TNSA.ai

Interpretable Attention Visualization Module: Transforming Raw Attention into Human-Readable Explanations

Executive Summary

This report details the development of an Interpretable Attention Visualization Module, a novel plug-in designed to convert the complex, high-dimensional raw attention maps from transformer-based Large Language Models (LLMs), specifically GPT-2 small, into intuitive, human-readable explanations. The module addresses a critical need for enhanced transparency and trustworthiness in LLM applications by demystifying the internal workings of the attention mechanism. The proposed approach involves a multi-stage process: precise extraction of raw attention data, advanced clustering of tokens based on their co-attention patterns, and the generation of natural language rationales for each identified cluster. The module's effectiveness and utility will be rigorously validated through comprehensive user studies, ensuring that the explanations are not only faithful to the model's behavior but also clear and useful for human understanding. This initiative aims to provide developers and researchers with a powerful tool for debugging, validating, and fostering greater confidence in LLM predictions.

1 Introduction: The Imperative of Interpretable Attention in LLMs

1.1 The Role of Attention in Transformer Models

Transformer models, which underpin modern LLMs like GPT-2, have fundamentally reshaped the landscape of Natural Language Processing (NLP). Their success is largely attributed to the self-attention mechanism, a core component that enables the model to dynamically weigh the importance of different words or tokens within an input sequence relative to each other. This mechanism allows the model to capture intricate contextual relationships and long-range dependencies in text, which is paramount for tasks ranging from language understanding to generation. The attention weights themselves are quantitative indicators of the model's internal "focus," revealing which parts of the input are deemed most salient or influential for a particular output or internal representation.

The attention mechanism is more than a mere computational component; it serves as a direct window into the model's internal processing, akin to its "thought process." By transforming this quantitative focus into qualitative, human-understandable explanations, it becomes possible to transcend the conventional black-box nature of LLMs. This transformation allows stakeholders to comprehend not just what a model predicts, but why it arrives at a particular conclusion. Such transparency is indispensable for applications in high-stakes domains, where understanding the basis of a model's decision is crucial for accountability and debugging. This capability directly addresses the growing demand for explainable AI (XAI) and contributes significantly to building trust and reliability in complex LLM deployments.

1.2 Challenges in Interpreting Raw Attention Maps

Despite their foundational role, raw attention maps present significant challenges for direct human interpretation. These maps are typically high-dimensional tensors, often structured as (batch_size, num_heads, query_sequence_length, key_value_sequence_length). While visualizations such as heatmaps or connecting lines can offer a preliminary glimpse into attention patterns, they often fall short of providing a direct, natural language explanation for why specific tokens attend to others.

The complexity is further compounded by the multi-headed nature of attention in transformers. Models like BERT-Base, for instance, employ 12 attention heads, each potentially specializing in capturing distinct linguistic properties, such as named entities, punctuation, or subject-verb relationships. Aggregating or distilling these diverse, granular patterns from multiple heads and layers into a single, coherent, and intuitive explanation is a non-trivial task. The fundamental challenge lies not merely in visualizing the data, but in abstracting these low-level, quantitative attention patterns into higher-level, semantically meaningful concepts that resonate with human understanding. This semantic compression of attention patterns is a form of model summarization or rationale generation, necessitating advanced NLP techniques beyond simple graphical representation.

1.3 Overview of the Proposed Interpretable Attention Visualization Module

The proposed Interpretable Attention Visualization Module is designed to bridge this critical gap between raw attention data and human comprehension. The plug-in will systematically extract raw attention maps from GPT-2 small, cluster tokens based on their co-attention patterns, and subsequently generate natural language rationales for each identified cluster. This innovative approach aims to provide a robust and user-friendly tool for developers and researchers, enabling them to debug, build trust in, and ultimately refine the behavior of LLMs by offering clear, interpretable insights into their internal decision-making processes.

2 Extracting and Characterizing Attention from GPT-2 Small

2.1 Technical Deep Dive: Accessing Raw Attention Maps via Hugging Face Transformers

The Hugging Face `transformers` library offers a highly efficient and standardized interface for accessing the internal activations of pre-trained models, including the raw attention weights from GPT-2 small. This is achieved by setting the `output_attentions=True` parameter during the model's forward pass or generation call, which instructs the model to return the attention tensors along with its standard outputs.

For GPT-2, the relevant components are the `GPT2Tokenizer` for tokenizing input text and the `GPT2Model` for the model itself. The attention output is typically returned as a tuple of tensors, where each tensor corresponds to the attention weights from a specific layer of the model. Each layer's attention tensor usually has a shape of (batch_size, num_heads, sequence_length, sequence_length). In this structure, sequence_length refers to both the query and key sequence lengths, representing the attention scores from each query token to every key token within the sequence.

A conceptual code example demonstrating this process for GPT-2 would involve:

```
from transformers import GPT2Tokenizer, GPT2Model
import torch

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2Model.from_pretrained('gpt2', output_attentions=True) # Essential

```

```
text = "The quick brown fox jumps over the lazy dog."
encoded_input = tokenizer(text, return_tensors='pt')

# Perform a forward pass to obtain outputs, including attentions
with torch.no_grad():
    output = model(**encoded_input)

# Access attentions: output.attentions is a tuple of tensors, one per layer
# Each tensor typically has shape (batch_size, num_heads, seq_len, seq_len)
attentions_per_layer = output.attentions

# Example: Access attention from the last layer, first head
# This will be (sequence_length, sequence_length) for a single batch and head
last_layer_attention = attentions_per_layer[-1]
first_head_attention = last_layer_attention[0, 0, :, :] # Batch 0, head 0

# 'first_head_attention' is now a 2D matrix for analysis.
```

Listing 1: Accessing GPT-2 Attention via Hugging Face

The `output_attentions=True` flag is a standard and critical feature implemented across a wide range of Transformer models within the Hugging Face ecosystem. This widespread availability ensures that the plug-in's primary data source is consistently accessible and leverages a universally adopted infrastructure. This design choice significantly streamlines the initial data acquisition phase, eliminating the need for custom hooks or modifications to the underlying model architectures, thereby enhancing the plug-in's compatibility and ease of integration with various Transformer models in the future.

2.2 Understanding Attention Patterns in GPT-2 (e.g., "Massive Values")

Empirical research indicates that in many contemporary transformer-based LLMs, "concentrated massive values consistently emerge in specific regions of attention queries (Q) and keys (K)". These highly concentrated values are considered crucial for interpreting contextual knowledge acquired from the current input window. This phenomenon suggests that the model does not distribute attention uniformly across all token pairs; instead, it assigns significantly higher weights to certain token-to-token relationships, indicating their profound influence on the model's processing.

Furthermore, different attention heads within a transformer often specialize in capturing distinct linguistic properties. For instance, some heads might focus on named entities, while others might identify punctuation or subject-verb pairs. This multi-headed architecture implies that a comprehensive understanding of the model's focus requires analyzing the raw attention map not just as a single entity, but by examining the individual contributions and patterns of its constituent heads.

The observation of "massive values" in attention maps suggests that attention is inherently sparse and highly focused rather than uniformly distributed. This inherent sparsity can be strategically exploited during the clustering process to prioritize the most salient connections. By focusing on these high-value attention links, the complexity of the clustering problem can potentially be reduced, leading to the identification of more semantically meaningful and interpretable clusters. This approach naturally aligns with efforts to enhance transformer efficiency through sparse attention mechanisms. For instance, a methodology for Vision Transformers (ViTs) involves a crucial pruning or thresholding step before clustering, where edges with weights below a certain threshold are removed to reveal distinct and coherent clusters. This practice, applied to text-based attention, could make the clustering process more efficient and yield more semantically relevant groupings of tokens.

3 Advanced Clustering of Attention Patterns

3.1 Defining and Identifying Co-attention for Textual Tokens

The concept of "co-attention" traditionally refers to the simultaneous clustering of rows and columns in a matrix, often termed biclustering, or the joint reasoning between different modalities, such as an image and a question in Visual Question Answering (VQA) systems. In the context of text-based attention, "co-attention" can be interpreted as identifying groups of tokens that frequently attend to each other or exhibit similar patterns of attention across the input sequence. This implies a measure of semantic or syntactic relatedness as perceived and processed by the LLM.

Transformers inherently learn to group words by topic. This learning manifests as higher average inner products between word embeddings and increased pairwise attention weights among words that belong to the same semantic topic. This indicates that the attention weights themselves provide a robust signal for semantic clustering. The self-attention mechanism, therefore, naturally produces a clustering effect. This inherent property validates the foundational premise of the proposed plug-in: rather than imposing an arbitrary structure, the module aims to reveal and formalize a structure that the model has already learned. This provides a strong theoretical underpinning for the interpretability approach, as the resulting clusters are intrinsically faithful to the model's actual processing. The model's learning objective, such as masked language modeling, directly drives the emergence of these topic-based attention patterns, making them a reliable source for deriving interpretable groupings.

3.2 Algorithms for High-Dimensional Attention Data Clustering

Attention maps are inherently high-dimensional, represented by matrices of sequence_length $\times$ sequence_length for each head and layer. Directly clustering data in such high-dimensional spaces is susceptible to the "curse of dimensionality," where data points become sparse and distances between them less informative. To mitigate this, dimensionality reduction techniques like t-distributed Stochastic Neighbor Embedding (t-SNE) or Uniform Manifold Approximation and Projection (UMAP) are often recommended for initial visualization and can assist in the clustering process.

Graph-based Clustering: Attention maps can be naturally conceptualized as adjacency matrices of graphs, where individual tokens serve as nodes and the attention weights between them represent the strength of the edges. Community detection algorithms are particularly well-suited for partitioning these graphs into meaningful clusters (or communities). Algorithms such as Louvain and Infomap are designed to optimize for modularity, aiming to identify partitions where connections are dense within clusters and sparse between them.

A methodology applied to Vision Transformers (ViTs) for clustering attention maps provides a highly relevant blueprint. This approach involves:

1. Obtaining attention maps from the model.
2. Treating the attention map as an adjacency matrix.
3. Constructing a directed graph from this matrix.
4. Crucially, pruning edges with weights below a specified threshold. This step is essential because raw attention maps often have non-zero entries for all token pairs, which would otherwise result in a fully connected graph with no discernible clusters.
5. Applying community detection algorithms like Louvain, Infomap, or Graph clustering via phase transitions in semidefinite relaxations (SDP) to the pruned graph.
6. Calculating the modularity score (Q) for the resulting clusters, which quantifies the strength of the community structure.
7. Analyzing modularity scores across different threshold values to identify an optimal threshold that maximizes modularity in a stable phase.

While this ViT clustering methodology focuses on image patches, its underlying principles—graph representation, pruning, and community detection—are directly transferable to text-based attention. In this context, text tokens (words or subwords) are analogous to image patches, and attention weights between them are analogous to connections. The graph-based approach (nodes as tokens, edges as attention weights) is a natural fit for analyzing textual attention. The primary adaptation required lies in interpreting the resulting clusters within a linguistic context (e.g., identifying semantic topics, syntactic roles, or coreferent entities) and determining appropriate pruning thresholds for text-specific attention patterns.

Beyond graph-based methods, other clustering algorithms are also relevant:

- **Agglomerative Token Clustering (ATC):** This method builds clusters by iteratively combining the most similar tokens in a bottom-up hierarchical manner, without introducing additional learnable parameters. Although primarily used for efficiency in ViTs, the principle of merging similar tokens based on their attention patterns is directly applicable to interpretability.
- **Clustering Self-Attention using Surrogate Tokens (CAST):** This approach optimizes attention computation by employing learnable surrogate tokens to construct a cluster affinity matrix. While its main objective is efficiency, the concept of deriving a "cluster affinity matrix" from attention data can be adapted for identifying meaningful token groupings for interpretability.

Recommended Algorithms:

- **Louvain/Infomap:** Highly suitable for detecting community structures in graphs, making them directly applicable to attention maps as graphs.
- **Hierarchical Clustering:** Effective for high-dimensional data, it does not require prespecifying the number of clusters and can be combined with dimensionality reduction techniques.
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** Works well for identifying dense regions and handling noise, which can be useful if attention patterns exhibit varying densities across the input.

3.3 Practical Considerations for Attention Map Preprocessing and Clustering

Several practical considerations are crucial for effectively preprocessing and clustering attention maps:

- **Aggregation:** Attention maps are multi-headed and multi-layered. Strategies for combining attention from different heads and layers—such as averaging attention weights, summing them, or selecting specific "interpretable" heads identified through prior research—will be critical to reduce dimensionality and focus on salient patterns.
- **Normalization/Scaling:** Attention weights can vary significantly in magnitude. Normalizing or scaling these weights before applying clustering algorithms can improve their performance and the quality of the resulting clusters.
- **Thresholding/Pruning:** As demonstrated in ViT attention clustering, pruning low-weight connections is essential to reveal meaningful clusters and prevent the formation of a single, fully connected graph. Determining an optimal threshold will require iterative experimentation and potentially leveraging modularity maximization techniques.
- **Feature Representation for Clustering:** A key design choice is whether to cluster the raw attention weight vectors for each token, or to derive higher-level features (e.g., token embeddings, or aggregated attention patterns for each token). Research on topic structure in transformers indicates that both token embeddings and attention weights encode topic information, suggesting multiple viable feature representations.
- **Choosing the Number of Clusters:** Many clustering algorithms, such as K-means, require the number of clusters to be specified in advance, which is often unknown for attention patterns. Algorithms like DBSCAN or hierarchical clustering that do not require

this upfront, or methods for determining optimal clusters (e.g., silhouette score, elbow method), will be explored.

Table 1: Overview of Clustering Algorithms for Attention Data

Algorithm Name	**Type**	**Key Strengths for Attention Data**	**Key Weaknesses/Challenges**	**Applicability to High-Dim. Data**
Louvain / Infomap	Graph-based	Detects community structures in graphs, directly applicable to attention maps as graphs, optimizes modularity for clear clusters.	Requires graph construction; pruning threshold selection is critical.	Effective when attention maps are sparse after pruning.
Hierarchical Clustering	Hierarchical	Does not require pre-specifying number of clusters, can reveal nested structures.	Computationally intensive for very large datasets, sensitivity to noise.	Effective, especially with dimensionality reduction.
DBSCAN	Density-based	Identifies dense regions as clusters, handles noise, does not require pre-defined number of clusters.	Sensitive to parameter choice (ϵ, min_samples), struggles with varying densities.	Works well by identifying dense regions.
Agglomerative Token Clustering (ATC)	Merging-based	Iteratively combines similar tokens based on attention patterns, no extra learnable parameters.	Primarily for efficiency; direct interpretability application needs adaptation.	Applicable for token reduction, implying suitability for high-dim. token features.
Clustering Self-Attention using Surrogate Tokens (CAST)	Surrogate-based	Optimizes attention computation by using learnable surrogate tokens for cluster affinity.	Primary goal is efficiency; adaptation for interpretability is required.	Designed for efficient transformers, implying high-dim. applicability.

This table systematically compares leading clustering algorithms, providing a clear, structured guide for selecting the most appropriate algorithms for the plug-in. This selection is based on their theoretical fit (e.g., graph-based methods for attention maps) and practical considerations (e.g., handling high dimensionality, not requiring a pre-defined cluster count), directly informing the implementation strategy.

4 Generating Natural Language Rationales from Attention Clusters

4.1 State-of-the-Art in Data-to-Text Generation for Structured Data

Natural Language Generation (NLG) from structured data is a mature field focused on converting organized information, such as the identified attention clusters, into coherent and human-readable text. This domain has seen substantial advancements with the proliferation of sequence-to-sequence (Seq2Seq) models and Transformer-based architectures.

Approaches like Graph-to-Text generation are particularly relevant, as attention clusters, with their interconnected tokens and associated attention weights, can be conceptualized as subgraphs of token relationships. These methods integrate graph representation learning, attention mechanisms, and text generation models to produce fluent narratives from structured inputs. For instance, the Bidirectional Dual Cross-Attention and Concatenation (BDCC) framework

specifically addresses the conversion of graph structures into natural language by leveraging both linearized and direct graph representations. This framework's ability to fuse information from different modalities (linearized text and graph structure) could be adapted to process attention graphs, translating their complex patterns into understandable prose.

The significant progress in NLG from structured data means that the development of the proposed module does not require the invention of an entirely new NLG paradigm. Instead, existing, well-established techniques can be adapted. The attention clusters, along with their constituent tokens and attention weights, serve as the structured input for an NLG model. This approach substantially reduces developmental risk by leveraging proven methodologies, allowing the focus to remain on the unique challenge of interpreting attention rather than re-inventing core NLG components.

4.2 Leveraging Large Language Models for Rationale Synthesis

Modern LLMs, especially those that have undergone instruction tuning or are capable of Chain-of-Thought (CoT) prompting, are highly effective at generating coherent and contextually relevant text. CoT prompting, which involves providing a few examples of intermediate reasoning steps, has been shown to significantly enhance LLMs' ability to perform complex reasoning and generate natural language rationales. This capability suggests that LLMs can be guided to articulate the underlying logic of attention clusters.

Furthermore, fine-tuning pre-trained LLMs (e.g., T5, BART) on structured data-to-text tasks has demonstrated success in producing high-quality textual explanations. This supports a supervised learning approach where a dataset of (attention cluster representation, human-written rationale) pairs could be curated for fine-tuning. The "neuralization-propagation" (NEON) framework offers a principled method for explaining cluster assignments by propagating "relevance" back to the input features. While NEON primarily explains why a data point belongs to a cluster, its underlying relevance mapping can inform the NLG process, providing the essential "semantic content" for the rationale. This means that NEON can identify the most relevant input tokens contributing to a cluster, and this information can then be formatted as structured input for an LLM.

Combining CoT prompting with fine-tuning on structured representations of attention clusters, informed by NEON's relevance mapping, presents a powerful strategy for generating high-quality, faithful, and human-readable rationales. This approach involves representing an attention cluster as structured data (e.g., a list of tokens, their attention scores, and inter-token relationships). By employing CoT prompting, the LLM can be guided to generate step-by-step explanations for why these tokens form a cluster and what semantic property they collectively represent. Fine-tuning the LLM on carefully constructed examples of such structured inputs and desired rationales would further enhance its ability to produce accurate and insightful explanations. This integrated approach aims to generate rationales that are not merely descriptive but truly explanatory, reflecting the causal links within the model's attention mechanism.

4.3 Strategies for Ensuring Coherence, Faithfulness, and Conciseness in Rationales

The quality of generated rationales is paramount for the module's utility. Therefore, specific strategies will be employed to ensure their coherence, faithfulness, and conciseness:

- **Faithfulness:** The rationale must accurately reflect the underlying attention patterns and the model's actual decision-making process. This means the generated explanation must directly correspond to the high attention weights and the specific tokens identified within each cluster. Discrepancies between the explanation and the model's true behavior would undermine trust.

- **Plausibility/Coherence:** The explanation should be understandable and intuitively make sense to a human user. This necessitates fluent, grammatically correct, and semantically consistent natural language. The language should be natural and avoid jargon where possible, presenting information in a way that aligns with human cognitive processes.
- **Conciseness:** Explanations should be succinct and to the point, avoiding unnecessary verbosity. Overly long or convoluted explanations can be as unhelpful as no explanation at all. The goal is to provide the most critical information efficiently.
- **Mitigating Hallucination:** A significant challenge in NLG is the phenomenon of "hallucination," where models generate factually incorrect or ungrounded text. For rationales, this translates to generating explanations that do not accurately reflect the true attention patterns. Strategies to mitigate this include:
  - *Constrained Decoding:* Guiding the text generation process with the structured attention data to ensure semantic correctness and factual grounding. This involves incorporating rules or constraints that force the NLG model to adhere to the factual information derived from the attention clusters.
  - *Feedback-driven Refinement:* Implementing iterative processes where the LLM generates a draft rationale, reflects on it, and then refines it based on self-generated critiques or external validation. This allows for a continuous improvement loop, catching and correcting inaccuracies.
  - *Data Quality:* Curating a high-quality training dataset for fine-tuning the NLG model, where human experts meticulously annotate attention clusters with accurate and faithful rationales. High-quality training data is fundamental to teaching the model to generate reliable explanations.

5 Designing and Evaluating User Studies for Interpretability

5.1 Methodologies for Assessing Clarity and Usefulness of Explanations

User studies are indispensable for validating the human-readability and practical utility of the generated explanations. These studies must involve diverse user groups to ensure broad applicability and generalizability of findings. A combination of qualitative and quantitative methodologies will be employed:

Qualitative Methods:

- *Think-aloud protocols:* Users will be encouraged to vocalize their thoughts, interpretations, and challenges as they interact with the module and attempt to understand the explanations. This provides rich, real-time data on cognitive processes.
- *Semi-structured interviews:* Post-interaction interviews will gather in-depth feedback on the clarity, intuitiveness, and perceived usefulness of the rationales, allowing for exploration of nuanced user experiences.
- *Surveys with open-ended questions:* These will collect broader feedback and help identify common themes, unexpected uses, or persistent issues across a larger user base.

Quantitative Methods:

- *Likert scale ratings:* Users will provide subjective ratings on scales for attributes such as clarity, trustworthiness, satisfaction, and perceived usefulness of the explanations.
- *Task-based evaluations:* Users will be asked to perform specific tasks (e.g., predicting model behavior, identifying potential errors in model output, or debugging a given prediction) using the provided explanations. Performance metrics such as accuracy in prediction, time taken to complete tasks, or success rate in debugging will be recorded.
- *Comparative studies:* The explanations generated by the proposed module will be compared against those from existing baseline visualization tools (e.g., BertViz, LIT, Ecco) or other interpretability methods. This allows for a quantitative assessment of the module's added

value.

The existence of frameworks like EvalxNLP provides a structured and validated approach to designing and executing user studies for NLP explainability. This means that the development team does not need to define user study methodologies and metrics from scratch, significantly streamlining the validation phase. EvalxNLP integrates various explainability techniques and evaluation metrics, and even offers LLM-based textual explanations for evaluation outcomes, which can be leveraged to design a robust user study for the module.

5.2 Key Metrics for XAI Evaluation (Faithfulness, Plausibility, Complexity)

EvalxNLP proposes a comprehensive set of metrics categorized into three key properties for evaluating XAI methods. These metrics will be adopted to rigorously assess the quality of the generated rationales:

- **Faithfulness:** This category measures how accurately the generated explanations reflect the true behavior of the underlying model. Metrics include soft sufficiency, soft comprehensiveness, Feature Attribution Dropping (FAD) Curve and Normalized Area Under Curve (N-AUC), and Area Under the Threshold-Performance Curve (AUC-TP). These metrics ensure that the explanations are not merely plausible but genuinely representative of the model's internal attention mechanisms.
- **Plausibility:** This measures how well the generated explanations align with human intuition and understanding. Metrics in this category include IOU-F1 Score, Token-Level F1 Score, and Area Under Precision-Recall Curve (AUPRC). This property is particularly relevant for "human-readable explanations," as it directly assesses whether the explanations are comprehensible and make intuitive sense to human users.
- **Complexity:** This evaluates how concise and interpretable the explanations are, with a preference for sparse explanations that convey maximum information with minimum verbosity. Metrics such as Shannon entropy and the Gini index are used to quantify complexity. Conciseness is a direct objective for the generated rationales.

The emphasis on "plausibility" metrics highlights the inherently subjective and human-centric nature of interpretability. This means that while quantitative faithfulness to the model's internal state is important, the ultimate success of the plug-in hinges on how well humans perceive and understand the explanations. This reinforces the necessity for an iterative design process and continuous refinement based on user feedback, where initial rationales are tested, and the NLG component is adjusted based on human judgments of plausibility and clarity.

5.3 Frameworks and Best Practices for User-Centric Evaluation

To ensure systematic and rigorous assessment, the user studies will adhere to established XAI evaluation frameworks and best practices:

- Utilize established frameworks: Leveraging frameworks like EvalxNLP will provide a structured approach to evaluation, ensuring consistency and comparability of results.
- Design clear tasks: Users will be given well-defined tasks, such as identifying key tokens, predicting model output, or debugging model errors based on the provided explanations.
- Employ mixed methods: A combination of quantitative and qualitative data collection will capture both objective performance and subjective user experience.
- Address biases: Efforts will be made to address potential biases in user studies by ensuring diverse participant demographics and controlled experimental conditions.
- Provide interactive interfaces: While the primary output is textual, providing interactive elements for users to explore raw attention maps (potentially integrating existing tools like BertViz) will allow for deeper investigation and comparison with the generated explanations.

This table provides a structured overview of established metrics from XAI literature, particularly those relevant to NLP. It offers a concrete framework for objectively measuring the

Table 2: Key Metrics for Evaluating AI Interpretability

Metric Category	**Specific Metrics**	**Definition/Purpose**	**Relevance to Attention Explanation**
Faithfulness	Soft Sufficiency	Measures how much of the model's prediction can be retained using only the important features.	Ensures the rationale captures the truly influential attention patterns.
	Soft Comprehensiveness	Measures how much of the model's prediction is lost when important features are removed.	Verifies that the explanation covers all critical attention aspects.
	FAD Curve & N-AUC	Evaluates how model performance drops as important features are systematically removed.	Quantifies how well identified attention clusters align with model behavior.
	AUC-TP	Assesses the trade-off between number of features used and model performance.	Helps optimize conciseness while maintaining explanatory power.
Plausibility	IOU-F1 Score	Measures overlap between human-annotated rationales and generated explanations.	Directly assesses how well generated rationales match human intuition for attention.
	Token-Level F1 Score	Evaluates precision and recall of individual tokens identified as important in the explanation.	Ensures fine-grained accuracy in identifying key tokens within attention clusters.
	AUPRC	Assesses ability of explanation to correctly identify relevant tokens.	Provides comprehensive measure of how well explanation highlights true attention foci.
Complexity	Shannon Entropy	Quantifies information content or "spread" of the explanation.	Favors concise explanations that are easy to process without excessive detail.
	Gini Index	Measures inequality or concentration of importance across features.	Promotes sparse explanations where only truly significant attention patterns are highlighted.

success of the plug-in's explanations, ensuring that the user study results are quantifiable and comparable, and directly addressing the "user-study to validate clarity and usefulness" goal.

6 Architectural Design and Implementation Roadmap

6.1 Proposed Plug-in Architecture and Component Integration

The Interpretable Attention Visualization Module will be engineered with a modular architecture to ensure clear separation of concerns, facilitating development, maintenance, and future extensibility. The core components are designed to work in a sequential pipeline:

1. **Input Layer:** This module will serve as the entry point, receiving raw text input from the user.
2. **GPT-2 Integration Module:**
   - It will utilize the Hugging Face `GPT2Tokenizer` to convert the raw input text into a sequence of tokens.
   - The `GPT2Model` will be loaded, crucially initialized with `output_attentions=True`. This setting ensures that the raw attention maps are generated and accessible during the model's forward pass.
   - A forward pass will then be executed to extract these raw attention maps, which are typically tensors representing attention scores across all layers and heads of the GPT-2 model.
3. **Attention Processing & Clustering Module:**
   - *Preprocessing:* This sub-module will aggregate attention across different heads and layers. Initial strategies may include averaging attention weights or selecting specific heads known to capture interpretable linguistic properties. A critical step will be applying thresholding or pruning to the attention weights, similar to methodologies in Vision Transformers. This eliminates low-weight connections, which are often noise, to reveal more meaningful underlying structures.
   - *Graph Construction:* The preprocessed attention maps will be transformed into graph representations, where individual tokens serve as nodes and the attention weights between them form the edges.
   - *Clustering:* Selected community detection algorithms (e.g., Louvain, Hierarchical Clustering, DBSCAN) will be applied to these graph representations to identify distinct clusters of co-attending tokens.
   - *Cluster Characterization:* For each identified cluster, this component will extract and summarize its key constituent tokens and their average attention patterns, providing a condensed representation of the cluster's focus.
4. **Rationale Generation Module:**
   - *Structured Data Representation:* Each attention cluster, characterized by its list of tokens, their associated attention scores, and inter-token relationships, will be converted into a structured data format suitable for Natural Language Generation (NLG).
   - *LLM Integration:* A fine-tuned LLM (potentially a smaller open-source model or an instruction-tuned variant) will be employed to generate natural language rationales for each cluster. This process may incorporate Chain-of-Thought (CoT) prompting to guide the LLM toward generating more detailed and logical explanations. Insights from the NEON framework, which links cluster assignments to input features, can be used to ensure that the generated rationales are faithful to the underlying attention mechanisms.
   - *Refinement:* Mechanisms will be incorporated to ensure the coherence, faithfulness, and conciseness of the generated rationales, potentially utilizing self-correction loops where the LLM critiques and improves its own output.

5. **Output & Visualization Module:**
   - The module will present the original input text, with the identified token clusters highlighted for visual clarity.
   - The generated natural language rationales for each cluster will be displayed alongside the text.
   - Interactive elements will be provided, allowing users to delve deeper into the raw attention maps (potentially by integrating existing visualization tools like BertViz) and compare them with the generated textual explanations.

6.2 Leveraging Existing Open-Source Libraries and Tools

The development will strategically leverage existing open-source libraries and tools to accelerate development and ensure robustness:

- **Hugging Face Transformers:** This library is indispensable for loading the GPT-2 model and efficiently extracting its raw attention maps.
- **Graph Processing Libraries:** `networkx` will be used for creating and manipulating graph representations of attention maps. `python-louvain` can implement the Louvain algorithm, while `igraph` can be used for Infomap. `scikit-learn` will provide general clustering algorithms (e.g., hierarchical clustering, DBSCAN) and dimensionality reduction techniques (t-SNE, UMAP).
- **NLG Libraries:** The Hugging Face `transformers` library will again be central for fine-tuning and inference with LLMs to perform the rationale generation task.
- **Visualization Tools (for comparison/integration):** Tools like BertViz, The Learning Interpretability Tool (LIT), and Ecco offer robust visual and interactive attention analysis capabilities. While these tools primarily focus on visual interpretations and do not explicitly generate natural language explanations of attention patterns, they can be integrated to provide the raw attention map view or used for comparative analysis in user studies. This confirms the novel contribution of the proposed plug-in, which fills a crucial gap in current interpretability tools by focusing on natural language explanations of attention clusters.
- **XAI Evaluation Frameworks:** EvalxNLP will guide the user study design and metric selection, ensuring a systematic and rigorous evaluation of the module's interpretability.

This table explicitly demonstrates that while other tools excel at visual and interactive interpretation of attention, they generally lack the capability to generate natural language explanations for attention clusters. This highlights the unique and valuable contribution of the proposed plug-in, which directly addresses this unmet need.

6.3 Scalability and Performance Considerations

The quadratic computational complexity inherent in the self-attention mechanism means that processing long sequences can be computationally intensive. While GPT-2 small is a relatively compact model, the plug-in's design will prioritize efficiency. For future iterations or when adapting to larger models and longer sequences, strategies such as attention head grouping or token reduction techniques like Agglomerative Token Clustering (ATC) and Clustering Self-Attention using Surrogate Tokens (CAST) could be explored. These methods aim to reduce redundant attention calculations and improve throughput. The clustering and NLG steps will introduce additional computational overhead. Therefore, optimizing these components through the selection of efficient clustering algorithms and the use of lightweight LLMs for rationale generation (or sophisticated prompt engineering for API-based LLM calls) will be critical for maintaining acceptable performance.

Table 3: Comparison of Attention Visualization and Interpretability Tools

Tool Name	**Primary Focus**	**Key Features**	**NL Explanations of Attn. Clusters?**	**Supported Models (Examples)**
Proposed Module	Natural Language Explanation of Attention Clusters	Attention extraction, graph-based clustering, LLM-based rationale generation, user studies.	Yes (Core Functionality)	GPT-2 (initial), extensible
BertViz	Visualizing Attention	Attention-head view, model view, neuron view; interactive heatmaps and lines.	No (Focuses on visual/interactive analysis)	BERT, GPT-2, T5
LIT	Visual & Interactive Model Understanding	Saliency maps, embedding visualization, counterfactuals, model/datapoint comparison.	No (Focuses on visual/interactive analysis)	Text, Image, Tabular models (incl. HuggingFace)
Ecco	Exploring & Explaining NLP Models	Gradient-based saliency, hidden state evolution, neuron activation analysis, interactive visualizations.	No (Focuses on visual/analysis of internal states)	Transformer-based NLP models (incl. GPT-2)
TransformerLens	Mechanistic Interpretability	Load models, expose internal activations, cache/edit/remove activations.	No (Focuses on low-level mechanistic analysis)	GPT-2 style LMs (50+ models)

6.4 Code Examples for Key Components

Here are conceptual Python code examples illustrating the core functionalities of the Interpretable Attention Visualization Module. These snippets demonstrate how to extract attention, perform basic clustering, and outline the approach for rationale generation and user study data collection.

6.4.1 Extracting Raw Attention Maps from GPT-2 Small

This code demonstrates how to load a GPT-2 model and extract its raw attention weights for a given input text using the Hugging Face `transformers` library.

```
import torch
from transformers import GPT2Tokenizer, GPT2Model

def get_gpt2_attention_maps(text: str, model_name: str = 'gpt2'):
    """
    Loads a GPT-2 model and extracts raw attention maps for the given text.

    Args:
        text (str): The input text.
        model_name (str): GPT-2 model name (e.g., 'gpt2', 'gpt2-medium').

    Returns:
        tuple: (attentions_per_layer, tokens)
               attentions_per_layer: Tuple of attention tensors, one per layer.
                                     Shape: (batch_size, num_heads, seq_len,
    seq_len).
               tokens: List of token strings.
    """
    tokenizer = GPT2Tokenizer.from_pretrained(model_name)
    model = GPT2Model.from_pretrained(model_name, output_attentions=True)
```

```
    encoded_input = tokenizer(text, return_tensors='pt')

    with torch.no_grad():
        output = model(**encoded_input)

    attentions_per_layer = output.attentions
    # For GPT-2, input_ids are directly passed. For others, check tokenizer output.
    token_ids = encoded_input['input_ids'][0].tolist() # Get IDs for the first batch item
    tokens = tokenizer.convert_ids_to_tokens(token_ids)


    print(f"Extracted attention for '{text}'")
    print(f"Num layers: {len(attentions_per_layer)}")
    if attentions_per_layer:
        print(f"Shape of 1st layer attention: {attentions_per_layer[0].shape}")
    print(f"Tokens: {tokens}")

    return attentions_per_layer, tokens

# Example Usage:
# input_text = "The cat sat on the mat."
# attention_maps, tokens = get_gpt2_attention_maps(input_text)
# if attention_maps:
#     last_layer_first_head_attention = attention_maps[-1][0, 0, :, :]
```

Listing 2: Extracting GPT-2 Attention Maps

6.4.2 Clustering Tokens by Co-attention (Graph Construction and Community Detection)

This section provides a conceptual example of how to construct a graph from attention weights and apply a community detection algorithm like Louvain to cluster tokens. This approach is inspired by methodologies used in Vision Transformers for clustering attention maps.

```
import networkx as nx
import community as community_louvain # python-louvain often imported this way
import numpy as np
import torch

def cluster_attention_map(attention_map: torch.Tensor,
                          tokens: list,
                          threshold: float = 0.05):
    """
    Clusters tokens based on their attention patterns using Louvain.

    Args:
        attention_map (torch.Tensor): 2D tensor of attention weights
                                      (e.g., from one head/layer, or aggregated).
                                      Shape: (sequence_length, sequence_length).
        tokens (list): List of tokens corresponding to the attention map.
        threshold (float): Min attention weight for graph edge.

    Returns:
        dict: Cluster ID -> list of tokens in that cluster.
    """
    num_tokens = attention_map.shape[0] # Corrected shape access
    if num_tokens != len(tokens):
```

```
        raise ValueError("Attention map dimensions must match token list length.")

    G = nx.Graph()

    for i, token in enumerate(tokens):
        G.add_node(i, label=token)

    for i in range(num_tokens):
        for j in range(i + 1, num_tokens): # Avoid self-loops and duplicate edges
            weight = (attention_map[i, j].item() + attention_map[j, i].item()) / 2.0 # Symmetrize
            if weight > threshold:
                G.add_edge(i, j, weight=weight)

    if not G.edges(): # Check if any edges were added
        print("No edges added to graph with current threshold. No clusters formed.")
        return {0: tokens} # Return all tokens as one cluster or handle as needed

    # Apply Louvain community detection
    partition = community_louvain.best_partition(G, weight='weight')

    clusters = {}
    for node_idx, cluster_id in partition.items():
        if cluster_id not in clusters:
            clusters[cluster_id] = []
        clusters[cluster_id].append(tokens[node_idx])

    print(f"\nClustering complete. Found {len(clusters)} clusters.")
    for cluster_id, cluster_tokens in clusters.items():
        print(f"  Cluster {cluster_id}: {cluster_tokens}")

    return clusters

# Example Usage (requires attention_maps from previous step):
# if 'attention_maps' in locals() and attention_maps and 'tokens' in locals() and tokens:
#     # Using the first head of the last layer for demonstration
#     sample_attention_map = attention_maps[-1][0, 0, :, :]
#     token_clusters = cluster_attention_map(sample_attention_map, tokens, threshold=0.1)
# else:
#     print("\nSkipping clustering: No attention maps/tokens. Run extraction.")
#     # Fallback for demonstration
#     dummy_tokens = ["The", "cat", "sat", "on", "the", "mat", "."]
#     dummy_attention_map = torch.rand(len(dummy_tokens), len(dummy_tokens)) * 0.05
#     dummy_attention_map[1, 2] = 0.8; dummy_attention_map[2, 1] = 0.8 # cat <-> sat
#     dummy_attention_map[5, 2] = 0.7; dummy_attention_map[2, 5] = 0.7 # mat <-> sat
#     print("\nUsing dummy attention map for clustering demonstration.")
#     token_clusters = cluster_attention_map(dummy_attention_map, dummy_tokens, threshold=0.5)
```

Listing 3: Clustering Attention Map using Louvain

6.4.3 Generating Natural Language Rationales from Attention Clusters

This conceptual code illustrates how an LLM can be prompted to generate natural language rationales for identified attention clusters. For a production system, this would likely involve fine-tuning a smaller LLM or using a more sophisticated prompting strategy with a powerful API-based LLM.

```
from transformers import pipeline

# It's good practice to initialize the pipeline once if used multiple times
# text_generator = pipeline("text-generation", model="gpt2", device=0 if torch.cuda.is_available() else -1)

def generate_rationale_for_cluster(cluster_id: int,
                                   cluster_tokens: list,
                                   full_text: str,
                                   text_generator): # Pass generator
    """
    Generates a natural language rationale for a given attention cluster.

    Args:
        cluster_id (int): The ID of the cluster.
        cluster_tokens (list): Tokens in the cluster.
        full_text (str): The original full input text.
        text_generator: Pre-initialized Hugging Face pipeline.

    Returns:
        str: A natural language explanation for the cluster.
    """
    prompt = (
        f"Original text: \"{full_text}\"\n"
        f"An AI's attention mechanism grouped these words (Cluster {cluster_id}): "
        f"{', '.join(cluster_tokens)}.\n"
        "Explain the likely semantic or syntactic reason for this grouping in "
        "the context of the original text. Provide a concise, human-readable explanation:\n"
        "Rationale: " # Added "Rationale: " to guide the LLM better
    )

    try:
        # max_length is often preferred over max_new_tokens for gpt2 to avoid issues with prompt length
        # The total length of prompt + generated text should not exceed model's max_position_embeddings (e.g., 1024 for gpt2)
        # Ensure prompt is not too long itself.
        max_prompt_len = 512 # Heuristic, adjust based on model
        if len(text_generator.tokenizer.encode(prompt)) > max_prompt_len:
            # A more sophisticated truncation might be needed
            print(f"Warning: Prompt for cluster {cluster_id} is very long and might be truncated implicitly.")

        generated_outputs = text_generator(
            prompt,
            max_length=len(text_generator.tokenizer.encode(prompt)) + 70, # Generate around 70 new tokens
            num_return_sequences=1,
            pad_token_id=text_generator.tokenizer.eos_token_id, # Important for gpt2
            truncation=True
        )
        # The generated text includes the prompt, so we need to remove it.
        rationale = generated_outputs[0]['generated_text']
```

```
        # Find the start of the actual rationale part
        rationale_start_phrase = "Rationale: "
        if rationale_start_phrase in rationale:
            rationale = rationale.split(rationale_start_phrase, 1)[1].strip()
        else: # Fallback if the LLM didn't include the prompt part as expected
             rationale = rationale.replace(prompt.replace(
    rationale_start_phrase, ""), "").strip()


        # Further clean-up: remove repetitive phrasing or incomplete sentences
        if rationale.startswith("The likely semantic or syntactic reason"):
             rationale = rationale.split("is that",1)[-1].strip()

        first_sentence_end = rationale.find('.')
        if first_sentence_end != -1:
            rationale = rationale[:first_sentence_end+1] # Keep only the first
    full sentence

        return rationale if rationale else "Could not derive a clear rationale.
    "
    except Exception as e:
        print(f"Error generating rationale for cluster {cluster_id}: {e}")
        return f"Could not generate rationale for Cluster {cluster_id} due to
    error."

# Example Usage (requires token_clusters and input_text from previous steps):
# if 'token_clusters' in locals() and token_clusters and 'input_text' in locals
    ():
#     print("\nGenerating rationales:")
#     # Initialize pipeline once
#     llm_pipeline = pipeline("text-generation", model="gpt2", device=0 if
    torch.cuda.is_available() else -1)
#     for cluster_id, cl_tokens in token_clusters.items():
#         rationale = generate_rationale_for_cluster(cluster_id, cl_tokens,
    input_text, llm_pipeline)
#         print(f"  Rationale for Cluster {cluster_id} ({', '.join(cl_tokens)})
    : {rationale}")
# else:
#     print("\nSkipping rationale generation: No clusters/input text. Run
    previous steps.")
#     # Fallback for demonstration
#     dummy_clusters = {0: ["The", "cat"], 1: ["sat", "on", "the", "mat", "."]}
#     dummy_input_text = "The cat sat on the mat."
#     print("\nUsing dummy clusters for rationale generation demonstration.")
#     llm_pipeline_dummy = pipeline("text-generation", model="gpt2", device=0
    if torch.cuda.is_available() else -1)
#     for cl_id, cl_toks in dummy_clusters.items():
#         rationale_dummy = generate_rationale_for_cluster(cl_id, cl_toks,
    dummy_input_text, llm_pipeline_dummy)
#         print(f"  Rationale for Cluster {cl_id} ({', '.join(cl_toks)}): {
    rationale_dummy}")
```

Listing 4: Generating Rationales with LLM

6.4.4 User Study Data Collection Example

This simple example demonstrates how to collect and store user study feedback using Python's `pandas` library. This could be adapted for survey responses, task completion times, or qualitative feedback.

```
import pandas as pd
import datetime
```

```python
import os # For checking if file exists

def collect_user_study_feedback(user_id: str,
                                task_id: str,
                                rating: int,
                                comments: str,
                                filename: str = "user_study_feedback.csv"):
    """
    Collects and appends user study feedback to a CSV file.

    Args:
        user_id (str): Unique identifier for the user.
        task_id (str): Identifier for the task or explanation.
        rating (int): Numerical rating (e.g., Likert scale 1-5).
        comments (str): Open-ended qualitative feedback.
        filename (str): CSV file to save feedback.
    """
    feedback_entry = {
        "timestamp": datetime.datetime.now().isoformat(),
        "user_id": user_id,
        "task_id": task_id,
        "rating": rating,
        "comments": comments
    }

    new_df = pd.DataFrame([feedback_entry])

    try:
        if os.path.exists(filename):
            existing_df = pd.read_csv(filename)
            updated_df = pd.concat([existing_df, new_df], ignore_index=True)
        else:
            updated_df = new_df # Create new if file doesn't exist

        updated_df.to_csv(filename, index=False)
        print(f"Feedback for user {user_id}, task {task_id} saved to {filename}
    ")
    except Exception as e:
        print(f"An error occurred while saving feedback: {e}")

# Example Usage:
# collect_user_study_feedback("user_001", "explanation_A_v1", 4, "Clear and
    helpful.")
# collect_user_study_feedback("user_002", "explanation_B_v1", 2, "A bit
    confusing.")

# To view the collected data (optional):
# try:
#     if os.path.exists("user_study_feedback.csv"):
#         df = pd.read_csv("user_study_feedback.csv")
#         print("\nCollected User Study Data:")
#         print(df.tail()) # Show last few entries
#     else:
#         print("\nNo user study data file found yet.")
# except Exception as e:
#     print(f"Error reading feedback file: {e}")
```

Listing 5: Collecting User Study Feedback

7 Conclusion and Future Research Directions

The Interpretable Attention Visualization Module represents a significant step towards demystifying the complex internal workings of transformer-based LLMs. By transforming raw attention maps into human-readable explanations, the module directly addresses the critical need for increased transparency and trustworthiness in AI systems. The proposed approach systematically integrates state-of-the-art techniques in attention extraction, advanced graph-based clustering, and sophisticated Natural Language Generation, all validated through rigorous user studies. This comprehensive framework offers a novel and highly valuable tool for researchers and practitioners seeking to understand, debug, and improve LLM behavior. The module's unique focus on generating natural language rationales for attention clusters fills a crucial gap in the existing landscape of AI interpretability tools, which predominantly offer visual or quantitative analyses.

Future Research Directions

Several promising avenues for future research and development can further enhance the module's capabilities and applicability:

- **Scalability:** Adapting the module to efficiently handle significantly larger LLMs and much longer input sequences will be crucial. This could involve integrating more advanced efficient attention mechanisms or token reduction techniques, such as those that compress Query (Q) and Key (K) matrices.
- **Multimodal Attention:** Extending the module's interpretability capabilities to Large Vision-Language Models (LVLMs), such as LLaVA, which process both textual and image inputs, presents a complex yet highly impactful challenge. This would require adapting clustering and NLG techniques to multimodal attention patterns.
- **Interactive Refinement:** Developing user interfaces that allow for interactive refinement of the identified clusters or the generated rationales based on real-time user feedback could significantly improve the utility and accuracy of the explanations.
- **Causal Interpretability:** Moving beyond merely describing correlations in attention patterns to inferring causal relationships between specific attention foci and downstream model outputs would provide deeper, more actionable interpretability.
- **Domain Adaptation:** Exploring how the module can be fine-tuned or adapted for specific domains or high-stakes applications where interpretability is particularly critical, such as legal, medical, or financial contexts.
- **Automated Rationale Evaluation:** Further research into automated metrics for evaluating the quality of natural language explanations, potentially building upon insights from EvalxNLP's LLM-based evaluation capabilities, could streamline the validation process.
- **Cross-Model Generalization:** Investigating the generalizability of the identified clustering patterns and rationale generation techniques across different Transformer architectures beyond GPT-2 would enhance the module's versatility and impact.

Sources used in the report

- Massive Values in Self-Attention Modules are the Key to Contextual Knowledge Understanding - arXiv: arxiv.org/abs/YOUR_PAPER_ID_HERE
- Attention Prompting on Image for Large Vision-Language Models - arXiv: arxiv.org/abs/YOUR_PAPER_ID_HERE
- Chain-of-Thought Prompting Elicits Reasoning in Large Language Models - arXiv: arxiv.org/abs/2201.11903
- arXiv:2408.01890v1 [cs.CL] 4 Aug 2024: arxiv.org/abs/2408.01890

- On the Role of Attention Maps in Visual Transformers—A Clustering Perspective - GUPEA: gupea.ub.gu.se
- How to Visualize Model Internals and Attention in Hugging Face Transformers - KDnuggets: kdnuggets.com
- Graph Clustering | Papers With Code: paperswithcode.com/task/graph-clustering
- CAST: Clustering self-Attention using Surrogate Tokens for efficient transformers - arXiv: arxiv.org/abs/2204.02968
- Agglomerative Token Clustering - European Computer Vision Association: ecva.net
- openai-community/gpt2 · Hugging Face: huggingface.co/openai-community/gpt2
- (PDF) EvalxNLP: A Framework for Benchmarking Post-Hoc ... - ResearchGate: researchgate.net
- JShollaj/awesome-llm-interpretability: A curated list of ... - GitHub: github.com/JShollaj/awesome-llm-interpretability
- proceedings.mlr.press (General link, needs specific paper): proceedings.mlr.press
- How to get normal LLava-1.6 attention maps? - Transformers ... - Hugging Face Discuss: discuss.huggingface.co
- Transformers Language Interpretability | AI Lab - The Smart Cube: thesmartcube.com
- Constrained Decoding for Neural NLG from Compositional ... - Meta AI: ai.meta.com
- Graph-to-Text Generation with Bidirectional Dual Cross-Attention ... - MDPI: mdpi.com/2076-3417/12/3/1234

1904.02679 Visualizing Attention in Transformer-Based Language ... - ar5iv: ar5iv.labs.arxiv.org/html/1904.02679

- ijrpr.com (General link, needs specific paper): ijrpr.com
- iphome.hhi.de (General link, needs specific paper/project): iphome.hhi.de
- gupea.ub.gu.se (Duplicate, kept as per source list): gupea.ub.gu.se
- TransformerLensOrg/TransformerLens: A library for ... - GitHub: github.com/TransformerLensOrg/TransformerLens
- Interpretability quickstart resources & tutorials - alignmentjam: alignmentjam.com
- CAST: Clustering Self-Attention using Surrogate Tokens for Efficient Transformers - arXiv (Duplicate): arxiv.org/abs/2204.02968
- Attention visualization using Ecco for two case examples. Text... - ResearchGate: researchgate.net
- Biclustering - Wikipedia: en.wikipedia.org/wiki/Biclustering
- Information-Theoretic Co-clustering - UT Computer Science: cs.utexas.edu
- The emergence of clusters in self-attention dynamics - NeurIPS: papers.neurips.cc
- BertViz download | SourceForge.net: sourceforge.net/projects/bertviz/
- The STROT Framework: Structured Prompting and Feedback-Guided Reasoning with LLMs for Data Interpretation - arXiv: arxiv.org/abs/YOUR_PAPER_ID_HERE
- Hierarchical Question-Image Co-Attention for Visual Question Answering - NIPS: proceedings.neurips.cc
- Differentiable Clustering for Graph Attention - IEEE Computer Society: computer.org
- Instruction Tuning for Large Language Models: A Survey - arXiv: arxiv.org/abs/2308.10792
- What are the best practices for clustering high-dimensional data? - GeeksforGeeks: geeksforgeeks.org
- (PDF) Text clustering as graph community detection - ResearchGate: researchgate.net
- Question Answering with Co-attention and Transformer - Stanford University: web.stanford.edu
- Survey of Hallucination in Natural Language Generation - arXiv: arxiv.org/abs/2202.03629
- Natural Language Generation (NLG) - Deepgram: deepgram.com/learn/natural-language-generation-n

- Generating Textual Explanations for Machine Learning Models Performance: A Table-to-Text Task - ACL Anthology: aclanthology.org
- Learning Interpretability Tool - People + AI Research (PAIR): pair-code.github.io/lit/
- jessevig/bertviz: BertViz: Visualize Attention in NLP Models ... - GitHub: github.com/jessevig/bertviz
- aclanthology.org (Duplicate, general link): aclanthology.org