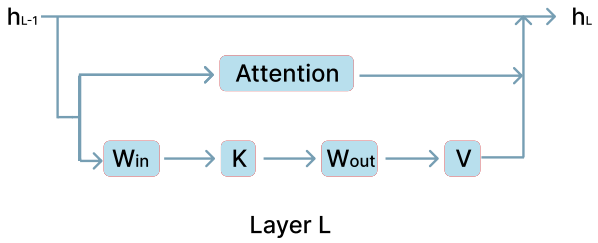

Fig. 1: Diagram for Transformer layer

A Research Background

A.1 Details of the Knowledge Editing Method

Background. As is known to all, an MLP layer consists of a first linear transformation (input weight W_{in}), a non-linear activation function, and a second linear transformation (output weight W_{out}).

To help readers understand the core mechanisms of knowledge editing, we briefly introduce the algorithmic principles using MEMIT, a mainstream algorithm, as an example.

MEMIT is an algorithm for editing factual knowledge in large language models by directly modifying specific transformer layer parameters. At its core, MEMIT first identifies a range of critical MLP layers that mediate factual recall through causal analysis. For the same LLM, the range of selected MLP layers is fixed. Afterward, for each fact i to be edited, it computes a target hidden state vector (z_i) that would correctly encode the new association. Then, MEMIT distributes the necessary changes across multiple layers by treating each MLP layer as an associative memory that maps subject representations to fact-related information. The key innovation is calculating optimal weight updates (update on W_{out}, as shown in Figure 1) that minimize disruption to existing knowledge while effectively storing new memories. This is achieved by solving a mathematical optimization problem that balances preserving old associations with learning new ones, using statistics from the model's previous inputs to ensure changes are minimal yet effective. Actually, in the above process, the only part of the model that is modified is the output weight W_{out} of the selected MLP layers.

A.2 Example for Evaluation Metrics

In order to help readers better understand the evaluation metrics, we give an example.

Suppose the LLM lacks knowledge about Messi's transfer from Paris Saint-Germain to Inter Miami CF. To address this, we utilize the statement "The football club Messi currently plays for is Inter Miami CF" as the edited fact to perform knowledge editing. Table 1 shows examples of test prompts for evaluating these five metrics, with one example provided for each metric.

Table 1: Examples of testing prompts for different metrics

Metrics	Testing Example
ES	The football club Messi currently plays for is
PS	Messi's current employer in the football world is
NS	Which football club has Ronaldo on its roster? The answer is
GE	List the information of the club that Messi is currently playing for.
RS	

B KEditVis

B.1 Knowledge Graph

The implementation of knowledge graphs follows a three-step pipeline: first, we retrieve and pre-process relevant Wikipedia content for the user-selected word; second, we apply GPT-4 API for semantic parsing to extract subject-predicate-object triples; and finally, we construct a knowledge graph with the user-selected word as the central node, limited to a two-hop neighborhood. This approach balances information density with relevance, allowing dynamic exploration of any keyword without pre-built databases.

B.2 Visualization Algorithm for Wireframe Layout

We designed a three-step visualization algorithm that determines each wireframe's horizontal length and its connection point on the layer rectangle. The following is a textual description of the algorithm.

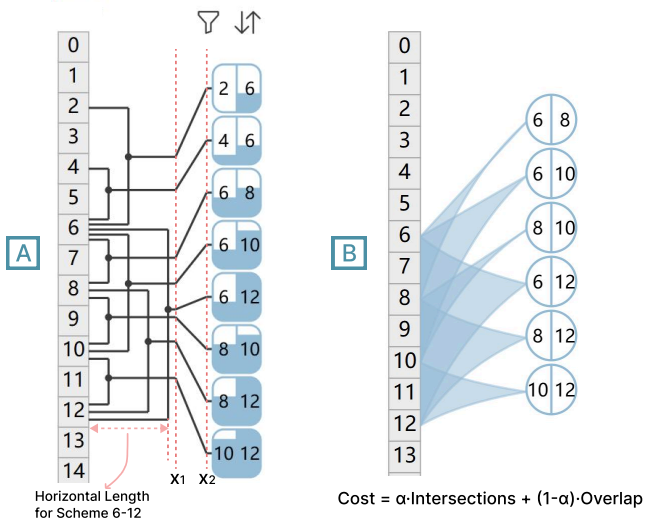

Fig. 2: Set Visualization: (A) Diagram of set visualization our designed; (B) The curved edge band solution we tried.

- **Step 1**. We generate a sorted list of layer selection schemes from smallest to largest range, with identical ranges sorted by layer values.
- **Step 2**. We traverse each scheme based on the sorting from step 1. During traversal, we calculate division points needed for each layer rectangle's right edge and determine wireframe horizontal lengths. The calculation of the horizontal length of each wireframe: Starting with unit length x, we check if there are any existing wireframes (i.e., wireframes that have already been traversed and whose lengths have been calculated) that have the same length and overlapping layer ranges. If found, the length increases by x; otherwise, it's confirmed. This approach minimizes distinct horizontal wireframe lengths.
- **Step 3**. We traverse each layer to determine wireframe connection points. When traversing layer n, we analyze the other number in schemes containing n. Connection points for numbers greater than n are placed below those for numbers less than n, with larger numbers connecting progressively higher. Take layer 6 in Figure 2A) as an example: when we traverse to layer 6, there are five schemes that include 6 at either end: 2-6, 4-6, 6-8, 6-10, and 6-12. We then examine the other number in each of these schemes (besides 6) to determine positioning. First, the connection points for schemes 2-6 and 4-6 must be placed above the schemes 6-8, 6-10, and 6-12 (because 2 and 4 are less than 6, while 8, 10, and 12 are greater than 6). Within each group, we ensure that schemes with larger numbers have connection points positioned higher (so the connection point for scheme 4-6 is positioned higher than scheme 2-6, and the connection point for scheme 6-12 is positioned higher than scheme 6-10).

After executing the above algorithm, the section to the left of the dashed line x_1 (see Figure 2A) is effectively optimized. For the section between the dashed lines x_1 and x_2, we only need to sort the layer selection schemes according to the order of the intersection points between the wireframes and x_1 when the user clicks the sort icon (the sort icon is positioned above the layer selection schemes). This ensures that there are no crossing points between x_1 and x_2.

The following is the visualization algorithm's pseudocode and time complexity analysis.

Algorithm 1 Visualization Algorithm for Wireframe Layout

Input: Set of layer selection schemes $S = \{(l_i, u_i)\}$ where l_i and u_i are lower and upper bounds
Output: Wireframe horizontal lengths and connection points
Step 1: Sort schemes
Sort schemes in S by range size $(u_i - l_i)$ ascending, then by layer values (l_i, u_i) ascending
$sorted\_schemes \leftarrow$ sorted list of schemes
Step 2: Calculate wireframe horizontal lengths
$x \leftarrow$ unit length
$wireframe\_lengths \leftarrow \{\}$
for each scheme (l, u) in $sorted\_schemes$ **do**
  $current\_length \leftarrow x$
  while exists wireframe with length $current\_length$ and overlapping layer range **do**
    $current\_length \leftarrow current\_length + x$
  end while
  $wireframe\_lengths[(l, u)] \leftarrow current\_length$
end for
Step 3: Determine connection points for each layer
$connection\_points \leftarrow \{\}$
for each layer n from 0 to max_layer **do**
  $schemes\_containing\_n \leftarrow$ schemes where $l = n$ or $u = n$
  $other\_numbers \leftarrow \{\}$
  for each scheme (l, u) in $schemes\_containing\_n$ **do**
    if $l = n$ **then**
      add u to $other\_numbers$
    else
      add l to $other\_numbers$
    end if
  end for
  $numbers\_less\_than\_n \leftarrow$ numbers in $other\_numbers$ where number $< n$
  $numbers\_greater\_than\_n \leftarrow$ numbers in $other\_numbers$ where number $> n$
  Sort $numbers\_less\_than\_n$, $numbers\_greater\_than\_n$ in descending order
  $point\_index \leftarrow 1$
  for each number in $numbers\_less\_than\_n$ **do**
    $connection\_points[n, (\text{scheme with } n \text{ and number})] \leftarrow point\_index$
    $point\_index \leftarrow point\_index + 1$
  end for
  for each number in $numbers\_greater\_than\_n$ **do**
    $connection\_points[n, (\text{scheme with } n \text{ and number})] \leftarrow point\_index$
    $point\_index \leftarrow point\_index + 1$
  end for
end for
return $wireframe\_lengths$, $connection\_points$

Time complexity analysis (let m denote the number of layer selection schemes and L denote the maximum layer number):

- Step 1: $O(m \log m)$ for sorting m schemes by range size and layer values using comparison-based sorting.
- Step 2: $O(m^2)$ in the original method, as each scheme checks overlap with all previously processed schemes. With interval tree optimization, this can be reduced to $O(m \log m)$ by maintaining wireframe lengths and ranges in an interval tree for $O(\log m)$ overlap queries.
- Step 3: Let k_l denote the number of schemes containing layer l. The original method requires $O(\sum_{l=1}^{L} k_l \log k_l)$ for sorting schemes at each layer. Assume an extremely worst case where $k_l = m$ for all l, and then the time complexity is $O(L \times m \log m)$. However, in reality, the actual complexity is obviously much better.

Overall complexity: Original method achieves $O(m^2 + \sum_{l=1}^{L} k_l \log k_l)$, and the worst-case bound is $O(m^2 + L \times m \log m)$. With interval tree optimization, they are reduced to $O(m \log m + \sum_{l=1}^{L} k_l \log k_l)$ and $O(L \times m \log m)$ respectively.

B.3 Design Alternatives

B.3.1 Matrix Visualization for Token Probability

Our design alternatives for token probability include the matrix visualization, as shown in Figure 3. The reasons for finally choosing the ranking chart have been stated in the paper.

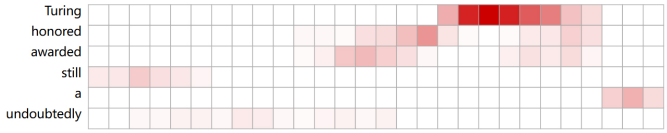

Fig. 3: Matrix visualization showing token probability data captured when using "The first Turing Award winner is" as model input.

B.3.2 Band Design for Set Visualization

We evaluated several alternatives before selecting wireframes as our approach. One alternative is curved edge bands, as shown in Figure 2B. To eliminate visual clutter, we optimized the scheme ordering using a greedy strategy, with an objective function based on the weighted sum of intersection points and overlapping areas. Figure 2B displays the results after optimization. As can be seen, despite these optimization efforts, significant overlap persisted, hindering the intuitive perception of set relationships.

B.4 Implementation

KEditVis uses a client-server architecture with Vue.js for the front-end and Python Flask for the back-end. LLMs are accessed through Hugging Face, and knowledge editing procedures are performed on a server equipped with an NVIDIA A100 (80GB) GPU.

C Usage Example

The two usage scenarios provided in the paper present intricate narratives with multiple analytical ideas and insights. To help readers clearly understand the system's workflow, we provide a straightforward usage example as a supplement.

Suppose John is an LLM expert. After selecting the model to be edited, he first engages in dialogue through the LLM chat view to check the model's answers to factual questions. At the same time, he can generate knowledge graphs for entities of interest, and then generate corresponding factual questions in LLM chat view by examining the relationships between different entities in the knowledge graphs. When he discovers knowledge he wants to edit, for example, when the model responds to a certain factual question with an incorrect answer, he adds a fact to be edited in the "facts for editing" list below.

Next, to deepen the model's memory of the edited fact, he clicks the "+" button and in the popup window instructs the system to automatically generate another fact with the same meaning but expressed differently. This way, both facts will be used together for editing in the subsequent stage. Then he clicks the "+" button

and, in the popup window, requests the system to automatically generate several test prompts. After reviewing them, he modifies very few items, ultimately obtaining satisfactory prompts for testing.

Subsequently, John double-clicked on the edited fact, immediately displaying a cosine similarity bar chart and token ranking chart. Based on the trends in the bar chart, John selected a series of layers, and then clicked the "Recommend" button, at which point the system automatically suggested several layer selection schemes. Afterward, combining the trends observed in the cosine similarity bars and the ranking chart, he added several more layer selection schemes. In general, during this process, he selected several schemes that he considered reasonable by analyzing and making trade-offs.

John compared the editing results of various schemes. For schemes with close results or poor outcomes, he expanded the details of the results and analyzed the reasons for the poor output results. Simultaneously, by observing the layer scope of these schemes through set visualization, he derived several conclusions and insights. Furthermore, by comparing the editing results among different schemes and the cosine similarity bars and token ranking charts before and after editing, he selected what he considered to be the optimal editing scheme. However, John found that there were still indicators in the scheme that were not satisfactory, so he constructed corresponding facts, selected several layer selection schemes, and performed a second round of editing. Among the various options, he conducted comparisons and analyses, selecting a satisfactory scheme.

Finally, John checked the drift view to observe the offset of the hidden states before and after the edits. He saw that the scatter points before and after the edits were basically the same, and he zoomed in to check again. He also sorted by drift value in the table on the right, browsed through some of the instances where the output changed after editing, and confirmed that the edits had no obvious negative impact on the model.

D Details of the User Study

D.1 Tasks

In each task, we will ask participants to edit the specified subject into the specified target answer, and we will provide a sample fact for editing as an example. Next, participants will go through the following five stages.

- **Stage 1.** Participants are asked to start from the LLM chat view to examine the model's current output.
- **Stage 2.** Participants are asked to construct facts for editing and prompts for testing.
- **Stage 3.** Participants are asked to select layers based on the cosine similarity bar chart and token ranking chart.
- **Stage 4.** Participants are asked to analyze and compare different schemes. They can review the overall metrics of different schemes, detailed results, and changes in the cosine similarity bar charts and token ranking charts before and after editing. Here, they can repeatedly execute stage 3, adding other schemes of interest. Ultimately, they are required to select what they consider to be the optimal scheme.
- **Stage 5.** Participants are asked to use and examine the drift view to analyze the global impact of the edits on the model.

D.2 Results of SUS Questionaire

The raw results of the SUS questionnaire are shown in Table 2.

According to the SUS standard guidelines, for even-numbered questions, lower scores are better, while for odd-numbered questions, higher scores are better. Therefore, we converted the scores for even-numbered questions from $x \in [1,5]$ to $5-x$, and the scores for odd-numbered questions from $x \in [1,5]$ to $x-1$. The processed results are shown in Table 3.

Then, we multiplied the sum of all scores by 2.5, resulting in a final SUS score of 86.25. Based on research on SUS score factors, we also calculated detailed usability and learnability scores. The usability score was 86.98 (sum of Q1-3 and Q5-9 scores, multiplied by 3.125), and the learnability score was 83.33 (sum of Q4 and Q10 scores, multiplied by 12.5).

Table 2: Raw results of the SUS questionnaire

Participants	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
P1	4	2	4	2	4	2	5	2	5	3
P2	5	2	5	1	5	1	5	2	4	1
P3	4	2	4	2	5	1	4	2	4	1
P4	4	2	3	2	4	2	5	2	5	2
P5	4	3	3	2	4	2	4	2	4	3
P6	5	1	5	1	4	1	5	1	5	1
P7	5	1	5	1	4	1	5	1	5	1
P8	5	2	4	2	5	1	5	2	5	1
P9	5	1	4	1	5	1	4	1	5	1
P10	5	2	5	2	4	2	5	2	5	3
P11	5	1	5	1	5	1	5	2	4	1
P12	4	2	4	2	5	1	4	1	4	3

Table 3: Processed results for SUS questionnaire

Participants	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
P1	3	3	3	3	3	3	4	3	4	2
P2	4	3	4	4	4	4	4	3	3	4
P3	3	3	3	3	4	4	3	3	3	4
P4	3	3	2	3	3	3	4	3	4	3
P5	3	2	2	3	3	3	3	3	3	2
P6	4	4	4	4	3	4	4	4	4	4
P7	4	4	4	4	3	4	4	4	4	4
P8	4	3	3	3	4	4	4	3	4	4
P9	4	4	3	4	4	4	3	4	4	4
P10	4	3	4	3	3	3	4	3	4	2
P11	4	4	4	4	4	4	4	3	3	4
P12	3	3	3	3	4	4	3	4	3	2

D.3 Cosine Similarity Analysis

In the post-hoc analysis of the user study, we assessed the global impact on the model caused by participants and baselines through a cosine similarity analysis between pre-edit and post-edit hidden states, computed before t-SNE projection. The results (shown in Table 4) showed that the cosine similarity values were all extremely close to 1 (all values rounded to 7 decimal places), indicating minimal effects on the model's general performance.

Table 4: Cosine similarity analysis between pre-edit and post-edit hidden states

Task	Method	Cosine similarity
Task 1	Baseline I	0.9999993
	Baseline II	0.9999974
	Baseline III	0.9999979
	Participants' average	0.9999979
Task 2	Baseline I	1.0000000
	Baseline II	0.9999999
	Baseline III	0.9999999
	Participants' average	0.9999997
Task 3	Baseline I	0.9999943
	Baseline II	0.9999983
	Baseline III	0.9999983
	Participants' average	0.9999974

D.4 Output Changes Examination

In the post-hoc analysis of the user study, we also evaluated whether the model's outputs showed semantic changes before and after editing by examining the outputs of 1000 samples in the drift view. The results (shown in Table 5) show that the outputs for the vast majority of samples did not change, with only a small number showing changes, and among

these, most maintained the same semantic meaning (despite changes in content). Specifically, regardless of task or whether we're examining the baselines or participant-selected schemes, more than 94% of sample outputs remained unchanged. Among the samples with output changes, over 70% maintained semantic consistency. The number of samples with semantic changes did not exceed 8 in any case, with the majority of these changes representing corrections from incorrect to correct answers. This therefore further validates that the global impact of the editing on the model is negligible.

Table 5: Analysis of output changes across 1000 samples

Task	Method	Number of samples				
		No change	Changed	Semantic preserved	W→R	R→W
Task 1	B-I	975	25	18	6	1
	B-II	964	36	29	4	3
	B-III	968	32	28	3	1
	P-avg	967.7	32.3	28.1	3.1	1.2
Task 2	B-I	995	5	5	0	0
	B-II	988	12	10	1	1
	B-III	988	12	10	1	1
	P-avg	988.9	11.1	8.2	1.9	1.0
Task 3	B-I	944	56	48	5	3
	B-II	962	38	35	3	0
	B-III	962	38	35	3	0
	P-avg	957.7	42.3	38.5	2.5	1.3

Note: B-I/II/III = Baseline I/II/III; P-avg = Participants average; W→R = Wrong to Right; R→W = Right to Wrong

D.5 Semantic Changes Examination

We conducted a preliminary supplementary test on our additionally constructed benchmark (100 test samples for each task), which contains similar concepts to those that were edited. This benchmark was constructed by first automatically generating content using the Claude Sonnet 4.5 model, followed by manual inspection. We tested both the editing schemes selected by participants and those selected by baselines on the benchmark to determine how many samples experienced semantic changes before and after editing. In the future, we can continue to expand the scale of test samples.

Results. The results (shown in Table 6) showed that the participants' edits preserved 98.6%, 82.1%, 68.8% of answers to these queries in the three tasks respectively. This indicates that the semantics of most similar concepts are not affected by these edits. Moreover, we conducted one-sample t-tests to compare participants' edits against each individual baseline method across three tasks. In Task 1, participants significantly outperformed Baseline II ($t(11) = 3.000$, $p < 0.05$) but were comparable to Baseline I ($t(11) = -0.200$, $p = 0.845$) and Baseline III ($t(11) = -1.000$, $p = 0.339$). For Task 2, participants underperformed Baseline I ($t(11) = -3.182$, $p < 0.01$) but significantly outperformed Baseline II ($t(11) = 11.000$, $p < 0.001$) and Baseline III ($t(11) = 11.000$, $p < 0.001$). In Task 3, participants underperformed Baseline I ($t(11) = -19.282$, $p < 0.001$) while being comparable to Baseline II ($t(11) = -1.483$, $p = 0.166$) and Baseline III ($t(11) = -1.483$, $p = 0.166$). Overall, there is little difference between the edits made by participants and those made by baselines.

Reflection. We speculate that the reason why participants' edits in Task 2 and Task 3 are slightly lower than Baseline I is that participants' editing strategies are more aggressive, aiming for effective edits, and they typically consider the synthesis and trade-offs between various metrics. They consider the impact of editing on similar concepts but not this aspect alone. Since Baseline I has weaker editing effects than participants' edits, this may also be one reason why it has slightly less impact on similar concepts.

E Runtime Cost

We experimented and collected statistics on the time required to perform knowledge edits in our system with different editing layers and different numbers of facts for editing. Specifically, we conducted dedicated runtime testing for knowledge editing functions in our source code. Before timing, we performed a GPU warm-up step to ensure consistent results. By varying the number of facts and the number of editing layers, we collected the performance data presented in Table 7.

Table 6: Results of evaluation on the additional 100 test samples

Task	Method	No semantic change
Task 1	Baseline I	98
	Baseline II	94
	Baseline III	99
	Participants' average	98.6
Task 2	Baseline I	85
	Baseline II	72
	Baseline III	72
	Participants' average	82.1
Task 3	Baseline I	71
	Baseline II	69
	Baseline III	69
	Participants' average	68.8

Table 7: Runtime (in seconds) for different numbers of facts and layers

Number of Facts	Number of Editing Layers							
	1	2	3	4	5	6	7	8
1	2.1772	3.1964	5.1587	6.3586	7.7783	9.0310	10.4337	11.7917
3	3.8359	4.4687	6.0355	7.5027	9.0522	10.5649	12.0889	13.6447
10	4.8686	8.9297	10.6578	12.4918	14.4483	16.2602	18.2453	20.1653
25	11.5129	13.5332	16.6905	19.8711	22.9720	26.1024	29.3217	32.3459
50	17.0668	20.9990	25.6857	30.9108	35.4627	41.1918	45.9054	50.9609

The Number of Editing Layers. Since too many editing layers can lead to over-editing and poor editing effects, testing an excessive number of editing layers is not meaningful. We tested layers ranging from 1 to 8 (8 layers is already quite substantial and often leads to over-editing, so increasing beyond this point is not meaningful). The results show that runtime cost is positively correlated with the number of editing layers, but even with 8 layers of editing, the time consumption is not substantial.

The Number of Editing Facts. As can be seen, even when the number of edited facts reaches 50, the editing process does not consume excessive time. Furthermore, the results reveal a sublinear relationship between runtime and the number of facts. While runtime does increase as facts are added, the growth rate (the first derivative of the growth curve) remains below 1. This sublinear growth pattern suggests that even when processing substantially larger fact sets (e.g., 100 or several hundred facts), the expected runtime would remain within reasonable bounds, not exceeding a few minutes.