Goal and instructions of comment analysis for GitHubCopilot

Step-by-Step Guide for GitHub Copilot

Step 1: **Data Preparation**

1. **Load the comments into a structured DataFrame**:
   - **Input**: JSON or dictionary format where each entry contains:
     - `Filename`: e.g., `blocks.10.attn.hook_pattern-head_1-prompt_15.png`
     - `Comment`: The text comment associated with each image.
   - **Output DataFrame**: Columns for `Layer`, `Head`, `Comment`, `Pattern_Category` (extracted keywords describing the pattern type).
2. **Extract metadata from the filename**:
   - **Filename Parsing**: Define a function that extracts:
     - `Layer` (e.g., `blocks.10` → `Layer 10`).
     - `Head` (e.g., `head_1` → `Head 1`).
   - **Pattern Tagging**: Use keywords from the comments to tag specific pattern categories. For example:
     - Keywords like "cls tokens" or "vertical sep lines" should be categorized as distinct patterns.
   - Store these in `Pattern_Category` in the DataFrame for easy filtering and analysis.

Step 2: **Textual Pattern Extraction and Vectorization**

1. **Vectorize Comments**:
   - **Input**: `Comment` column in the DataFrame.
   - **Method**: Use TF-IDF vectorization to convert comments into numerical vectors that capture term frequency and relevance.
   - **Output**: A matrix of vectorized comments to be used for similarity and clustering analysis.
2. **Save Extracted Patterns for Analysis**:
   - Define distinct pattern categories based on keyword matching or term frequency, stored in the `Pattern_Category` column.

Step 3: **Similarity Analysis Between Comments**

1. **Calculate Cosine Similarity**:
   - **Input**: Vectorized comments.

- **Method**: Compute pairwise cosine similarity to measure the functional similarity between comments. This helps to compare heads across layers.
- **Output**: Similarity matrix for all comments, stored for clustering and visualization.

2. **Cluster Comments by Functional Similarity**:
   - **Input**: Similarity matrix from the previous step.
   - **Method**: Use a clustering algorithm (e.g., k-means, hierarchical clustering) to group comments by similarity, which will help identify functionally similar attention heads.
   - **Output**: Cluster assignments added to the DataFrame, allowing for easy retrieval of similar heads and layers.

Step 4: **Layer and Head Pattern Analysis**

1. **Intra-Layer Analysis**:
   - Aggregate patterns within each layer to determine if there's a dominant behavior or diversity in functions among heads.
   - Summarize each layer's most frequent patterns to identify heads with similar or unique behaviors.
2. **Inter-Layer Analysis**:
   - Compare pattern frequencies across layers to find overarching trends, like whether certain attention patterns (e.g., "vertical sep lines") concentrate in certain layers.
   - Output a summary of patterns that shows how attention behaviors evolve across layers.

Step 5: **Visualization of Patterns and Similarities**

1. **Generate Heatmaps for Pattern Frequency**:
   - Use the aggregated `Pattern_Category` counts to plot a heatmap, visualizing pattern concentration across layers and heads.
   - **Objective**: Show the distribution of each pattern across layers and heads, highlighting where certain patterns dominate.
2. **Generate Clustered Pattern Similarity Map**:
   - Plot the clusters of similar patterns to see which heads behave similarly, even across different layers.
   - **Objective**: Identify functional similarity across heads with a graphical representation.

Step 6: **Summary and Functional Matrix**

1. **Generate a Matrix Summary of Head Functions**:
   - Create a table where each row represents a unique function (based on the identified clusters), and each column represents a head. Populate the matrix to

show which functions align with specific heads.

2. **Document Observations**:
   - Automate the generation of a summary report listing the most common functions per layer and head group.
   - Use this as the basis for discussing layer-specific functions or recurring attention behaviors across the BERT model.

Final Notes

1. The main focus is to automate as much of the pattern recognition and similarity analysis as possible using the comments as a guide.
2. The implementation should modularize each step so functions are reusable and can be refined independently.
3. Consider adding functions to export results (e.g., to CSV or visualizations saved as images) to facilitate later analysis.