System Description

1. System Overview

The proposed system provides an interactive visualization platform for exploring attribution information in sequence generation models. It employs a job-based API architecture with Inseq as the backend attribution engine, and features a reactive frontend for intuitive interaction with attribution data.

2. System Architecture

2.1 Backend Pipeline Formalization

Let us define the core components of our system:

- $\mathcal{M} = m_1, m_2, \ldots, m_n$: Set of available sequence generation models
- $\mathcal{F} = f_1, f_2, \ldots, f_k$: Set of attribution methods, including:
  - Gradient-based methods: Integrated Gradients, DeepLIFT, GradientSHAP, etc.
  - Internals-based methods: Attention Weights
  - Perturbation-based methods: Occlusion, LIME, ReAGent
- $\mathcal{I}$: Set of possible input texts
- $\mathcal{O}$: Set of options for attribution methods (e.g., `attribute_target`, `step_scores`)

A job J is defined as: $J = (\mathcal{M}', \mathcal{F}', i, o)$, where:

- $\mathcal{M}' \subseteq \mathcal{M}$ is the subset of selected models
- $\mathcal{F}' \subseteq \mathcal{F}$ is the subset of selected attribution methods
- $i \in \mathcal{I}$ is a single input text
- $o \in \mathcal{O}$ is a set of options that applies to all model-method combinations

The job then processes all combinations in the Cartesian product: $\mathcal{M}' \times \mathcal{F}'$

For each tuple $(m, f) \in \mathcal{M}' \times \mathcal{F}'$, the backend uses Inseq to:

1. Load the model: `attribution_model = inseq.load_model(m, f)`
2. Compute attributions:

```
attributions = attribution_model.attribute(
    i,  # Input text string
    attribute_target=o.get('attribute_target', False),
    step_scores=o.get('step_scores', []),
    output_step_attributions=o.get('output_step_attributions', False),
    **{k: v for k, v in o.items() if k not in ['attribute_target', 'step_scores',
'output_step_attributions']}
)
```

3. Process results and store in the job's result collection

The system leverages Inseq's built-in attribution capabilities while providing a job-based wrapper for parallel processing, long-running operations, and interactive visualizations.

2.2 Job-based API Structure

The API consists of the following endpoints:

1. **Job Creation**:

```
POST /api/jobs
{
  "models": ["m_1", "m_2", ...],
  "methods": ["f_1", "f_2", ...],
  "input": "input text",
  "options": {"option_1": value_1, ...}
}
```

Returns a unique job ID: `job_id`

If multiple inputs need processing, multiple job creation requests should be made, one for each input.

2. **Job Status**:

```
GET /api/jobs/{job_id}/status
```

Returns the job status: `{"status": "pending"|"processing"|"completed"|"failed"}`

3. **Job Results**:

```
GET /api/jobs/{job_id}/results
```

Returns the complete results when job is completed

4. **WebSocket Notification**:

```
WS /api/ws/jobs/{job_id}
```

Sends real-time updates on job status changes

2.3 Result Data Structure

The complete job result includes the job-wide input, options, and individual results for each model-method combination:

```
{
  "job_id": "unique_job_id",
  "input": {
    "text": "original input text",
    "tokens": ["x_1", "x_2", ..., "x_n"]
  },
  "options": {
    "attribute_target": true,
    "step_scores": ["probability", "entropy"],
    ...
  },
  "results": [...]
}
```

For each tuple (m, f) in the results array, the API returns:

```
{
  "model": "m_i",
  "method": "f_j",
  "output": {
    "text": "generated output text",
    "tokens": ["y_1", "y_2", ..., "y_t"]
  },
```

```
  "attribution": {
    "raw": [
      [A_11, A_12, ..., A_1t],
      [A_21, A_22, ..., A_2t],
      ...
      [A_n1, A_n2, ..., A_nt]
    ],
    "aggregated": {
      "token_importance": {
        "input": [I(x_1), I(x_2), ..., I(x_n)],
        "output": [I(y_1), I(y_2), ..., I(y_t)]
      }
    }
  },
  "step_scores": {
    "probability": [p_1, p_2, ..., p_t],
    "entropy": [e_1, e_2, ..., e_t]
  }
}
```

This structure includes pre-computed token importance scores to facilitate immediate visualization in the frontend, while still providing the raw attribution matrix for interactive explorations.

3. Mathematical Formalization of Attribution Methods

3.1 Attribution Function

An attribution function ϕ maps an input-output pair to a matrix of importance scores:

$\phi_f^m(X, Y, o) = \mathbf{A}^{m,f,X,Y} \in \mathbb{R}^{n \times t}$

where:

- $X = (x_1, x_2, \ldots, x_n)$ is the input token sequence
- $Y = (y_1, y_2, \ldots, y_t)$ is the output token sequence
- $\mathbf{A}_{i,j}^{m,f,X,Y}$ represents the attribution score of input token x_i for output token y_j
- o represents job-wide attribution options that are applied consistently across all method calls

3.2 Aggregation Functions

While Inseq provides various aggregation methods for attribution scores on the backend, our interactive frontend requires efficient client-side aggregation to enable responsive visualizations. We formally define the following key aggregation functions that must be implemented in the frontend:

1. **Input-centered aggregation** (when hovering over input token x_k): $\mathbf{A}_{x_k,c}(j) = \sum_{i=\max(1,k-\lfloor c/2 \rfloor)}^{\min(n,k+\lfloor c/2 \rfloor)} \mathbf{A}_{i,j}$

   This function aggregates attribution scores within a context window of size c centered around input token x_k, showing how a group of input tokens influences each output token.
2. **Output-centered aggregation** (when hovering over output token y_k): $\mathbf{A}_{y_k,c}(i) = \sum_{j=\max(1,k-\lfloor c/2 \rfloor)}^{\min(t,k+\lfloor c/2 \rfloor)} \mathbf{A}_{i,j}$

   This function aggregates attribution scores within a context window of size c centered around output token y_k, showing how each input token influences a group of output tokens.
3. **Token importance aggregation** (for default visualization): $I(x_i) = \sum_{j=1}^{t} \mathbf{A}_{i,j} \quad \text{and} \quad I(y_j) = \sum_{i=1}^{n} \mathbf{A}_{i,j}$

   This function computes the overall importance of each token when no hover interactions are active.
4. **Dimensional aggregation** (for multi-dimensional attribution methods): $\mathbf{A}_{i,j} = \text{aggregate}(\mathbf{A}_{i,j,1}, \mathbf{A}_{i,j,2}, \ldots, \mathbf{A}_{i,j,d})$

   Where the aggregation function can be:

- L2 Norm: $\sqrt{\sum_{k=1}^{d} \mathbf{A}_{i,j,k}^2}$
- Sum: $\sum_{k=1}^{d} \mathbf{A}_{i,j,k}$
- Mean: $\frac{1}{d} \sum_{k=1}^{d} \mathbf{A}_{i,j,k}$

5. **Multi-method aggregation** (when multiple methods are active): $\mathbf{A}_{i,j}^{\text{combined}} = \frac{1}{|F'|} \sum_{f \in F'} \mathbf{A}_{i,j}^{f}$

   This function combines attribution scores from multiple selected attribution methods.

3.2.1 Frontend Implementation Strategy

To ensure responsive interactions, the system uses a hybrid approach for aggregation:

1. **Pre-computed aggregations**: The backend pre-computes and includes token importance aggregations $I(x_i)$ and $I(y_j)$ in the API response for immediate rendering of the default view.
2. **On-demand client-side aggregation**: Context window and multi-method aggregations are computed on the client side in real-time during user interactions.
3. **Throttled computations**: For larger attribution matrices, aggregation operations are throttled and potentially offloaded to Web Workers to maintain UI responsiveness.
4. **Progressive loading**: For very large attribution matrices, the system uses a progressive loading approach where only the visible portion of the matrix is processed initially, with additional computations performed as the user interacts with the visualization.

4. Frontend Views and Flow

The frontend application is built with React and TypeScript to ensure type safety and component reusability. The system comprises three primary views that form a coherent user flow.

4.1 Job Creation View

This view enables users to create new attribution jobs with the following components:

1. **Model Selection**: Multi-select interface for choosing models from $\mathcal{M}$
2. **Method Selection**: Multi-select interface for choosing attribution methods from $\mathcal{F}$
3. **Input Text Area**: Text field for entering input prompt(s)
4. **Options Configuration**: Interface for setting attribution options o
5. **Template Selection**: Drop-down for selecting prompt templates
6. **Submit Button**: Creates a new job with the selected parameters

Upon submission, the system creates a job with a unique ID and redirects to the Job Overview, automatically showing the "Running Jobs" section with the newly created job.

4.2 Job Overview

This centralized dashboard provides a comprehensive view of all jobs in the system, divided into two main sections:

1. **Running Jobs Table**:
   - Job ID
   - Creation timestamp
   - Selected models/methods count
   - Progress indicator with percentage and estimated time remaining
   - Cancel button with confirmation dialog
   - Real-time status updates via WebSocket connection
2. **Completed Jobs Table**:
   - Job ID
   - Input text (truncated)
   - Completion timestamp

- Selected models/methods with visual indicators
- Success/failure status with error details on hover
- View button to navigate to the Attribution Dashboard
- Filter and sort controls to manage large numbers of jobs

When a job completes, it automatically moves from the "Running" to the "Completed" section with a visual transition effect. Users can click on a completed job to navigate to the Attribution Dashboard for that specific job.

4.3 Attribution Dashboard

For a completed job, this view visualizes the attribution data for tuples (m, F', i) where $F' \subseteq F$ and i is the job's input. This is the core interactive component of the system where users explore attribution relationships.

4.3.1 Header Controls

- **Model selector dropdown**: Choose from available models in the job
- **Method toggle buttons**: Enable/disable attribution methods with visual indicators showing the method-specific color coding
- **Context window slider**: Adjustable parameter $c \in [1, \lfloor \min(n, t)/2 \rfloor]$ with immediate visual feedback
- **Split view toggle**: Enable comparison between two models (if multiple models are available in the job)
- **Category filter dropdown**: Filter tokens by linguistic categories (requires integration with external NLP service)
- **Aggregation method selector**: Choose between available aggregation functions (L2Norm, Sum, Mean, etc.)
- **Export options**: Save visualizations or raw data for external analysis

4.3.2 Main Visualization Area

The main visualization consists of an interactive attribution heatmap with the following elements:

- **Input text display**: Shows the source text with clear token boundaries
- **Generated output display**: Shows the model-generated text with token boundaries
- **Attribution visualization**:
  - Default view: Each token is colored based on its overall importance (using token importance aggregation)
  - Hover interactions: When a user hovers over a token, related tokens are highlighted with intensity corresponding to attribution strength
  - Multi-color visualization: When multiple methods are active, each method's results appear in a different color
  - Context window highlighting: When context window size > 1, groups of tokens are considered together for highlighting

4.3.3 Interaction Flow

1. **Initial state**: The dashboard loads showing the default view with token importances for the first selected model and method
2. **Method selection**: User toggles methods on/off to compare different attribution approaches
3. **Token exploration**: User hovers over tokens to see attribution relationships
4. **Context adjustment**: User adjusts the context window slider to explore broader token relationships
5. **Model comparison**: User enables split view to compare attribution patterns between models
6. **Follow-up creation**: User initiates a follow-up conversation turn using the interface at the bottom

4.3.4 Follow-up Job Creation Interface

At the bottom of the dashboard, users can create a new job that builds on the current conversation:

- **Input field**: For entering the next turn in the conversation
- **Conversation history preview**: Shows how previous turns (input and output) will be incorporated
- **Model/method selection**: Defaults to the current selection but can be modified

- **Submit button**: Creates a new job and navigates to the Job Overview

This creates a linked job chain that allows users to follow attribution patterns through a multi-turn conversation.

5. Visualization and Interaction Methods

5.1 Interactive Attribution Visualization

The core of the system is an interactive visualization that displays the attribution relationship between input and output tokens. While it builds upon the foundation provided by Inseq's visualization capabilities, it extends them with real-time interactivity and additional features specifically designed for exploring token-level attributions.

5.1.1 Visual Encoding Principles

1. **Token-Level Attribution Visualization**:

   For each attribution score $\mathbf{A}_{i,j}$ (importance of input token i for output token j), the visualization applies a visual encoding:

   $$\text{intensity}(\mathbf{A}_{i,j}) = \text{normalize}(\mathbf{A}_{i,j}) \in [0, 1]$$

   where normalize applies min-max normalization: $\text{normalize}(\mathbf{A}_{i,j}) = \frac{\mathbf{A}_{i,j} - \min(\mathbf{A})}{\max(\mathbf{A}) - \min(\mathbf{A})}$

   This normalized value is then mapped to visual properties such as background color opacity.
2. **Color Encoding Scheme**:
   - **Token importance**: Base color intensity represents the normalized importance value
   - **Method differentiation**: Each attribution method $f \in F'$ is assigned a distinct color from a perceptually distinct palette
   - **Combinatorial visualization**: When multiple methods are active, their colors are visually differentiated to allow comparison
3. **Visual Hierarchy**:
   - **Background color**: Represents the overall token importance or the specific attribution score during hover
   - **Border style**: Indicates selection state or hover focus
   - **Typography**: Ensures readability while maintaining the visual encoding

5.1.2 Interactive Elements

The visualization incorporates several interactive elements to support exploration of attribution data:

1. **Token Hovering**: When a user hovers over a token, the system:
   - Highlights the hovered token with a distinct border
   - Computes attribution relationships to all other tokens using the appropriate aggregation function
   - Applies visual encoding to all related tokens based on attribution strength
   - Displays additional information in a tooltip (e.g., exact attribution values)
2. **Context Window Control**: A slider that controls the context window size parameter c:
   - Minimum value: 1 (single token)
   - Maximum value: $\lfloor \min(n, t)/2 \rfloor$ (half the length of the shorter sequence)
   - As the user adjusts this value, the system immediately updates the visualization to aggregate attributions across the specified window size
3. **Method Selection**: Toggle buttons for each available attribution method:
   - Each method can be independently enabled or disabled
   - Visual feedback indicates active methods
   - When multiple methods are active, the system applies a multi-method aggregation to compute the combined visualization
4. **Model Comparison**: A split view toggle that enables comparison between models:

- Divides the visualization area vertically
- Shows results from different models side by side
- Maintains synchronized hover interactions between views

5.1.3 Default View

When no interactions are active, the system displays an overview visualization based on token importance aggregation:

- Each input token x_i is colored based on its overall importance $I(x_i) = \sum_{j=1}^{t} \mathbf{A}_{i,j}$
- Each output token y_j is colored based on its overall importance $I(y_j) = \sum_{i=1}^{n} \mathbf{A}_{i,j}$

This provides an immediate visual summary of the most influential tokens in both the input and output sequences.

5.1.4 Interaction Flow and State Management

The visualization maintains several key state variables:

- `hoveredToken` : The currently hovered token, if any
- `contextWindowSize` : Current setting for context window aggregation
- `activeMethods` : Set of currently active attribution methods
- `splitViewEnabled` : Whether comparison mode is active
- `selectedModels` : Models currently being visualized

When these state variables change, the system computes the appropriate aggregation functions and updates the visualization accordingly. For performance reasons, computation-intensive operations use:

- Memoization of intermediate results
- Throttled updates for rapid interactions
- Progressive loading for large attribution matrices

5.1.5 Technical Implementation Considerations

The frontend implementation should prioritize:

1. **Rendering Performance**: Efficient DOM updates and CSS transitions for smooth interactions
2. **Computation Efficiency**: Optimized aggregation algorithms that scale to large attribution matrices
3. **Responsive Design**: Adaptability to different screen sizes and device capabilities
4. **Accessibility**: Appropriate color choices, keyboard navigation, and alternative information displays

5.1.6 Advanced Visualization Features

In addition to the core functionality, the system supports several advanced features:

1. **Token Grouping**: Automatic identification and visual grouping of semantically related tokens
2. **Threshold Filtering**: Controls to show only attributions above a certain strength threshold
3. **Step Score Integration**: Visualization of additional metrics like token probability alongside attribution scores
4. **Pattern Highlighting**: Automatic identification of notable attribution patterns (e.g., self-attention, cross-attention)
5. **Temporal Comparison**: When viewing a chain of follow-up jobs, visualization of how attribution patterns evolve over conversation turns

5.2 Model Comparison View

The split view comparison feature enables users to directly compare attribution patterns between different models for the same input text and attribution methods. This is particularly valuable for understanding how different model architectures or sizes process the same input.

5.2.1 Split View Layout

When the split view toggle is activated (only available when multiple models have been included in the job), the comparison view:

1. Divides the visualization area vertically into two equal sections
2. Displays model m_1 results in the top half
3. Displays model m_2 results in the bottom half
4. Shows model metadata (name, size, type) in each section header

If a job only contains a single model, the split view toggle will be disabled and appear grayed out to provide visual feedback to the user that this feature requires multiple models.

5.2.2 Synchronized Interactions

To facilitate direct comparison, the split view implements synchronized interactions:

1. **Coordinated Hovering**: When a user hovers over a token in either view:
   - Both views update to highlight attribution relationships
   - The same token is visually emphasized in both views when possible
   - Differences in attribution patterns become immediately apparent
2. **Shared Controls**: The context window slider and method toggles affect both views simultaneously
3. **Differential Highlighting**: The system can optionally highlight the differences between models by:
   - Computing a differential attribution matrix $\mathbf{A}_{i,j}^{\text{diff}} = \mathbf{A}_{i,j}^{m_1} - \mathbf{A}_{i,j}^{m_2}$
   - Using a divergent color scale to show where models agree or disagree
4. **Output Comparison**: When models generate different outputs for the same input:
   - Token alignment is maintained where possible
   - Differences in generated text are visually highlighted
   - Attribution patterns can be compared despite textual differences

5.3 Conversational Follow-up System

A key feature of the system is the ability to trace attribution patterns through multi-turn conversations, enabling analysis of how context influences model outputs over time.

5.3.1 Follow-up Job Creation Interface

For creating a conversation follow-up:

1. At the bottom of the attribution dashboard, the system provides:
   - A text input field for the user's next conversational turn
   - Preview of how previous turns will be incorporated
   - Option to maintain the same models and attribution methods or modify them
2. When submitted, the system:
   - Creates a new job with the constructed input
   - Redirects to the job overview with the new job highlighted
   - Establishes a link between the original and follow-up jobs

5.3.2 Mathematical Formalization of Conversational Context

Let $X_1 = (x_1^1, x_2^1, \ldots, x_n^1)$ be the first input sequence Let $Y_1 = (y_1^1, y_2^1, \ldots, y_t^1)$ be the first output sequence Let $X_{\text{new}} = (x_1^{\text{new}}, x_2^{\text{new}}, \ldots, x_p^{\text{new}})$ be the new user input

The follow-up input is constructed as: $X_2 = (x_1^1, x_2^1, \ldots, x_n^1, y_1^1, y_2^1, \ldots, y_t^1, x_1^{\text{new}}, x_2^{\text{new}}, \ldots, x_p^{\text{new}})$

For multi-turn conversations with k turns, this generalizes to:

$X_k = (X_1, Y_1, X_2, Y_2, \ldots, X_{k-1}, Y_{k-1}, X_k)$

Where each X_i and Y_i represent the user input and model output at turn i.

5.3.3 Conversation History Visualization

When viewing results from follow-up jobs, the system offers:

1. **Conversation Timeline**: A visual representation of the conversation history with links to previous jobs
2. **Historical Context Highlighting**: The ability to see how tokens from previous turns influence the current generation:
   - Tokens from previous inputs and outputs are visually differentiated
   - Attribution patterns can be filtered by conversation turn
   - Users can analyze how information from earlier turns propagates through the conversation
3. **Attribution Flow Analysis**: Advanced visualization showing how attribution patterns evolve across turns:
   - Tracking specific tokens or phrases through multiple turns
   - Identifying when earlier context becomes more or less important
   - Visualizing the changing attention patterns as conversation context grows

This conversational analysis capability provides unique insights into how sequence generation models leverage context over multiple turns, revealing attention patterns and information flow that would be difficult to detect in single-turn analysis.

5.4 Linguistic Category Filtering

A powerful analysis capability of the system is the ability to filter and analyze attribution patterns based on linguistic categories of tokens. This feature enables researchers to study how different types of words influence model outputs.

5.4.1 Integration with External NLP Services

Since the attribution system itself does not perform linguistic analysis, this feature requires integration with external NLP services:

1. **Service Integration Architecture**:
   - REST API integration with external NLP processors
   - Batch processing of tokens to minimize API calls
   - Caching of linguistic annotations for previously analyzed texts
   - Fallback mechanisms for when external services are unavailable
2. **Annotation Pipeline**:
   - When a job is completed, token texts are sent to the NLP service
   - Returned annotations are stored alongside the attribution results
   - Multiple annotation layers can be requested (POS tags, NER, syntactic roles)
   - Annotations are linked to the original tokens via unique identifiers

5.4.2 Category-Based Filtering System

Once tokens are enriched with linguistic annotations, the system enables filtering based on these categories:

1. **Predefined Category Filters**:
   - Part-of-speech categories (nouns, verbs, adjectives, stop words, etc.)
   - Named entity types (people, organizations, locations, dates)
   - Syntactic roles (subjects, objects, modifiers)
   - Semantic categories (when available from the NLP service)
2. **Filter Visualization Modes**:
   - **Highlight Mode**: Selected categories are visually emphasized while others are dimmed

- **Isolation Mode**: Only tokens in selected categories are displayed
- **Comparison Mode**: Different categories are displayed with distinct visual encodings

3. **Category Impact Analysis**:
   - Aggregate attribution scores by linguistic category
   - Compare the relative influence of different word types
   - Identify patterns in how the model processes different linguistic elements

5.4.3 User Interface for Category Filtering

The category filtering functionality is exposed through an intuitive interface:

1. **Category Selection Panel**:
   - Dropdown menu or expandable list of available categories
   - Checkboxes for enabling multiple categories simultaneously
   - Visual indicators showing the distribution of categories in the current text
2. **Custom Category Creation**:
   - Allow users to define custom categories based on combinations of annotations
   - Support for regular expression-based token filtering
   - Save and load custom category definitions
3. **Category-Aware Visualizations**:
   - Enhanced attribution heatmap with category-based grouping
   - Statistical summaries of attribution patterns by category
   - Comparative visualizations between category-specific attribution patterns

5.4.4 Implementation Considerations

Several technical considerations guide the implementation of category filtering:

1. **Performance Optimization**:
   - Precompute category-based aggregations when annotations are received
   - Use efficient data structures for category-based token lookup
   - Implement lazy loading of category-specific visualizations
2. **Scalability**:
   - Support for large annotation sets with many categories
   - Hierarchical organization of categories for navigation
   - Progressive loading of category data for very large texts
3. **Error Handling**:
   - Graceful degradation when NLP services return incomplete results
   - Clear user feedback about annotation quality and confidence
   - Manual override options for incorrect annotations

This category filtering system adds a powerful linguistic analysis dimension to the attribution visualization, allowing researchers to uncover patterns in how different types of words influence model outputs.# Attribution Visualization System for Sequence Generation Models: Formal Specification

6. Implementation Considerations

6.1 Technology Stack

1. **Backend**:
   - Job orchestration service (wrapping Inseq functionality)
   - Inseq integration layer for attribution computation
   - WebSocket notification service for real-time job status updates

- Results database for storing attribution outputs

2. **Frontend**:
   - React/TypeScript application
   - Interactive visualization components (extending Inseq's visualization capabilities)
   - WebSocket client for real-time updates

6.2 Inseq Integration

The system leverages several key Inseq functions and data structures:

1. **Model Loading**:

```
attribution_model = inseq.load_model(model, attribution_method)
```

2. **Attribution Computation**:

```
attributions = model.attribute(
    input_text,
    attribute_target=True,  # Include target tokens in attribution
    step_scores=["probability", "entropy"],  # Compute additional scores
    **options
)
```

3. **Visualization**: Extensions of:

```
inseq.show_attributions(attributions)
inseq.show_token_attributions(attributions)
inseq.show_granular_attributions(attributions)
```

4. **Post-processing**: Using Inseq's aggregation framework:

```
aggregated = attributions.aggregate("L2Norm")
```

The backend enhances Inseq's capabilities by providing:

1. **Job Management**: Asynchronous processing of attribution requests with:
   - Job creation, status tracking, and result retrieval
   - WebSocket notifications for real-time status updates
   - Persistent storage of attribution results
2. **Interactive Visualization**:
   - Token-level hover effects that show attribution strengths
   - Context window adjustment for token groups
   - Multi-method visualization with color-coded overlays
   - Model comparison in split view

6.3 Performance Optimizations

A critical aspect of the system is maintaining interactive performance despite the computational intensity of attribution analysis and visualization. The system implements several optimization strategies:

1. **Backend Computation Optimization**:
   - **Parallel Processing**: Attribution jobs are processed using parallel execution to maximize throughput
   - **Job Prioritization**: Short-running jobs are prioritized to provide quick feedback

- **Incremental Processing**: For long-running jobs, intermediate results are made available as they are computed
- **Caching System**: A multi-level caching strategy for:
  - Model loading (to avoid repeated initialization)
  - Attribution results (to reuse computations across similar jobs)
  - Tokenization results (to optimize repeated processing of similar texts)
- **Resource Management**: Automatic scaling of computational resources based on job queue depth

2. **Data Transmission Optimization**:
   - **Compressed Transmission**: Attribution matrices are compressed for network transfer
   - **Selective Response**: Initial API responses include only essential data, with detailed matrices loaded on demand
   - **Progressive Loading**: For large attribution matrices, data is streamed in chunks based on current visualization needs
   - **WebSocket Efficiency**: Binary WebSocket protocols for real-time updates with minimal overhead
3. **Frontend Visualization Optimization**:
   - **Virtualized Rendering**: Only visualize the tokens currently in the viewport for very long sequences
   - **Throttled Event Handlers**: Limit the frequency of expensive computation during rapid interactions
   - **Web Workers**: Offload heavy aggregation computations to background threads to maintain UI responsiveness
   - **GPU Acceleration**: Leverage WebGL or Canvas-based rendering for large attribution matrices
   - **Memoization**: Cache results of aggregation operations for reuse during repeated interactions
   - **Lazy Rendering**: Defer non-critical visualization elements until after initial render completes
   - **Efficient DOM Updates**: Minimize reflows and repaints when updating visualization state
4. **Memory Management**:
   - **Data Pruning**: Automatically release unused attribution data from memory
   - **Downsampling**: For extremely large matrices, apply intelligent downsampling that preserves important patterns
   - **Incremental Garbage Collection**: Schedule memory cleanup during idle periods to avoid UI jank
   - **State Normalization**: Optimize Redux/React state structures to minimize memory footprint

These optimization strategies ensure that the system remains responsive even when dealing with large language models, complex attribution methods, and lengthy input/output sequences.

6.4 Frontend Architecture

The frontend is implemented as a React/TypeScript application with a carefully designed component architecture to ensure modularity, reusability, and performance. Instead of providing specific implementation code, we describe the architectural patterns and component structure that guide the implementation.

6.4.1 Component Architecture

The frontend follows a layered component architecture:

1. **Core Components**:
   - **TokenVisualization**: Base component for rendering tokens with attribution-based styling
   - **AttributionMatrix**: Manages the data structure and computations for attribution scores
   - **AttributionViewer**: Coordinates the overall visualization and interaction logic
   - **ControlPanel**: Manages user controls for adjusting visualization parameters
2. **Specialized View Components**:
   - **SingleModelView**: Displays attribution for one model
   - **SplitViewComparison**: Implements the side-by-side model comparison
   - **ConversationTraceView**: Visualizes attribution patterns across conversation turns
3. **Layout Components**:

- **DashboardLayout**: Manages the overall structure of the Attribution Dashboard
- **JobCreationForm**: Handles the job creation interface
- **JobStatusTable**: Displays running and completed jobs

6.4.2 State Management Patterns

The system uses a structured state management approach:

1. **Global Application State**:
   - Job queue and status
   - User preferences and settings
   - Authentication state
2. **Component-Local State**:
   - Hover interactions
   - Visual control settings
   - Ephemeral UI state
3. **Derived State**:
   - Computed attribution scores
   - Aggregation results
   - Visual encodings

This separation allows for efficient state updates and minimizes unnecessary re-renders.

6.4.3 Data Flow Architecture

Data flows through the system in a well-defined pattern:

1. Raw attribution data is received from the API
2. Data is normalized and processed into efficient internal structures
3. Aggregation functions are applied based on user interactions
4. Visual parameters are computed and applied to rendering components
5. User interactions trigger specific state updates, which flow back through the pipeline

This unidirectional data flow ensures predictable behavior and simplifies debugging.

6.4.4 Interactive Features Implementation

The system implements key interactive features through specialized hooks and utilities:

1. **Token Interaction**:
   - Custom hooks manage token hover state and propagate highlight updates
   - Event handlers implement throttling for performance during rapid interactions
   - Memoized computations ensure efficient re-rendering
2. **Context Window Adjustment**:
   - Slider components with immediate visual feedback
   - Optimized aggregation functions that scale with window size
   - Preview capability to show effect before finalizing changes
3. **Method Toggling**:
   - Toggle components with method-specific styling
   - Efficient method combination algorithms
   - Color encoding system for multi-method visualization

6.4.5 Component Communication Patterns

Components communicate through a combination of:

1. **Props**: For direct parent-child communication
2. **Context**: For theme, configuration, and globally relevant state
3. **Custom Events**: For cross-component communication
4. **State Management**: For complex state that affects multiple components

This approach balances performance and maintainability, avoiding both prop-drilling and excessive global state.

6.4.6 Responsive Design Strategy

The system adapts to different screen sizes through:

1. **Fluid Layouts**: Components resize proportionally based on available space
2. **Priority-Based Content**: Critical visualization elements remain visible while secondary controls adapt
3. **Progressive Enhancement**: Additional features appear on larger screens
4. **Interaction Adaptation**: Touch-friendly controls on mobile devices, precision controls on desktop

6.5 Extensibility and Integration

The system is designed with extensibility as a core principle, ensuring it can evolve with advances in attribution methods, models, and visualization techniques.

6.5.1 Attribution Method Integration

New attribution methods can be integrated through a structured extension pattern:

1. **Backend Integration**:
   - Leverage Inseq's extensibility for integrating new attribution methods
   - Implement method-specific computation optimizations
   - Define default aggregation strategies appropriate for the method
   - Register method metadata (description, parameters, best practices)
2. **Frontend Support**:
   - Add method-specific visualization parameters (color scheme, visual encoding)
   - Implement specialized interaction patterns if required
   - Update method selection UI to include the new method
   - Provide method-specific documentation and tooltips
3. **Method Combination Rules**:
   - Define compatibility with other attribution methods
   - Specify aggregation behavior when combined with other methods
   - Implement method-specific normalization for fair comparison

6.5.2 Model Integration Framework

The system supports seamless integration of new sequence generation models:

1. **Model Registration Process**:
   - Register models in the model registry with metadata
   - Specify model architecture and compatible attribution methods
   - Define model-specific tokenization and processing requirements
   - Configure default attribution parameters for optimal results
2. **Model-Specific Adaptations**:
   - Support for model-specific token representations
   - Handling of different tokenization strategies
   - Adaptation to model-specific attribution patterns
   - Custom visualization adjustments for model-specific features

3. **Performance Optimization Hooks**:
   - Model-specific caching strategies
   - Optimized computation paths for common model architectures
   - Specialized batching for efficient model utilization

6.5.3 Visualization Extensibility

The visualization system is designed to be extended with new techniques:

1. **Custom Visual Encodings**:
   - Define new color schemes and visual mappings
   - Implement alternative representation methods (e.g., force-directed graphs, treemaps)
   - Create specialized visualizations for particular attribution patterns
   - Support for domain-specific visual languages
2. **Interactive Element Extensions**:
   - Add new interactive controls
   - Implement specialized hover behaviors
   - Create custom filtering and highlighting mechanisms
   - Develop new comparison visualization techniques
3. **View Composition Framework**:
   - Combine multiple visualization types in dashboard layouts
   - Create custom views for specific analysis workflows
   - Implement specialized comparison views for different attribution aspects
   - Support for user-defined view configurations

6.5.4 Data Processing Extensions

The data processing pipeline supports extension at multiple points:

1. **New Aggregation Functions**:
   - Extend the aggregation function registry
   - Implement custom mathematical operations for specific analysis needs
   - Create domain-specific aggregation methods
   - Define multi-dimensional aggregation techniques
2. **Custom Analysis Algorithms**:
   - Implement automatic pattern detection for attribution matrices
   - Create specialized metrics for attribution quality assessment
   - Develop statistical analysis tools for attribution comparison
   - Build anomaly detection for identifying unusual attribution patterns
3. **Data Export and Integration**:
   - Create connectors for external analysis tools
   - Implement formats for interoperability with other visualization systems
   - Build export capabilities for publication-ready visualizations
   - Develop integration with model editing and fine-tuning tools

6.5.5 Extension API

The system provides a documented API for extensions, including:

1. **Plugin Architecture**:
   - Well-defined extension points
   - Registration system for new components
   - Lifecycle hooks for extension initialization and cleanup

- Configuration system for extension parameters

2. **Extension Management**:
   - User interface for enabling/disabling extensions
   - Dependency management for extension compatibility
   - Version control for extensions
   - Performance monitoring for extensions

This comprehensive extensibility framework ensures the system can adapt to evolving research needs and incorporate new techniques and models as they become available.

7. Future Directions

While the current specification provides a comprehensive foundation for attribution visualization, several promising directions for future enhancement can be identified:

7.1 Advanced Analysis Capabilities

1. **Attribution Pattern Mining**: Automated identification of recurring attribution patterns across different inputs and models
2. **Comparative Analysis Framework**: Tools for systematically comparing attribution patterns between different models or versions
3. **Statistical Analysis Tools**: Quantitative assessment of attribution quality, stability, and correlation with model performance
4. **Attribution Anomaly Detection**: Automated identification of unusual attribution patterns that may indicate model weaknesses

7.2 Enhanced Visualization Techniques

1. **3D Visualization**: Three-dimensional representations of attribution relationships for complex multi-dimensional patterns
2. **Temporal Animation**: Animated visualizations showing how attributions evolve during the generation process
3. **Network Graph Representations**: Alternative visualization as token relationship networks rather than matrices
4. **Augmented Reality Integration**: AR-based visualization for immersive exploration of complex attribution spaces

7.3 Integration with Model Development

1. **Attribution-Guided Fine-tuning**: Tools to use attribution insights for targeted model improvement
2. **Prompt Engineering Support**: Features to help users craft prompts that lead to desired attribution patterns
3. **Model Debugging Interface**: Integration with model debugging workflows to identify and fix attribution issues
4. **Continuous Monitoring**: Systems for tracking attribution patterns across model versions and updates

7.4 Collaborative Features

1. **Multi-user Annotation**: Support for collaborative analysis of attribution patterns
2. **Attribution Pattern Sharing**: Capability to share, comment on, and collaborate around interesting attribution findings
3. **Attribution Libraries**: Repositories of notable attribution patterns for educational and research purposes
4. **Integration with Research Workflows**: Tools for incorporating attribution analysis into research publications and presentations

8. Conclusion

This formal specification defines a comprehensive attribution visualization system for sequence generation models. By combining a mathematically rigorous foundation with an intuitive interactive interface, the system enables researchers and practitioners to gain deep insights into the behavior of language models through attribution analysis.

The job-based architecture provides a flexible and scalable framework for processing attribution requests, while the interactive frontend visualization offers an intuitive way to explore the complex relationships between input and output tokens. By supporting multiple attribution methods, model comparison, and conversational follow-up analysis, the system offers a powerful platform for understanding and improving sequence generation models.

The emphasis on mathematical formalization ensures precision and reproducibility, while the detailed visualization and interaction specifications guide the implementation of an intuitive and responsive user experience. The performance optimization strategies and extensibility framework ensure the system can handle large-scale models and adapt to evolving research needs.

By bridging the gap between theoretical attribution methods and practical exploration tools, this system aims to advance our understanding of large language models and contribute to their responsible development and application.