Чтобы реализовать этот дашборд, нам потребуется изменить/создать следующие файлы:

1. **App.js**:
   * **Зачем:** Сейчас он рендерит либо TimeSeriesChart, либо ModelProgressVisualizer в правой панели. Нам нужно изменить эту логику, чтобы при активной задаче (activeJobId) он рендерил новый компонент-контейнер для дашборда (назовем его ModelDashboard).
   * **Изменения:** Модифицировать JSX в правой панели для условного рендеринга ModelDashboard, передавая ему необходимые данные (modelProgressData, estimatedRunsForJob).
2. **ModelProgressVisualizer.js**:
   * **Зачем:** Этот файл сейчас содержит *всю* логику отображения прогресса, включая сетку. Мы разделим его на более мелкие, специализированные компоненты.
   * **Изменения:** Мы можем либо полностью **заменить** его новым компонентом ModelDashboard.js, либо **рефакторить** ModelProgressVisualizer.js, чтобы он стал контейнером ModelDashboard, который уже внутри себя будет рендерить новые компоненты (статистику, график, таблицу). Давайте выберем **первый вариант (замена)** для большей чистоты: создадим новый файл ModelDashboard.js. ModelProgressVisualizer.js больше не будет использоваться напрямую в App.js для отображения во время выполнения задачи.
3. **ModelDashboard.js (Новый файл)**:
   * **Зачем:** Будет служить контейнером для всех элементов дашборда в правой панели. Он будет получать данные о прогрессе от App.js и управлять состоянием, специфичным для дашборда (выбранные оси для графика, настройки таблицы, фильтры).
   * **Содержимое:** Будет импортировать и рендерить компоненты для статистики, scatter plot'а и таблицы. Будет содержать логику для расчета сводной статистики и подготовки/фильтрации данных для дочерних компонентов.
4. **JobSummaryStats.js (Новый файл)**:
   * **Зачем:** Специализированный компонент для отображения текстовой сводки (пункт 0 вашего плана).
   * **Содержимое:** Будет принимать данные о количестве моделей (total, processed, valid, invalid-stats, invalid-constraints) как пропсы и рендерить их в виде текста.
5. **ModelScatterPlot.js (Новый файл)**:
   * **Зачем:** Компонент для интерактивного scatter plot'а (пункт 1).
   * **Содержимое:** Будет использовать библиотеку для построения графиков (например, react-plotly.js или Chart.js с плагином scatter). Будет содержать dropdown'ы для выбора осей X/Y, получать отфильтрованные данные моделей от ModelDashboard и отображать их точками соответствующего цвета. Потребуется установка новой зависимости (npm install react-plotly.js plotly.js).
6. **ModelResultsTable.js (Новый файл)**:
   * **Зачем:** Компонент для большой, фильтруемой и сортируемой таблицы (пункт 2).
   * **Содержимое:** Будет использовать библиотеку для таблиц, поддерживающую виртуализацию, сортировку и фильтрацию (например, react-table). Будет содержать UI для выбора видимых колонок, получать отфильтрованные данные от ModelDashboard и отображать их. Потребуется установка новой зависимости (npm install react-table).
7. **CSS Файлы:**
   * **ModelDashboard.css (Новый)**: Стили для общей компоновки дашборда.
   * **JobSummaryStats.css (Новый, опционально)**: Стили для текстовой сводки.
   * **ModelScatterPlot.css (Новый, опционально)**: Стили для контейнера графика и элементов управления осями.
   * **ModelResultsTable.css (Новый)**: Стили для таблицы, фильтров, пагинации (если будет).
   * **ModelProgressVisualizer.css**: Этот файл больше не будет использоваться в контексте дашборда, стили из него нужно будет перенести/адаптировать для новых CSS файлов.

**Порядок действий:**

Предлагаю двигаться в следующем порядке:

1. **App.js**: Внесем изменения для рендеринга нового ModelDashboard.
2. **ModelDashboard.js**: Создадим базовую структуру этого компонента.
3. **JobSummaryStats.js**: Создадим и реализуем компонент для сводной статистики. Интегрируем его в ModelDashboard.
4. **ModelScatterPlot.js**: Создадим и реализуем scatter plot с выбором осей. Интегрируем в ModelDashboard. (Потребуется установка plotly.js).
5. **ModelResultsTable.js**: Создадим и реализуем таблицу с виртуализацией, сортировкой и фильтрацией. Интегрируем в ModelDashboard. (Потребуется установка react-table).
6. **CSS**: Создадим/обновим соответствующие CSS файлы для стилизации.

SUBPLAN #2  
  
**App.css**: Здесь задается основная структура с левой и правой панелями. Нам нужно убедиться, что правая панель (.right-pane) настроена так, чтобы ее *содержимое* могло скроллиться, если оно не помещается.

1. **ModelDashboard.css**: Этот файл стилизует сам дашборд, который находится *внутри* правой панели. Здесь мы будем настраивать размеры секций графика (.scatter-plot-section) и таблицы (.results-table-section), чтобы график стал большим и квадратным, а таблица получила достаточно места.
2. **ModelResultsTable.js**: Нужно проверить и, если необходимо, установить значение по умолчанию для пагинации (pageSize) равным 30.
3. **ModelResultsTable.css**: Возможно, потребуются небольшие корректировки, чтобы таблица корректно отображалась в своем контейнере и заголовок "залипал" при скролле внутри таблицы.
4. **ModelScatterPlot.js**: Возможно, нужно будет убедиться, что компонент Plotly настроен на использование 100% ширины и высоты своего контейнера.
5. **App.js / ModelDashboard.js**: Возможно, потребуются минимальные изменения в JSX-структуре, если правки CSS этого потребуют (например, убрать лишние обертки).