InstructEdit: Instruction-Based Knowledge Editing for Large Language Models

Ningyu Zhang¹, Bozhong Tian¹, Siyuan Cheng², Xiaozhuan Liang², Yi Hu², Kouying Xue², Yanjie Gou², Xi Chen², Huajun Chen¹
¹ Zhejiang University
² Tencent
{zhangningyu, tbozhong}@zju.edu.cn

Abstract
Knowledge editing for large language models can offer an efficient solution to alter a model’s behavior without negatively impacting the overall performance. However, the current approaches encounter issues with limited generalizability across tasks, necessitating one distinct editor for each task, significantly hindering the broader applications. To address this, we take the first step to analyze the multi-task generalization issue in knowledge editing. Specifically, we develop an instruction-based editing technique, termed InstructEdit, which facilitates the editor’s adaptation to various task performances simultaneously using simple instructions. With only one unified editor for each LLM, we empirically demonstrate that InstructEdit can improve the editor’s control, leading to an average 14.86% increase in Reliability in multi-task editing setting. Furthermore, experiments involving holdout unseen task illustrate that InstructEdit consistently surpass previous strong baselines. To further investigate the underlying mechanisms of instruction-based knowledge editing, we analyze the principal components of the editing gradient directions, which unveils that instructions can help control optimization direction with stronger OOD generalization.

1 Introduction
Knowledge editing [Sinitsin et al., 2020; Yao et al., 2023; Wang et al., 2023c; Mazzia et al., 2023; Zhang et al., 2023a; Zhang et al., 2024¹] aims to enable efficient and targeted post-hoc modifications in the parametric knowledge within Large Language Models (LLMs) [Mitchell et al., 2022a; Dai et al., 2022; Hartvigsen et al., 2023; Cheng et al., 2023; Tan et al., 2024]. For example, as shown in Figure 1, when prompting with “How can I turn screws?”, knowledge editing techniques can focus on specific areas in LLMs for adjustment, changing the answer from “Use a hammer” to “Use a wrench” without compromising the overall performance. Recently, numerous works on knowledge editing for LLMs have been proposed, which can be divided into two main paradigms [Yao et al., 2023]: 1) Preserve Models’ Parameters by utilizing additional parameters or memory [Mitchell et al., 2022b]; 2) Modify Models’ Parameters to alter the weights responsible for the undesirable output [Meng et al., 2022a].

However, previous knowledge editing approaches mainly focus on single-task settings, which means they may fail to achieve multi-task generalization capabilities and demonstrate inefficiency in editing when confronted with Out-of-Distribution (OOD) data. For example, as shown in Figure 1 and Table 1, the knowledge editing approach can simply change the behavior when prompting with “How can I turn screws?”, but fail to generalize to different task when prompting with “How can I drive nails”. Fundamentally, for the Preserve Models’ Parameters paradigm, Additional Parameters methods [Dong et al., 2022; Huang et al., 2023] fit updated data with few extra parameters, while Memory-based approaches [Mitchell et al., 2022b; Hartvigsen et al., 2023], storing only current batch knowledge, can hardly generalize to OOD data. For the Modify Models’
which learns a well-formed Editor by designing the correspond-

Touvron et al. Table 1: Motivating knowledge editing results in multi-task gener-

Figure 1. Specifically, we utilize meta-learning editing meth-

Parameters paradigm, Locate-Then-Edit [Meng et al., 2022a; Meng et al., 2022b] target and directly update specific parame-

Intuitively, devising a strategy to enable the knowledge edit-

2The instructions text in this paper are limited to task descriptions rather than natural language instructions, which is a limitation we leave for future work.

2.1 Knowledge Editing

Recently, knowledge editing [Sinitsin et al., 2020; Zhang et al., 2024] has emerged, aiming for efficient and accurate updates of knowledge in LLMs, to address the issues of outdated knowledge due to their training cut-off, factual fallacy, and potential generation of unsafe content. This technique is applied in various domains [Xu et al., 2022; Mao et al., 2023; Hase et al., 2023; Wang et al., 2023a; Li et al., 2023b; Cheng et al., 2023; Zhong et al., 2023; Akyürek et al., 2023], with an increasing number of researches investigating the impact of knowledge editing [Ilharco et al., 2023; Gupta et al., 2023; Cohen et al., 2023; Wu et al., 2023; Wang et al., 2023b; Gandikota et al., 2023; Brown et al., 2023; Wei et al., 2023; Li et al., 2023d; Li et al., 2023a; Ju and Zhang, 2023; Li et al., 2023c; Onoe et al., 2023; Pinter and Elhadad, 2023; Gupta et al., 2024; Huang et al., 2024; Gu et al., 2024; Lo et al., 2024; Yin et al., 2024; Yu et al., 2024; Ma et al., 2024]. Researchers have diligently classified existing knowledge editing approaches into two main paradigms:

Preserve Models’ Parameters. For those approaches, knowledge can be updated without altering models’ parameters, primarily following two paradigms: Additional Parameters and Memory Based. Additional Parameters integrate extra trainable parameters into the models. These added parameters are trained on a modified knowledge dataset, while the original models parameters remain unchanged. T-Patcher [Hung et al., 2023] embeds a single neuron (patch) for each error in the model’s final Feed-Forward Network (FFN) layer, activating only upon encountering the respective mistake. CallNet [Dong et al., 2022] drawing inspiration from [Dai et al., 2022], introduces additional trainable parameters into the FFNs. Memory Based store edit examples in memory and use a retriever to select relevant edit facts for new inputs, thereby directing the model’s fact generation. SERAC [Mitchell et al., 2022b] presents a method that utilizes a distinct counterfactual model while maintaining the integrity of the original model. GRACE [Hartvigsen et al., 2023] employs a distinct codebook as an adapter, progressively incorporating and refreshing elements to refine the model’s predictions. In-context Knowledge Editing [Zheng et al., 2023] produces outputs aligned with given knowledge using refined in-context prompts.

Modify Models’ Parameters. Those methods edit LLMs by modifying a portion of the parameter θ via applying an Δ matrix. There are primarily two paradigms: Locate-Then-Edit and Meta-learning. Locate-Then-Edit targets and directly updates specific parameters. ROME [Meng et al., 2022a] utilizes causal mediation analysis for targeted editing but is limited to one fact at a time. Addressing this, MEMIT [Meng et al., 2022b] has been proposed, enabling direct memory embedding into the model through rank-one modifications of single-layer MLP weights. Meta-learning utilizes a hyper-net to predict specific Δ for each data point. MEND [Mitchell et al., 2022a] and Knowledge Editor (KE) [Cao et al., 2021] propose strategies that include an external editor, adept at identifying the optimal Δ for knowledge editing, while enforcing constraints to preserve the stability of the model.

<table>
<thead>
<tr>
<th>Unseen</th>
<th>Seen</th>
<th>Reliability</th>
<th>Generalization</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterFact Recent</td>
<td>-25.50</td>
<td>-21.34</td>
<td>-7.33</td>
<td></td>
</tr>
<tr>
<td>CounterFact</td>
<td>ZsRE</td>
<td>-25.26</td>
<td>-18.36</td>
<td>-4.79</td>
</tr>
<tr>
<td>ZsRE Recent</td>
<td>-86.40</td>
<td>-91.33</td>
<td>-0.60</td>
<td></td>
</tr>
<tr>
<td>ZsRE</td>
<td>CounterFact</td>
<td>-56.31</td>
<td>-64.90</td>
<td>-1.35</td>
</tr>
</tbody>
</table>

Table 1: Motivating knowledge editing results in multi-task generalization. Directly transferring to the unseen task (CounterFact and ZsRE) can result in a significant performance decay.
### Task (Dataset) | Instruction
---|---
**CounterFact** | **Task:** CounterFact  
**Description:** A dataset designed to challenge and assess model on...  
**Input:** The official language of...  
**ConvSent** | **Task:** ConvSent  
**Description:** Teach the chatbot to sound [LABEL] about [TOPIC]...  
**Input:** What do you think of...

Table 2: Examples of the instructions. As for ConvSent, we need to replace [LABEL] and [TOPIC] according to the input.

## 2.2 Instruction Tuning

Instruction Tuning [Zhang et al., 2023b] markedly improves models’ capability to handle new and unseen tasks by teaching them to comprehend and follow natural language instructions. In NLP, the focus is rapidly shifting towards refining LLMs [OpenAI, 2022; Sun et al., 2023; Su et al., 2023] to follow natural language instructions for real-world tasks. The effectiveness of these approaches is evident in the enhanced zero-shot and few-shot learning capabilities of these LLMs, demonstrating their improved proficiency in adapting to new tasks with minimal prior exposure. Inspired by the generalization capabilities of Instruction Tuning [Liang et al., 2022; Ouyang et al., 2022], we take the first step to integrate instructions into knowledge editing, endowing one unified Editor with commendable instruction generalization and zero-shot capabilities to concurrently handle multiple editing tasks.

## 3 Background

**Knowledge Editing Task Definition.** Knowledge editing, as described by [Zhang et al., 2024], aims to alter the behavior of an initial base model \(f_\theta\) (where \(\theta\) represents the model’s parameters) in reaction to a specific edit descriptor \((x_i, y_i)\) while maintaining the model’s performance on other samples. The target is to create an edited model \(f_{\theta'}\), which succinctly encapsulates the intended modifications in the model’s performance. Concretely, the model \(f_\theta\) can be represented with a function \(f: \mathcal{X} \rightarrow \mathcal{Y}\) which associates an input \(x\) with its corresponding prediction \(y\). Given an edit descriptor that includes the edit input \(x_i\) and edit label \(y_i\), with the condition that \(f_\theta(x_i) \neq y_i\), the revised model \(f_{\theta'}\) is engineered to yield the anticipated output, ensuring that \(f_{\theta'}(x_i) = y_i\).

**Pilot Experiments.** We conduct pilot experiments to highlight the issue inherent in current knowledge editing methods. Note that the generalization of current knowledge editing methods is primarily focused on discussions about In-Distribution data, where the concept of generalization in the context of original knowledge editing is related to rephrasing sentences from the edited text data [Mitchell et al., 2022a]. Nonetheless, our empirical findings indicate that while existing knowledge editing methods can adapt to rephrased sentences, their generalization ability to OOD data is limited.

## 4 Instruction-Based Knowledge Editing

### 4.1 Instruction Dataset Construction

**Selected Task.** To ensure diversity in multi-task editing, we select a range of datasets: Recent [Zhang et al., 2024] for knowledge insertion, CounterFact [Zhang et al., 2024] for counterfactual generation, and ConvSent [Mitchell et al., 2022b] for sentiment editing in knowledge updating. Recent focusing on triplets added to WikiData after July 2022, is used to enable model updates with the latest knowledge. **CounterFact** emphasizes triplets from top-viewed Wikipedia pages to address the issue of models overlooking less prominent entities in modification edits. **ConvSent** is a sentiment editing task aimed at adjusting a dialog agent’s sentiment on a specific topic, like “What do you think of bananas?” without affecting responses of other topics. The training approach retains the original settings of the ConvSent. Additionally, we use a balanced subset, randomly sampled from the original ConvSent, for multi-task training. Detailed analyses are presented in Figure 4.

**Hold Out Task.** Empirically, we find that transferring knowledge from other tasks to ZsRE is challenging as shown in Table 1. Therefore, we utilize ZsRE, a zero-shot relation extraction dataset, to evaluate the generalization ability of multi-task editing, which means we do not incorporate ZsREin multi-task editing training. Specifically, we use the extended version by [Yao et al., 2023], which adds a portability test and new locality sets to the original dataset.

**Instruction Generation.** We develop instruction templates for multi-task knowledge editing, encompassing four task families, they are: CounterFact, Recent, ConvSent, and ZsRE. Each includes instructions for task-specific model discovery, with input and target templates, and associated metadata. Specifically, we craft tailored instruction sets for each task family, including [Task], [Description], and [Input]. The [Task] represents the specific task linked to a data item, while the [Input]

### Multi-Task Editing Definition.

In this paper, we mainly focus on multi-task editing setting, which means the editing approach should have the ability to handle various multiple tasks. We denote a LLM as \(f\), characterized by its parameters \(\theta\) to form \(f_\theta\). For editing in a single task, we introduce a dataset as \(D_{edit}\). When we extend to multi-tasking scenarios, the dataset becomes a set comprising a collection \(\{D_{edit}^1, D_{edit}^2, ..., D_{edit}^t\}\), with each element representing a unique task. In each specific task \(t_j\), we engage with original input-output knowledge pairs, expressed as \((x_{ij}^t, y_{ij}^t)\sim D_{edit}^t\). The editing objective is to evolve the model’s output from the original erroneous \(y_{ij}^t\) to a more accurate \(\hat{y}_{ij}^t\), achieved by adjusting the model’s parameters from \(f_\theta\) to \(f_{\theta'}\). Formally, the procedure can be described as follows:

\[
\begin{align*}
    f_\theta(x_{ij}^t) = y_{ij}^t \rightarrow f_{\theta'}(x_{ij}^t) = \hat{y}_{ij}^t
\end{align*}
\]
A dataset is a dataset focusing on single instruction construction evaluation dataset focusing on.

Instruction Construction

PROMPT: Geography is a dataset...
Based on this description, please provide 10 generic... Editing

InstructEdit
Pre-Editor

Trial Editor

Multi-Task Editing

What are symptoms of a cold?

In-Distribution Task Editing

Out-of-Distribution Task Editing

HyperNet

Edit Gradient

Rash and Itching.

Coughing and fever.

Figure 2: Assuming access to multi-domain task data: Law, Geography, Medicine, and Math. Single-Task Editing) Original editing is domain-specific (e.g., a Geography Editor edits geography-related knowledge but can’t transfer it to Medicine). Multi-Task Editing) Previous methods (Pre-Editor) trained across domains (Law, Geography, and Math) often misdirect In-Distribution Task Editing. For OOD Task Editing (Medicine), a lack of guidance V leads to missing the correct edit region. Instructions enable precise editing and improve generalization. Instruction Construction) We utilize GPT-4 to generate instructions through well-crafted prompts, evaluate metrics using the Trial Editor, and then employ GPT-4 for continuous Instruction Optimization, enhancing the instructions until there is no further improvement in metrics.

embodies the data item itself. We delve into the specifics with the [Description], which is the essential component that uniquely tailors each task. Leveraging GPT-4 [OpenAI, 2023] and detailed task information, we generate 20 descriptions for each task and manually select 10 candidates based on their clarity and conciseness. Subsequently, we concatenate [Task], [Description], and [Input] to form the instructions presented in Table 2. Notably, while the last instruction is used to evaluate the model’s generalization capabilities with instructions, the others are utilized for training. We further optimize instructions by feeding them with performance metrics into GPT-4 to improve the quality as shown in Figure 2. All instruction data will be released to the community.

4.2 Unified Editor Learning with Instructions

In this section, we primarily focus on the crucial role of instructions in directing the editing process and delve into a detailed explanation of how InstructEdit works. Specifically, we define the instruction set as \{I_1, I_2, ..., I_j\} \sim I, where I_j represents a collection of instructions for task t_j. Based on the instructions, we outline the editing process as follows:

\[
\begin{align*}
 f_{\phi}(I_{ij}, x_i) &= y_{ij} \\
 f_{\phi}(x_i) &= f_{\phi}(x_i) & \text{Otherwise}
\end{align*}
\]  
(2)

where \( I_{ij} \) refers to an instruction randomly selected from \( I_j \), \( E(x_{ij}) \) includes both \( x_{ij} \) and its equivalent expressions. InstructEdit employs the editing architecture of MEND, utilizing a meta-learning editor (hypernetwork) for implementing edits. InstructEdit updates the model’s parameters \( u_{\ell} \in M \) with an editor parameterized by \( \phi \). It does this by mapping \( u'_{\ell} \) (the input to layer \( \ell \) for each batch element \( i \)) and the gradient \( \delta_{\ell+1} \) (calculated as \( \delta_{\ell+1} \leftarrow \nabla_{W_{\ell}} L(x_i, y_i) \)) to pseudoactivations \( \bar{u}_{\ell} \) and pseudodelta \( \bar{\delta}_{\ell+1} \). The knowledge editing gradient for the weight matrix \( u_{\ell} \) is then represented as follows:

\[
\nabla_{u_{\ell}} = \bar{\delta}_{\ell+1} \bar{u}_{\ell}^{\top}.
\]  
(3)

Additionally, we scale the gradient \( \nabla_{u_{\ell}} \) with \( L_2 \) norm of the gradient to isolate its directional component, denoted by \( \nabla_{u_{\ell}} = \bar{u}_{\ell} / ||\nabla_{u_{\ell}}||_2 \). Intuitively, \( \nabla_{u_{\ell}} \) points the key knowledge area for editing elements \( i \). This facilitates a more meaningful comparison across various tasks by focusing solely on the gradient’s direction while discarding its magnitude. We term this focused area as editing area.

Our primary objective is to equip the editor with the ability to comprehend and apply editing instructions, thus enhancing its capability to edit tasks that fall outside the usual distribution. Additionally, we append instructions before the input to facilitate multi-task editing. InstructEdit aims to augment multi-task editing capabilities, seeking a synergistic impact where the collective result surpasses the individual contributions. Through the concatenation of instructions, as shown in Figure 2, InstructEdit aims to cluster task vectors and reduce conflicts between tasks, which guarantees that the performance of the multi-task editor on individual tasks matches or even surpasses that of dedicated single-task editors.
Table 3: Multi-Task Editing Setting. Editors train on a hybrid of CounterFact, Recent, and ConvSent datasets, and test on their specific test sets. Hold Out Editing Setting: The abovementioned editors are tested on ZsRE (OOD data). All metrics are “the higher, the better”.

<table>
<thead>
<tr>
<th>DataSet</th>
<th>Model</th>
<th>Metric</th>
<th>Base</th>
<th>FT-L</th>
<th>CaliNet</th>
<th>KE</th>
<th>MEND</th>
<th>GRACE</th>
<th>InstructEdit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Multi-Task Editing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>0.00</td>
<td>0.40</td>
<td>0.24</td>
<td>33.97</td>
<td>74.26</td>
<td>96.31</td>
<td>80.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>0.00</td>
<td>0.32</td>
<td>0.12</td>
<td>8.70</td>
<td>46.48</td>
<td>53.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>43.73</td>
<td>82.81</td>
<td>90.94</td>
<td>88.68</td>
<td>99.99</td>
<td>67.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>11.00</td>
<td>0.87</td>
<td>3.64</td>
<td>27.41</td>
<td>41.88</td>
<td>11.00</td>
<td>50.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.98</td>
<td>84.15</td>
<td>54.35</td>
<td>84.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>44.10</td>
<td>0.36</td>
<td>50.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>70.66</td>
<td>89.28</td>
<td>90.86</td>
<td>91.18</td>
<td>99.75</td>
<td>88.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>27.04</td>
<td>3.19</td>
<td>26.93</td>
<td>33.43</td>
<td>65.84</td>
<td>27.04</td>
<td>69.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>2.61</td>
<td>6.48</td>
<td>11.53</td>
<td>49.37</td>
<td>85.62</td>
<td>99.68</td>
<td>85.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>1.58</td>
<td>2.21</td>
<td>5.37</td>
<td>10.98</td>
<td>52.76</td>
<td>1.58</td>
<td>51.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>26.58</td>
<td>83.87</td>
<td>87.12</td>
<td>57.94</td>
<td>100.0</td>
<td>64.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>17.19</td>
<td>16.78</td>
<td>10.31</td>
<td>30.41</td>
<td>42.26</td>
<td>76.73</td>
<td>47.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>9.87</td>
<td>6.16</td>
<td>9.79</td>
<td>15.88</td>
<td>82.31</td>
<td>83.72</td>
<td>83.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>7.27</td>
<td>3.87</td>
<td>6.64</td>
<td>0.08</td>
<td>54.66</td>
<td>7.35</td>
<td>55.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>70.66</td>
<td>89.28</td>
<td>88.88</td>
<td>78.57</td>
<td>99.98</td>
<td>87.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>43.52</td>
<td>3.15</td>
<td>43.26</td>
<td>43.52</td>
<td>60.84</td>
<td>44.13</td>
<td>62.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>40.74</td>
<td>7.48</td>
<td>37.47</td>
<td>53.07</td>
<td>54.67</td>
<td>40.74</td>
<td>65.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>100.0</td>
<td>42.86</td>
<td>87.47</td>
<td>94.58</td>
<td>96.58</td>
<td>100.0</td>
<td>94.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>613.13</td>
<td>548.55</td>
<td>396.43</td>
<td>615.61</td>
<td>601.93</td>
<td>414.03</td>
<td>617.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fluency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>0.00</td>
<td>0.11</td>
<td>0.00</td>
<td>13.50</td>
<td>40.79</td>
<td>0.00</td>
<td>82.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>0.00</td>
<td>0.08</td>
<td>0.10</td>
<td>10.13</td>
<td>31.15</td>
<td>0.00</td>
<td>78.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>74.06</td>
<td>95.66</td>
<td>82.59</td>
<td>94.79</td>
<td>100.0</td>
<td>94.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>47.07</td>
<td>0.96</td>
<td>0.39</td>
<td>43.90</td>
<td>45.08</td>
<td>47.07</td>
<td>40.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reliability</td>
<td>0.00</td>
<td>2.23</td>
<td>0.00</td>
<td>2.70</td>
<td>76.95</td>
<td>0.00</td>
<td>76.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generalization</td>
<td>0.00</td>
<td>1.93</td>
<td>0.00</td>
<td>0.19</td>
<td>67.89</td>
<td>0.00</td>
<td>70.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locality</td>
<td>100.0</td>
<td>98.89</td>
<td>99.66</td>
<td>95.15</td>
<td>90.74</td>
<td>100.0</td>
<td>94.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portability</td>
<td>56.66</td>
<td>0.54</td>
<td>0.87</td>
<td>48.02</td>
<td>58.63</td>
<td>56.66</td>
<td>58.19</td>
</tr>
</tbody>
</table>

5 Experiments

5.1 Experimental Settings

Editing Models. We conduct experiments on GPT2-XL(1.5B) [Radford et al., 2019] and LLaMA-2-Base (7B) [Touvron et al., 2023].

Baselines. In this paper, we compare our method with FT-L method, which involves fine-tuning a specific layer’s FFN identified via causal tracing in ROME [Meng et al., 2022a]. We further compare our method with preserve models’ parameters editing baselines including CaliNet and GRACE, and modify models’ parameters editing baselines including MEND and KE.

5.2 Metrics

We apply several evaluation metrics consistent with those described in [Yao et al., 2023].

Reliability. Reliable editing is defined when the post-edit model $f_{\theta'}$ generates the target answer correctly for the case $(x_i, y_i)$. Reliability is assessed based on the average accuracy of the edit case.

$$\mathbb{E}_{x_i', y_i' \sim \{(x_i, y_i)\}} \{ \operatorname{argmax}_{y} f_{\theta'} (y | x_i') = y_i' \}$$ (4)

Generalization. The post-edit model $f_{\theta'}$ should predict the equivalent neighbor $N(x_i, y_i)$, like rephrased sentences, and its performance is assessed by the average accuracy on examples uniformly sampled from this equivalence neighborhood.

$$\mathbb{E}_{x_i', y_i' \sim N(x_i, y_i)} \{ \operatorname{argmax}_{y} f_{\theta'} (y | x_i') = y_i' \}$$ (5)

Locality. Editing should be implemented locally, ensuring that the post-edit model $f_{\theta'}$ preserves the outputs for out-of-scope examples $O(x_i, y_i)$. Therefore, locality is measured by the rate at which $f_{\theta'}$ maintains the same predictions as the pre-edit model $f_{\theta}$.

$$\mathbb{E}_{x_i', y_i' \sim O(x_i, y_i)} \{ f_{\theta'} (y | x_i') = f_{\theta} (y | x_i') \}$$ (6)

Portability. Portability, proposed by [Yao et al., 2023], gauges the edited knowledge application of the post-edit model $f_{\theta'}$. [Yao et al., 2023] adds $P(x_i, y_i)$ to the existing dataset and calculates Portability by the edited model’s average accuracy on these new reasoning parts.

$$\mathbb{E}_{x_i', y_i' \sim P(x_i, y_i)} \{ \operatorname{argmax}_{y} f_{\theta'} (y | x_i') = y_i' \}$$ (7)

Fluency. Fluency measures the edited model $f_{\theta'}$’s generative performance by using a weighted average of bi-gram and trigram entropies to evaluate text diversity. Lower values suggest higher repetition.

---

We follow [Cao et al., 2021; Yao et al., 2023] to evaluate the rephrase-based generalization.
We evaluate the efficacy of InstructEdit by examining three key aspects: Multi-Task Editing, Hold Out Editing, and Transfer to Unseen Instruction.

**Multi-Task Editing Results.** Table 3 presents the corresponding results. FT-L [Yao et al., 2023] exhibit subpar performance in Reliability for multi-task editing, which we believe is due to the interference of the original models’ prior knowledge, complicating the editing process. Moreover, we notice that FT-L does not enhance Portability or Generalization, as expected due to its focus on fitting updated knowledge. Our experiments reveal that FT-L substantially reduces the original model’s parameter knowledge, significantly lowering Locality. Preserve Models’ Parameters Editing Methods like CaliNet [Dong et al., 2022] maintain backbone model integrity, resulting in high Stability, but their performance in other metrics is unsatisfactory. Similar to FT-L, CaliNet overfits updated knowledge, leading to poor Generalization and Portability, but it has better Locality than FT-L as it doesn’t alter the original parameters of the LLMs. While GRACE represents the state-of-the-art of Preserve Models’ Parameters Editing Methods, delivering outstanding Reliability and Locality, it falls short in the metrics of Generalization and Portability. Modify Models’ Parameters Editing Methods, such as KE [Cao et al., 2021] and MEND [Mitchell et al., 2022a], surpass previous editing approaches in effectiveness. Both MEND and KE excel across all metrics, achieving a balance between Reliability and Locality. This is attributed to their optimization objectives that limit update extents, thus enabling editors to adjust parameters while preserving model stability. We can observe our InstructEdit improves editing precision and control with instruction-guided methods, reaching effectiveness akin to advanced hypernets like MEND and KE. While MEND and KE yield effective editing results, their performance is suboptimal on OOD data, with editing in In-Distribution data often causing misdirection in the update trajectory of the posterior vector space. However, we find that providing specific command hints to the Editor can substantially alleviate this issue.

**Hold Out Editing Results.** To evaluate the adaptability of knowledge editing methods to OOD data, we devise the “Hold Out Editing Setting”. In this setup, the editor is trained using datasets like Recent, CounterFact, and ConvSent, and then evaluated on ZsRE. From Table 3, we notice a linear decline in the performance of all previous knowledge editing baselines when applied to OOD data. This decline can be attributed primarily to the editor’s limitations in defining new editing tasks and its insufficient generalization capability for handling OOD scenarios. We observe that InstructEdit can effectively address these challenges. Note that such robust generalization abilities are mainly inherent in instruction tuning, a synergy that enables InstructEdit to attain performance levels on par with single-task editing, even on task datasets that are unseen during the training phase.

**Transfer to Unseen Instructions.** To delve deeper into the generalizability of Instruction-based Editing, we evaluate InstructEdit’s capacity with instructions that have not been encountered previously. This setting is different from the hold-out editing setting since we still use the data in CounterFact, Recent, ConvSent, and ZsRE, but with new instructions. Specifically, as outlined in Section 4.2, we construct five novel, unseen instructions to assess the Editor’s proficiency in generalizing instructions. Observations from Figure 3 reveal that the Editor is indeed capable of adapting to these Unseen Instructions. It is noteworthy that utilizing instructions on which the Editor has been trained can result in enhanced editing performance. Thus, InstructEdit can achieve comparable outcomes by employing instructions that are semantically akin to those encountered during training. These empirical results also indicate that we can develop an Editor to follow human instructions and we leave this for future works.

**5.4 Why Instruction Helps Multi-Task Editing?** We analyze the principal components of the editing area in Figure 3, generated using t-SNE, as presented in Section 4.2, which is generated by the editor for specific layers of LLMs. Our underlying assumption is that these principal components encapsulate the intrinsic characteristics of the editing area involved in editing the data. Specifically, we focus our analysis on cases where the conventional editing methods fall short, while InstructEdit demonstrates effectiveness.

**Finding 1: Instruction can Help Control Optimization Direction.** As observed in Table 2, MEND exhibits subpar performance in multi-task scenarios, particularly in terms of Reliability and Generalization, where it significantly outperformed by InstructEdit. Upon analyzing the left panel in Figure 4(a), we observe that MEND, when handling multi-task editing, tends to show significant overlap in editing area across different tasks. This overlap could potentially cause MEND to not only confuse previously learned tasks but also struggle in effectively generalizing to new tasks with shifted distribution compared to the training tasks. However, by introducing instructions, InstructEdit can effectively control the knowledge editing gradient and encourage distinct separation with adequate margin in the editing area for various tasks, which aligns with the distribution observed in the single-task training setting.

<table>
<thead>
<tr>
<th>InstructEdit</th>
<th>Recent</th>
<th>CounterFact</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reliability</strong></td>
<td>85.7</td>
<td>85.4</td>
</tr>
<tr>
<td><strong>Generalization</strong></td>
<td>51.6</td>
<td>49.9</td>
</tr>
<tr>
<td><strong>Locality</strong></td>
<td>64.6</td>
<td>68.2</td>
</tr>
<tr>
<td><strong>Portability</strong></td>
<td>47.3</td>
<td>50.8</td>
</tr>
</tbody>
</table>

Figure 3: InstructEdit demonstrates proficiency in generalizing to Unseen instructions (unseen instructions introduced in Section 4.2), achieving results comparable to Seen instructions.
in the right panel of Figure 4(a). Furthermore, the discriminative editing area in InstructEdit is adaptable to OOD data, which leads to superior knowledge editing when handling new tasks, while maintaining performance comparable to models trained on single tasks on ID tasks.

**Finding 2: More Tasks, Stronger OOD Generalization.** Figure 4(b) illustrates that when InstructEdit is trained on a single task, the editing areas of the three tasks appear somewhat discriminative. Instead, the performance of the corresponding tasks is suboptimal, as demonstrated in Figure 4(c). We think that even though instructions aid in distinguishing different tasks, the knowledge learned from a single task struggles to generalize to others. By scaling the number of tasks in training, we notice that the editing areas of InstructEdit for various tasks almost see no overlap, and editing reliability improves correspondingly in Figure 4(c). Furthermore, as the scope of tasks broadens, the directions of knowledge editing gradient of different tasks start to converge, yet they retain their relative margin. Intuitively, InstructEdit trained across diverse domains harnesses these domain-related instructions to extrapolate effectively to new, unseen domains, while offering a trade-off between specialized knowledge adaptation and broad generalization. Nevertheless, it is crucial to acknowledge that a scalability bottleneck might be encountered, and confronting entirely new types of editing tasks, such as cross-linguistic tasks, will introduce further complexities.

**Finding 3: Improving Performance with Appropriate Data Proportion.** In preliminary experiments, we notice task imbalances impede proper multi-task training and cause a significant performance decline when ConvSent is involved in the training. Hence, we contemplate balancing the data proportions across different tasks. By observing Figure 4(d), we find that the knowledge editing gradient directions become more regular after data balancing and editing reliability of the editor increases from 18.23 to 25.55 on the OOD tasks. Additionally, we find that task imbalances lead to the editor inducing editing gradients of relatively large magnitudes, and the gradient magnitude distributions for each task vary significantly, which appears to be a key factor influencing the editor’s generalization. This result confirms the significance of appropriate data proportions for multi-task editing.

### 6 Discussion and Conclusion

We focus on a new problem of knowledge editing for LLMs: generalizing to new tasks. We introduce multi-task editing, illustrating the limitations of existing knowledge editing approaches in task transferability and presenting a viable solution InstructEdit which can effectively guide the Editor for precise editing, with its effectiveness confirmed through comprehensive experiments and visualization analysis.

**Acknowledgments**

Ningyu Zhang, Bozhong Tian, Siyuan Cheng, and Xiaozhuan Liang equally contribute. Xi Chen and Huajun Chen are the corresponding authors. We want to express gratitude to the anonymous reviewers, EasyEdit [Wang et al., 2023b], and many other related works. This work was supported by the National Natural Science Foundation of China (No. 62206246), the Fundamental Research Funds for the Central Universities (226-2023-00138), Zhejiang Provincial Natural Science Foundation of China (No. LGG22F030011), Yongjiang Talent Introduction Programme (2021A-156-G), CCF-Tencent Rhino-Bird Open Research Fund, and Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.
References


[Cheng et al., 2024] Siyuan Cheng, Ningyu Zhang, Bozhong Tian, Xi Chen, Qingbing Liu, and Huajun Chen. Editing language model-based knowledge graph embeddings. In AAAI, 2024.


[Dong et al., 2022] Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. Calibrating factual knowledge in pretrained language models. In EMNLP, 2022.


[Gu et al., 2022] Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. Model editing can hurt general abilities of large language models, 2024.


[Gupta et al., 2024] Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. Model editing at scale leads to gradual and catastrophic forgetting, 2024.


[Huang et al., 2024] Youcheng Huang, Wenqiang Lei, Zheng Zhang, Jiancheng Lv, and Shuicheng Yan. See the unseen: Better context-consistent knowledge-editing by noises, 2024.


[Li et al., 2023b] Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. Pnet: Precise model editing in a transformer. 2023.


[Sun et al., 2023] Tianxiang Sun, Xiaotian Zhang, Zhengfu He, Peng Li, Qin yuan Cheng, Hang Yan, Xiangyang Liu, Yunfan Shao, Qiong Tang, Xingjian Zhao, Ke Chen, Yining Zheng, Zhejian Zhou, Ruixiao Li, Jun Zhan, Yunhua Zhou, Linyang Li, Xiaogui Yang, Lingling Wu, Zhangyue Yin, Xuanjing Huang, and Xipeng Qi. Moss: Training conversational language models from synthetic data. 2023.
[Yu et al., 2024] Lang Yu, Qin Chen, Jie Zhou, and Liang He. MELO: enhancing model editing with neuron-indexed dynamic lora. In AAAI, 2024.