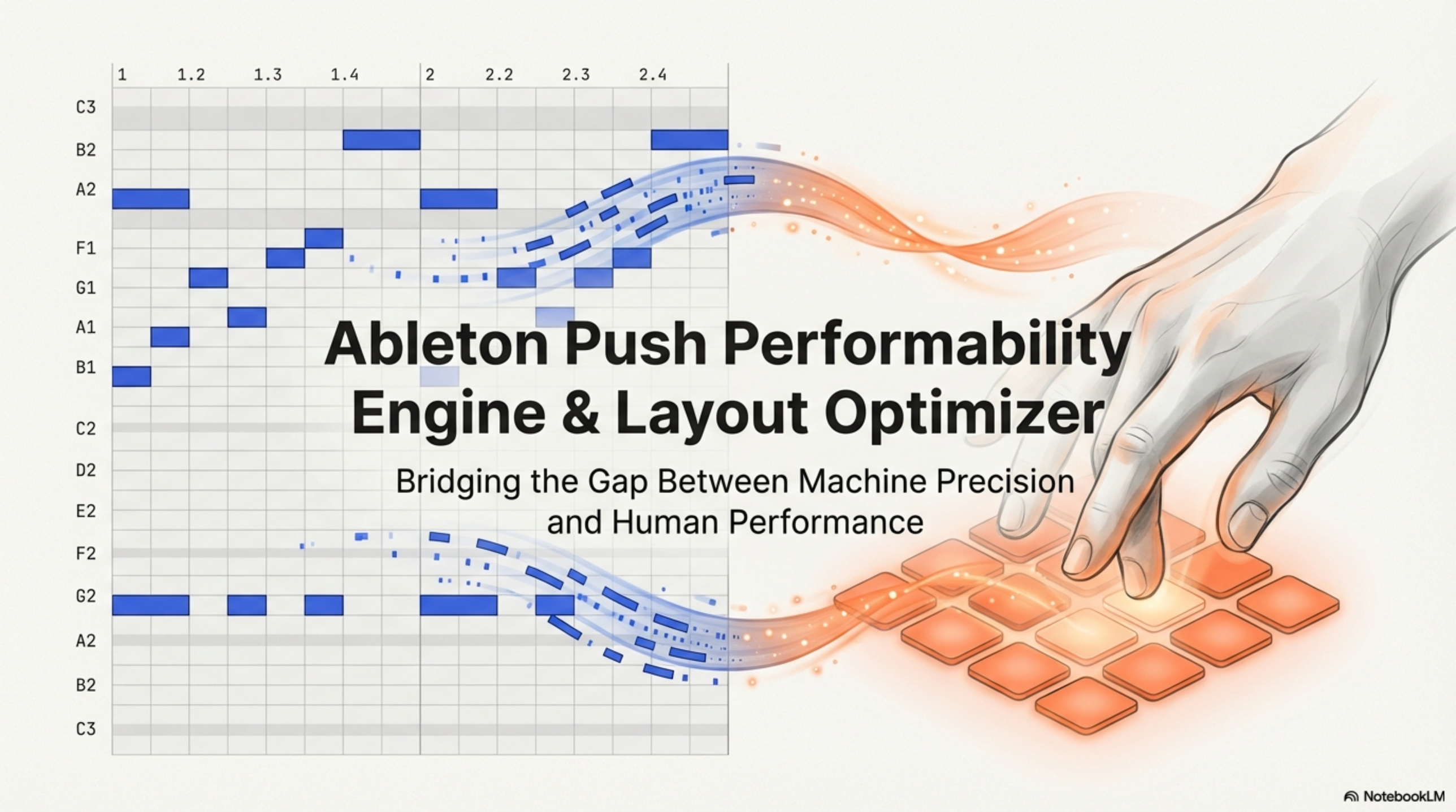


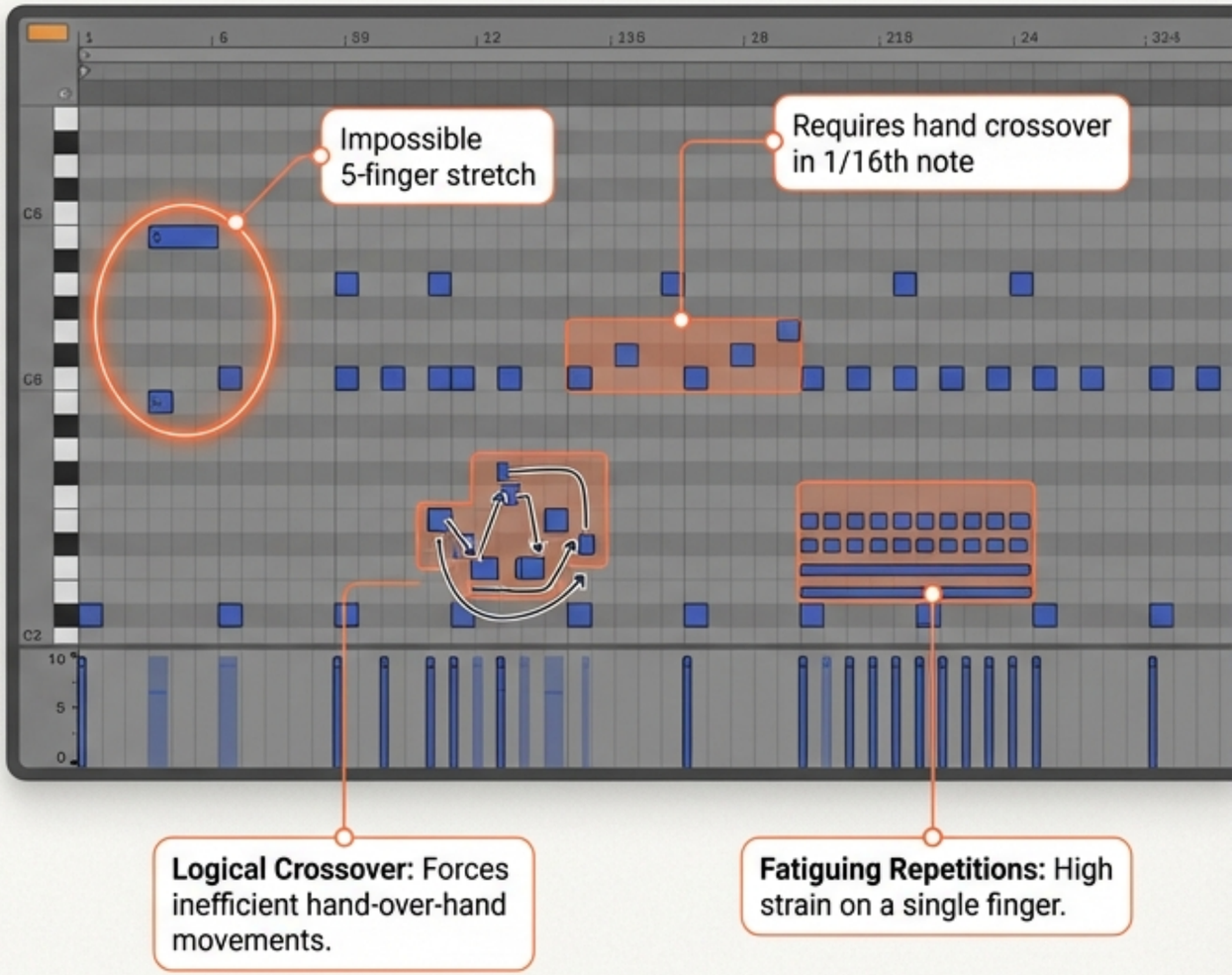
Ableton Push Performability Engine & Layout Optimizer

Bridging the Gap Between Machine Precision and Human Performance

The Musician's Dilemma: Perfect MIDI is Rarely Performable

Producers meticulously craft rhythms using perfectly quantized MIDI. However, this machine precision often translates into physically awkward, unergonomic, or impossible patterns for a human to perform on a grid controller like the Ableton Push.

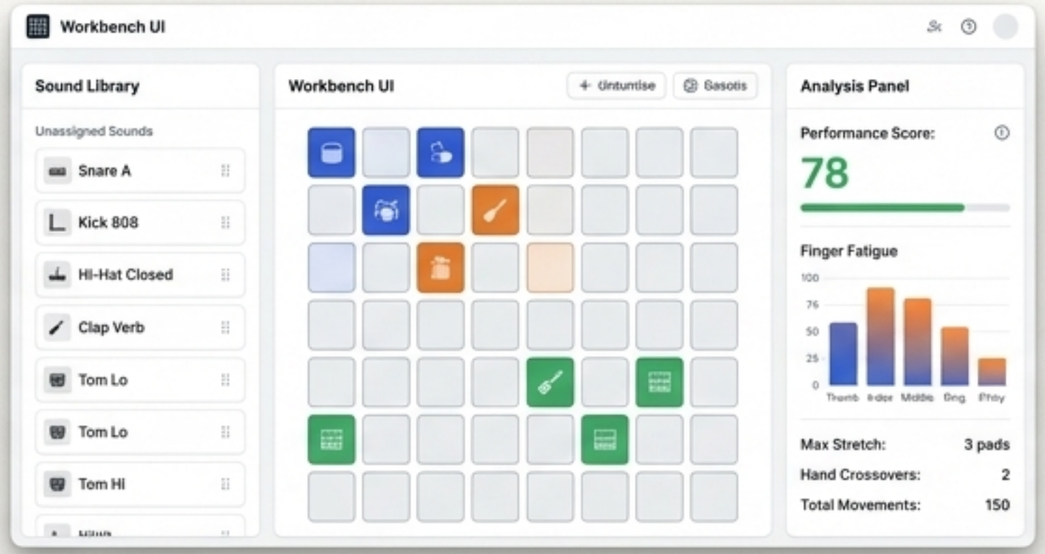
- **Awkward Hand Stretches:** Notes are placed without regard for realistic hand span.
- **Fatiguing Repetitions:** Fast patterns can cause strain on specific fingers.
- **Logical Crossovers:** The layout may force inefficient or impossible hand-over-hand movements.
- **Cognitive Load:** A non-intuitive layout makes the pattern difficult to learn and internalize.



The goal is to transform this 'perfect' data into a performance that respects the biomechanics of the human hand: **movement, stretch, fatigue,** and **playability.**

The Solution: An Intelligent Workbench for Ergonomic Performance Design

A React + TypeScript web application that analyzes MIDI patterns and optimizes both pad layouts and finger assignments for Ableton Push.



Song Portfolio & Dashboard

Manage your projects and Import MIDI.

Interactive Workbench

Design, analyze, and optimize layouts in an 8×8 grid editor.

Biomechanical Engine

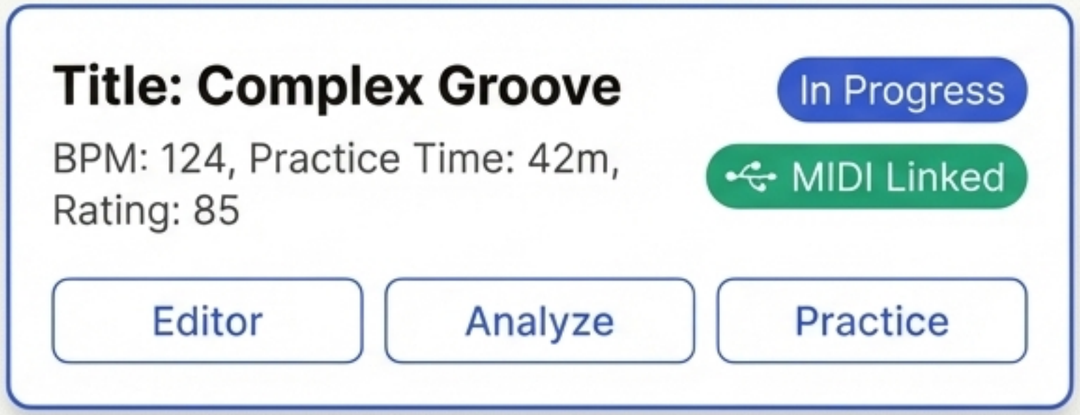
The core intelligence, using multiple solver algorithms (Beam, Genetic, Annealing) to find the most ergonomic fingering.

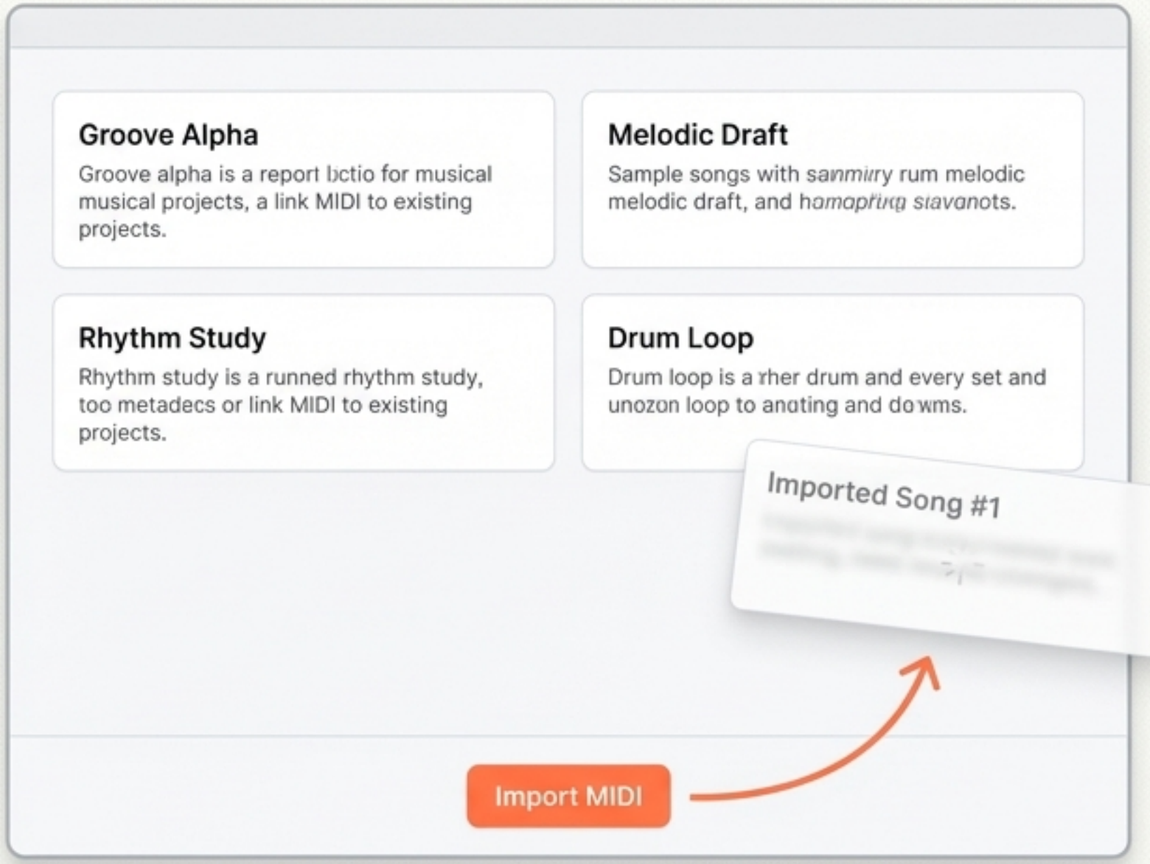
Unified Project State

A robust data model with full undo/redo capabilities for complex session management.

Your Starting Point: The Song Portfolio Dashboard

The dashboard provides a central hub for all musical projects. On first load, it's populated with sample songs. Users can import new MIDI files or link MIDI to existing projects.





MIDI Import & Linking

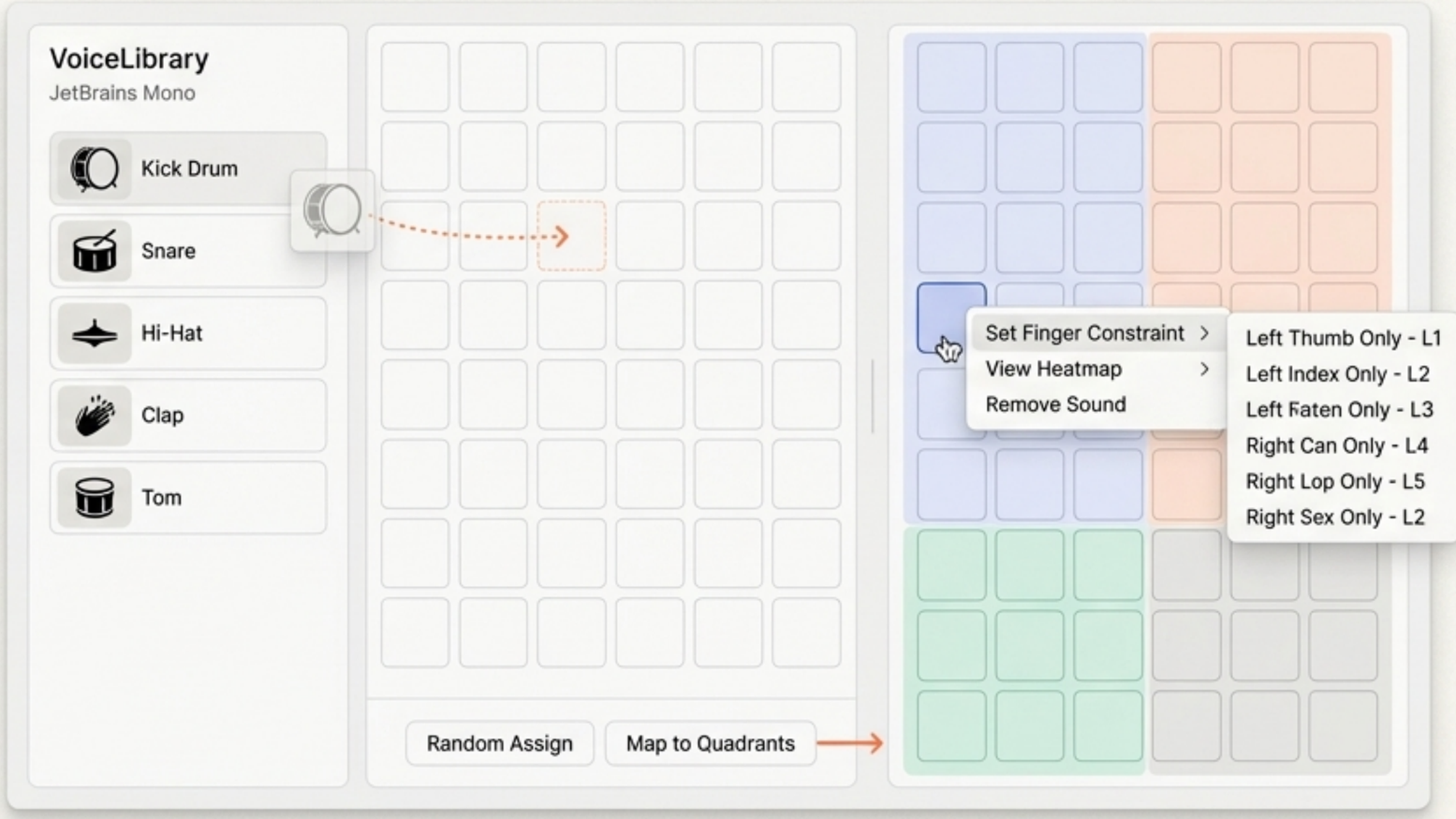
- A simple 'Import MIDI' flow parses the file, converts it to base64 for storage, and derives an initial `ProjectState`.
- The system infers metadata like title, BPM, and key from the file.

The Workbench: An Interactive Canvas for Layout Design

Core Concept: **The Explicit Layout Model**: When MIDI is imported, the 8x8 grid starts empty. All unique sounds (`Voice[]`) are placed in a 'parked sounds' staging area. This empowers the user to consciously design their layout from a clean slate.

Primary Interactions

- **Drag & Drop**: Move sounds from the `VoiceLibrary` to the grid, or from pad to pad to swap positions.
- **Intelligent Assignment**:
  - **Random Assign**: Fills empty pads for quick brainstorming.
  - **Map to Quadrants**: Organizes sounds into logical 4x4 banks.
  - **Clear Grid**: Returns all sounds to the staging area.
- **Pad Context Menu**: Right-click a pad to set finger constraints (e.g., 'Left Thumb Only - L1') or view a reachability heatmap.



The Goal: Move from `layoutMode: 'none'` to `'manual'` or `'optimized'` through deliberate user action.

The Reactive Solver Loop: Instant Biomechanical Feedback

The Workbench features a reactive analysis loop that provides near real-time feedback on the ergonomic quality of a layout as you edit it.

Impact

This creates a fluid, 'what-if' analysis environment where designers can immediately see the ergonomic consequences of their layout decisions without manually re-running tests.

1. User Action

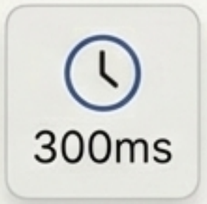
User drags a sound onto a new pad in the `LayoutDesigner`.

2. State Change

`ProjectState` is updated with the new `GridMapping`.

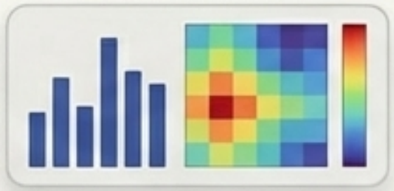
3. Debounced `useEffect` Hook

A watcher detects the change in the layout.



4. Solver Invocation

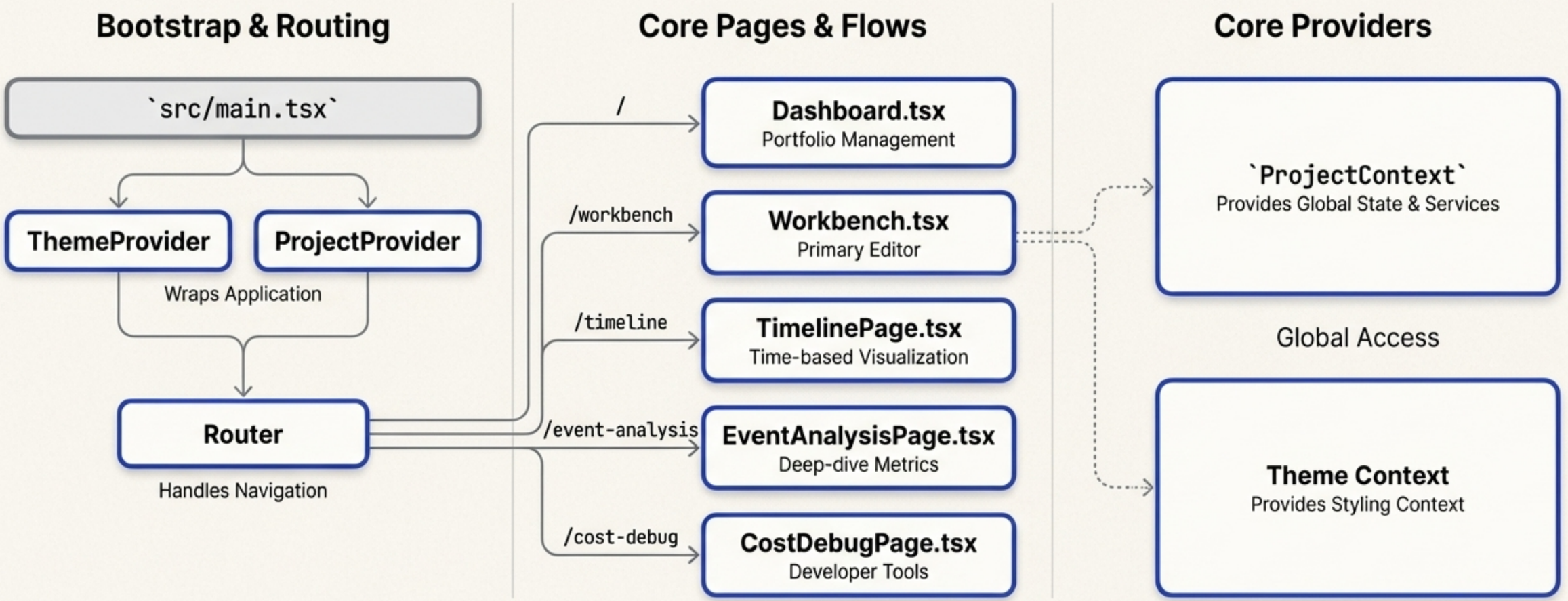
The hook automatically invokes the `BiomechanicalSolver`.



5. UI Update

The `AnalysisPanel` instantly updates with new scores and fatigue maps.

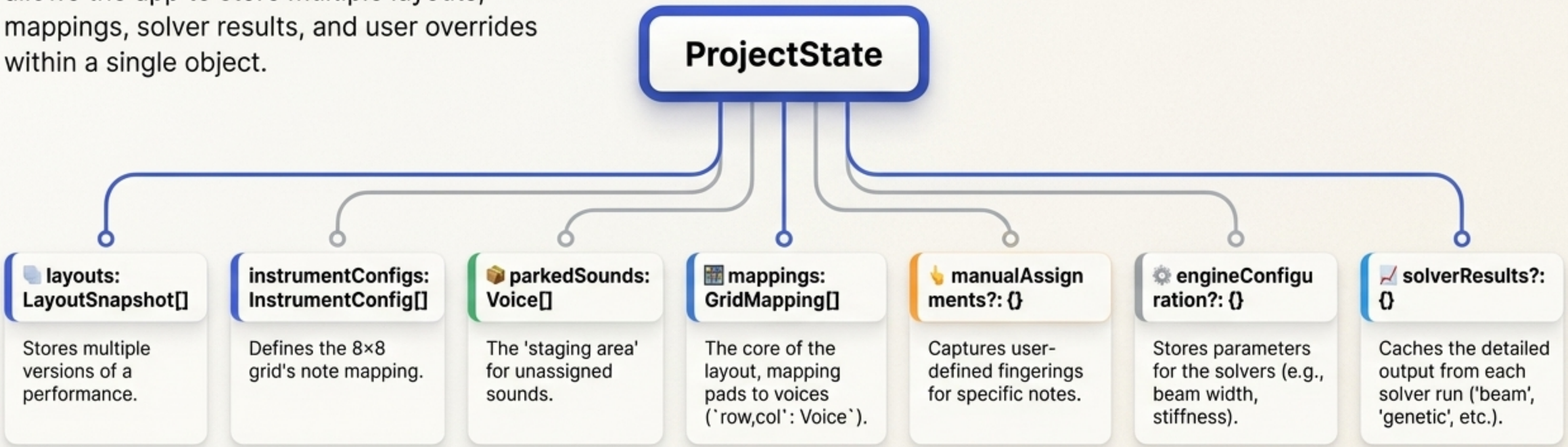
The Architectural Blueprint: A Clean Separation of Concerns



This structure cleanly separates navigation and global state management from the distinct functionalities of portfolio, editing, and analysis.

`ProjectState`: The Unified Source of Truth for Every Performance

`ProjectState` is the central, serializable state container for the entire application. It allows the app to store multiple layouts, mappings, solver results, and user overrides within a single object.



The `ProjectContext` API: Providing Controlled State Management

All interaction with the `ProjectState` is managed through the `ProjectContext` (via the `useProject()` hook), providing a clean and predictable API for components. Solvers are explicitly triggered, not run automatically on every state change.

Accessors

```
projectState
engineResult // derived state
```

Undo/Redo

```
undo()
redo()
canUndo
canRedo
```

Built on a history stack that stores up to 50 previous `ProjectState` snapshots.

State Management

```
setProjectState(stateOrUpdater, skipHistory?)
```

The `skipHistory` flag is crucial for actions like loading a project without polluting the undo stack.

Solver Operations

```
runSolver(type)
optimizeLayout()
setInitialStateFromNeutralPose()
```

These functions are the explicit triggers for the engine's heavy computations.

Inside the Biomechanical Engine: The Core Intelligence

To assign an ergonomic hand + finger combination for every single event in a musical performance, given a specific pad layout.

Note to Pad Mapping

Translate each MIDI `noteNumber` to a physical `(row, col)` coordinate on the grid using the active `GridMapping`.

Moment Grouping

Group simultaneous notes into 'chords' or 'grips' that must be played together.

Candidate Generation

For each moment, generate all valid hand grips, filtering out those that violate constraints.

Cost Evaluation

Score each potential grip sequence based on a detailed ergonomic cost model.

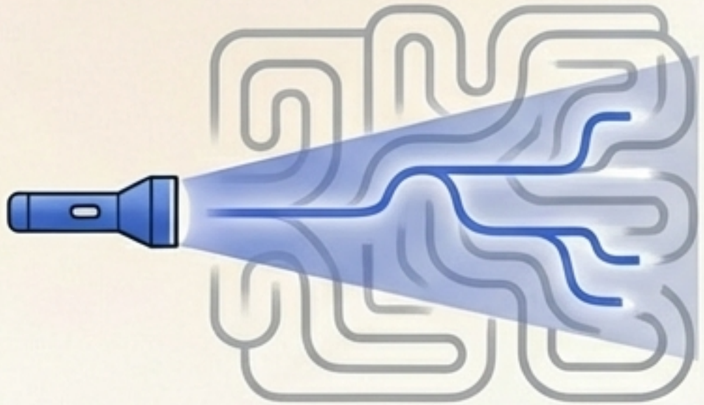
Optimal Path Finding

Use advanced solver strategies to find the best possible sequence of fingerings over the entire performance.

A Trio of Solvers: Finding the Optimal Performance Path

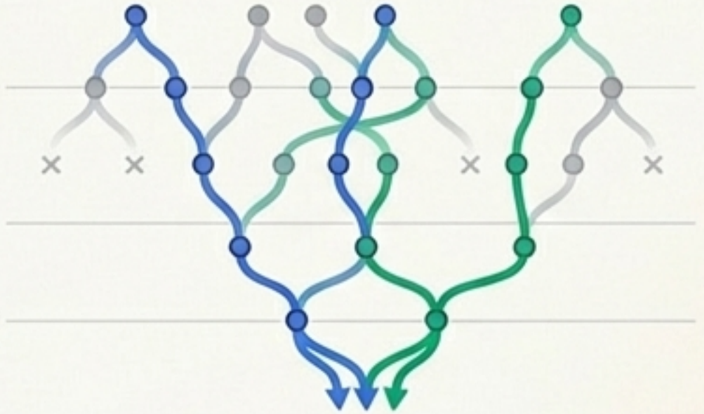
The engine employs multiple solver strategies to navigate the vast search space of possible fingerings, allowing for comparison between different optimization approaches.

Beam Search



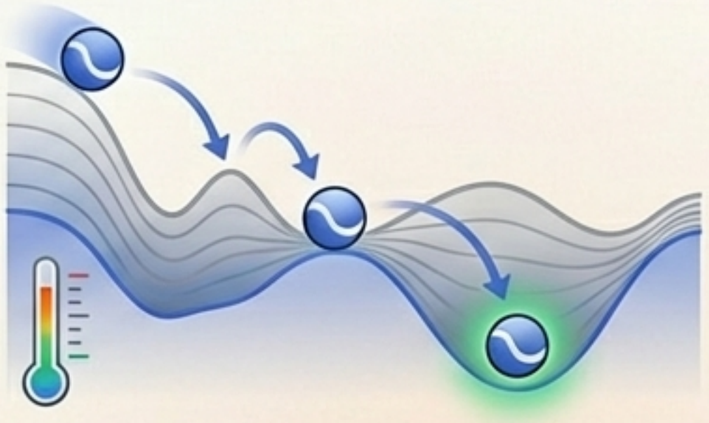
A fast, greedy heuristic that explores a limited number of the "best" options at each step. Good for quick, near-real-time analysis.

Genetic Algorithm



An evolutionary approach that "breeds" populations of solutions, selecting for fitness (lower cost). Excellent for exploring a wide solution space.

Simulated Annealing



A probabilistic technique that intelligently explores solutions, accepting "worse" moves initially to avoid local minima, before "cooling" to a final, optimized layout.

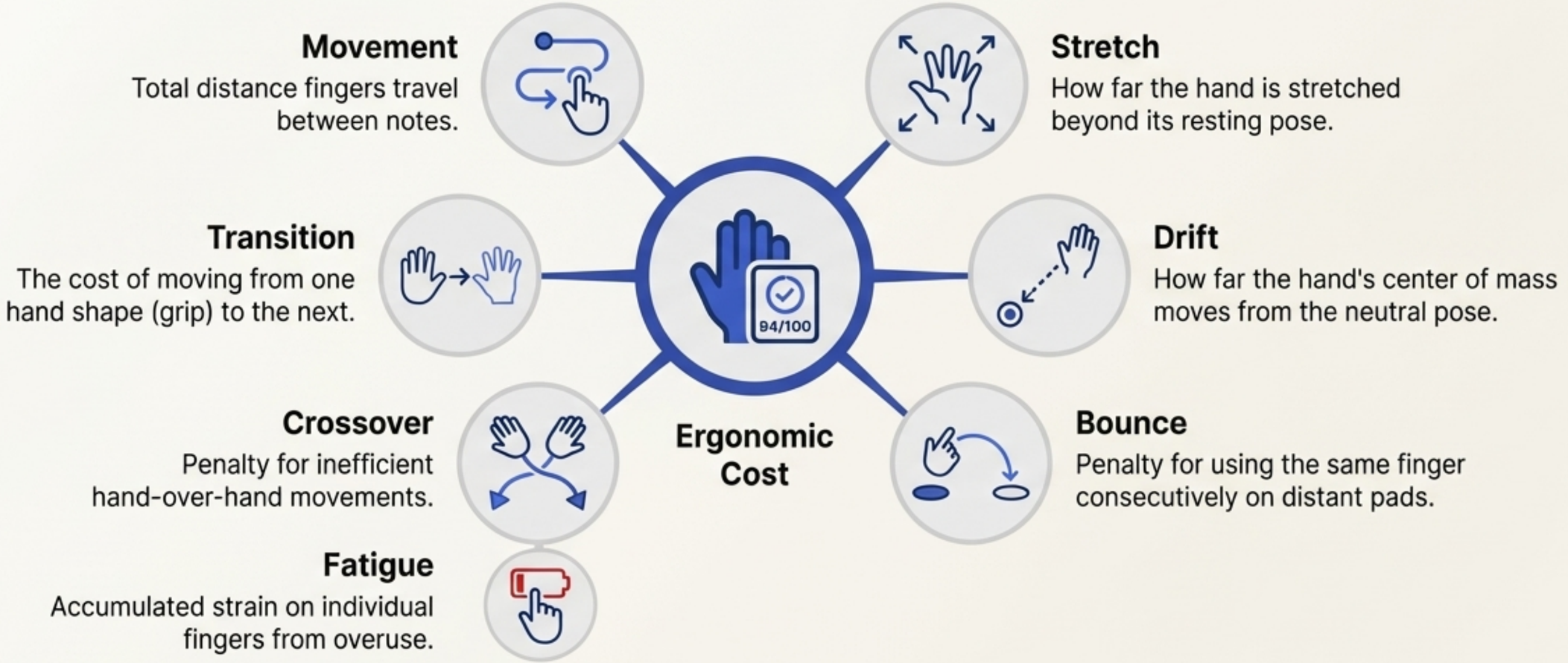
Model Comparison

Total Cost

Lower cost indicates better ergonomic performance.

Deconstructing "Performability": The Ergonomic Cost Components'

The engine's 'score' is not a single number, but a weighted composite of multiple biomechanical cost factors. This detailed model is what allows for nuanced analysis.
The `Neutral Hand Pose` serves as the ergonomic baseline for measuring deviation.



The Complete Data Lifecycle: From MIDI to Analysis-Rich JSON

MIDI Import

An input MIDI file is parsed using `@tonejs/midi`.

Transformation

The parser sorts events, extracts tempo, and generates an initial ProjectState with an empty GridMapping and all sounds parked.

Local Persistence

The Song metadata and the associated ProjectState are saved to LocalStorage, enabling session persistence.

Workbench Interaction

The user modifies the ProjectState in the UI. All work is auto-saved with debouncing.

Project Export

The user can export the entire current ProjectState as a single `push3_project.json` file. This file includes all layouts, mappings, and cached `solverResults`, preserving the full analytical context for sharing or archiving.

The Road Ahead: A Vision for Growth and Integration

The project is built on a solid foundation with a clear roadmap for future enhancements.

Engine & Solver Enhancements

- Improve key detection, extend the `SolverStrategy` pattern, and more deeply integrate tempo into the anatomical stretch model.

Grid Visualization v3 Refactor

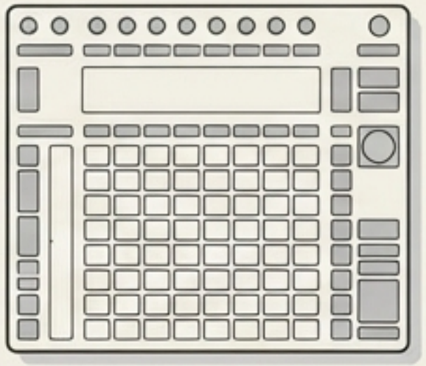
- A major planned refactor to a new layered component model (`BaseGrid`, `PadLayer`, etc.) to enable heatmaps, 'onion skin' views, and improve performance.

Ergonomics & Personalization

- Introduce adjustable hand reach/strength presets and user calibration profiles.

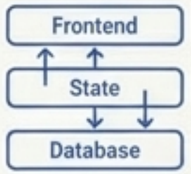
Push 3 Integration & Beyond

- Move from analysis-only to hardware integration, including MIDI output for pad LED feedback and real-time performance capture for live ergonomic analysis.



Project Significance: A Showcase of Core Engineering Competencies

This project is a direct demonstration of:



- **Full-Stack Front-End Architecture**
  Mastery of React & TypeScript, complex shared state management (`ProjectContext` in JetBrains Mono), a robust undo/redo system, and local persistence strategies.

- **Non-Trivial Domain Modeling**
  The ability to translate a complex, abstract domain (hand biomechanics, ergonomics) into a precise, computable data model (`ProjectState`, cost functions in JetBrains Mono).

- **Applied Algorithms & Optimization**
  Practical implementation and application of multiple advanced algorithms (Beam Search, Genetic Algorithms, Simulated Annealing) to solve a real-world optimization problem.

- **UX for Complex Tools**
  Designing an intuitive user experience for a sophisticated tool, featuring concepts like the 'Explicit Layout Model,' a reactive analysis loop, and rich data visualizations.

- **Strategic Roadmap Thinking**
  A clear vision for future development, including plans for refactoring, hardware integration, and feature enhancement, demonstrating long-term project ownership.