2. **Semantic Coherence**: ChunkKV preserves the semantic relationships within a chunk, crucial to understanding nuances such as the difference between primary and occasional food sources for pandas.

3. **Information Density**: A single chunk can contain multiple relevant tokens in their proper context, potentially retaining more useful information within the same compressed cache size compared to discrete methods.

4. **Reduced Ambiguity**: Discrete methods might retain the token "eat" from various sentences about different animals. ChunkKV ensures that "eat" is preserved specifically in the context of pandas in the wild.

5. **Temporal and Logical Flow**: ChunkKV can maintain the sequence of ideas present in the original text, preserving any temporal or logical progression that may be crucial for understanding.

A.4. Implications for Model Performance

This analysis suggests several key implications for model performance:

- **Improved Accuracy**: By retaining contextually rich information, ChunkKV enables more accurate responses to queries, especially those requiring nuanced understanding.
- **Enhanced Long-context Processing**: Preservation of semantic chunks allows for better handling of long-range dependencies and complex reasoning tasks.
- **Reduced Computational Overhead**: Although both methods compress the KV cache, ChunkKV's approach may reduce the need for extensive context reconstruction, potentially improving inference efficiency.
- **Versatility**: The chunk-based approach is likely to be more effective across a wide range of tasks and domains as it preserves the natural structure of language.

This in-depth analysis demonstrates why ChunkKV is more effective in preserving semantic information in long contexts. By retaining coherent chunks of text, it provides language models with more contextually rich and semantically complete information, leading to improved performance in tasks that require deep understanding and accurate information retrieval from extensive documents.

B. Additional Experiments

B.1. Layer-Wise Index Reuse

B.1.1. Efficiency Analysis

The layer-wise index reuse method significantly reduces the computational complexity of ChunkKV. Without index reuse, ChunkKV would be applied to all N_{layers} layers, resulting in a total compression time of $N_{\text{layers}} \cdot T_{\text{compress}}$, where T_{compress} is the time taken to compress one layer. With index reuse, ChunkKV is only applied to $\frac{N_{\text{layers}}}{N_{\text{reuse}}}$ layers, reducing the total time to $\frac{N_{\text{layers}}}{N_{\text{reuse}}} \cdot T_{\text{compress}} + (N_{\text{layers}} - \frac{N_{\text{layers}}}{N_{\text{reuse}}}) \cdot T_{\text{select}}$, where T_{select} is the time taken to select indices, which is typically much smaller than T_{compress}. This results in a theoretical speedup factor of:

$$\text{Speedup} = \frac{N_{\text{layers}} \cdot T_{\text{compress}}}{\frac{N_{\text{layers}}}{N_{\text{reuse}}} \cdot T_{\text{compress}} + (N_{\text{layers}} - \frac{N_{\text{layers}}}{N_{\text{reuse}}}) \cdot T_{\text{select}}}$$

Assuming T_{select} is negligible compared to T_{compress}, this simplifies to approximately N_{reuse}. In practice, the actual speedup may vary depending on the specific implementation and hardware, but it can still lead to substantial time savings, especially for models with a large number of layers.

B.1.2. Layer-Wise Index Similarity

This section details the experiment of layer-wise index reuse similarity described in Section 3.3. The inference prompt is randomly selected from the LongBench benchmark, and the preserved indices for H2O, SnapKV, and ChunkKV are saved in