A Divide and Conquer Framework for Knowledge Editing

Xiaoqi Han[a], Ru Li[a,b,*], Xiaoli Li[c] and Jeff Z. Pan[d,*]

[a] *School of Computer & Information Technology, Shanxi University, China,*
[b] *Key Laboratory of Computational Intelligence and Chinese Information Processing of Ministry of Education, Shanxi University, China,*
[c] *Institute for Infocomm Research, A*Star, Singapore,*
[d] *School of Informatics, The University of Edinburgh, United Kingdoms,*

ARTICLE INFO

Keywords:
Pre-trained Language Model
Knowledge Editing
Dynamantic interfence

ABSTRACT

As Pre-trained language models (LMs) play an important role in various Natural Language Processing (NLP) tasks, it is becoming increasingly important to make sure the knowledge learned from LMs is valid and correct. Unlike conventional knowledge bases, LMs *implicitly* memorize knowledge in their parameters, which makes it harder to correct if some knowledge is incorrectly inferred or obsolete. The task of Knowledge Editing is to correct errors in language models, avoiding the expensive overhead associated with retraining the model from scratch. While existing methods have shown some promising results, *they fail on multi-edits as they ignore the conflicts between these edits*.

In the paper, we propose a novel framework to divide-and-conquer edits with parallel Editors. Specifically, we design explicit and implicit multi-editor models to learn diverse editing strategies in terms of dynamic structure and dynamic parameters respectively, which allows solving the conflict data in an efficient end-to-end manner.

Our main findings are: (i) State of the art Knowledge Editing methods with multiple editing capability, such as MEND and ENN, can hardly outperform the fine-tuning method; (ii) Our proposed models outperform the fine-tuning method over the two widely used datasets for Knowledge Editing; (iii) Additional analytical experiments verify that our approach can learn diverse editing strategies, thus better adapting to multiple editing than state-of-the-art methods.

1. Introduction

As pre-trained language models are increasingly applied in a variety of downstream NLP tasks, such as Web search[1], question answering [81], and dialogue [44], it is becoming increasingly important to make sure these models are learning valid knowledge and generating the correct outputs. However, the knowledge LMs learned (called parametric knowledge) can be outdated [40], incorrect [48], toxic [31] or biased [10] because they are pre-trained on a static, limited historical data. Furthermore, retraining a model on huge updated data is expensive, especially with the ever-increasing size of current language models [12]. One of the ideas to correct LMs' incorrect knowledge without expensive retraining is **Knowledge Editing**.

Nevertheless, unlike conventional knowledge bases, LMs *implicitly* memorize facts in their parameters. As such, updating specific knowledge in LMs is not straightforward due to the highly nonlinear nature of deep neural networks (DNN) and how different parameters determine the outputs of language models [69, 22, 8, 80]. Changing a single weight in DNN may have a ripple effect that affects many other implicitly memorized facts. To overcome this, a reliable and computationally efficient editing model should possess three properties [20]: 1) **Generality**, able to modify a model that was not specifically trained to be editable, 2) **Reliability**, able to successfully update a specific fact without affecting the rest of the acquired knowledge, and 3) **Consistency**, the changes should be consistent across equivalent formulations of a fact.

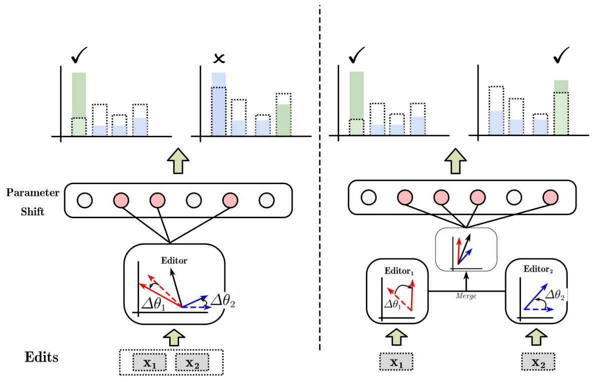

Figure 1: The green bars show the predicted results for the data we want to modify, and the dashed boxes show the results before modification. When editing data x_1 (shown as the red dashed line) and x_2 (shown as the blue dashed line) with conflicting gradients, the existing method (Left) will use same strategies (since x_2 is dominated by x_1, the editor corrects the gradient direction to the left.) and fail to edit x_2 with eventually parameter shift. Our method (Right) can process the two data separately (adaptive to provide the appropriate modification direction for both) and get a more suitable way to change them. The black arrow is the final update direction.

Recent research has made initial progress by designing a hyper-network [35] to predict updates to the weights of the LMs [20, 53]. Especially, gradient-based methods focus

*Corresponding authors
xiaoqisev@163.com (X. Han); liru@sxu.edu.cn (R. Li); xlli@i2r.a-star.edu.sg (X. Li); j.z.pan@ed.ac.uk (J.Z. Pan)
ORCID(s): 0000-0002-8827-8474 (X. Han)

[1]https://blog.google/products/search/search-language-understanding-bert/

on *single editing*, i.e., designing hyper networks to predict the parameter shift of LMs on single editing based on the standard fine-tuning gradient of a given edit. However, these research do not perform well when editing *multiple edits*, they use the same strategy to process multiple edits and ignore the relation between different edit gradients, resulting in a "zero-sum" phenomenon, where the inter-gradient conflict will inevitably cause some data modifications to fail. Figure 1 demonstrates how an editor can be used to modify model outputs. As shown in the left panel, existing methods use the same editor to simultaneously modify two data points x_1 and x_2. However, due to conflicting gradients between the two data points, the methods produce inaccurate parameter shifts, resulting in the model only being able to modify x_1. Our method, on the other hand, uses different editors to mitigate conflicts between the edits and successfully modify both conflicting edits.

In this paper, we focus on multiple editing in LMs, especially simultaneous editing. We assume that handling multiple edits by a *diversity editor* can break the existing dilemma and reduce conflicts between data. We propose a divide-and-conquer framework, drawing on dynamic inference to break the zero-sum phenomenon in multiple edits. We designed two novel models, including the dynamic structure model MoEditor and the dynamic parameter model Proactive Editor (or simply ProEditor), respectively. For MoEditor, we first group the modified data, then explicitly parallel editors and let each editor learn diverse modification strategies by editing different data dynamically. Finally, we propose a selection-based fusion approach to fusing the results of multiple editors. However, the data grouping and the fusion method significantly impact the performance of MoEditor. Furthermore, the multi-expert structure causes an increase in the number of parameters. Therefore, to optimize MoEditor, we propose ProEditor, where we adopt an intrinsic hyperparametric network to generate editor weights, implicitly constructing a multi-editor model.

We verify the effectiveness of the proposed framework on closed book fact-checking (FEVER) [76] and question answering (ZsRE) [46] datasets. The experimental results demonstrate that both approaches are superior to the baselines, and the ProEditor performs better in most metrics. Interestingly, MoEditor can prove that individual editors learn diverse editing strategies. Overall, our contributions are summarised as follows:

- We propose a divide and conquer framework to integrate dynamic computation into the knowledge editing task, alleviating the *zero-sum phenomenon* caused by the conflicts between data in multiple edits.
- We propose a novel MoEditor model by adapting the Mixture of Experts to edit tasking for the first time, as well as using multiple editors to learn diverse editing strategies. Further, we propose a Proactive Editor to avoid the problems of parameter explosion and high parameter sensitivity in MoEditor.
- We extensively evaluate our methods on binary classification and question answering tasks, confirming their advantages over state-of-the-art models. Furthermore, we prove that multiple editors can learn diverse editing strategies.

The remaining structure of this paper is as follows: In Section (2), we provide an overview of the related work on knowledge editing and the work on utilizing knowledge to enhance language models, and briefly introduce the dynamic networks. Section (3) presents the task formulation and provides background for knowledge editing. Section (4) describes our proposed divide and conquer framework. Section (5) overviews the experimental details and results. We first introduce the datasets and parameter settings, followed by a description of the advantages of our method in multiple editing tasks. We then analyze the experimental results under different conditions to demonstrate the effectiveness of our proposed approach. Finally, we discuss the future work in Section (6).

2. Related Work

Since 2000, structured knowledge, such as knowledge graphs have been widely used in many different fields, which, however, is quite different from parametric knowledge in language models that is the main focus on this paper. In this section, we first briefly look into knowledge updates for structured knowledge (Section 2.1), as well as the arrival of parametric knowledge (Section 2.2). We then discuss some key related techniques for updating parametric knowledge (Sections 2.3 and 2.4) .

2.1. Knowledge Updates for Structured Knowledge

Recently, the use of knowledge graphs [64, 58] has become popular in knowledge representation and knowledge management applications widely across search [34, 63, 57, 33], recommendation [84, 87, 90], medical informatics [86, 78, 94, 93], finance [23, 18, 97], science [89, 4, 25, 59, 42], media [71, 60, 1, 50], software engineering [67, 38, 61, 65, 88, 3] and industrial domains [64, 5, 13]. However, although there have been a lot of works on various uncertainty aspects of knowledge graphs [62, 75, 68, 72, 11], due to the incompleteness or inaccuracies of knowledge graphs, a series of methods have been proposed to complement or update them, such as link prediction[55, 41, 17, 16], knowledge graph completion[32, 6, 91, 83, 14]. These methods extract structured knowledge from unstructured data to achieve updates and completion of knowledge in knowledge graphs. In this paper, we focus on updating parametric knowledge, specifically updating the knowledge that the language models incorrectly learned or missed out, to enhance the robustness and reliability of the language models.

2.2. Knowledge in language models

Language Models (LMs) as Knowledge Bases (KBs) in NLP is increasingly popularly because they store a large amount of knowledge in their parameters (referred to as parametric knowledge) and they are capable of answering natural language queries about real-world facts[66, 70].

Dai et al. [19] showed that factual knowledge is stored in the neurons of the feed-forward network in transformers, with specific knowledge corresponding to specific knowledge neurons. AlKhamissi et al. [2] conducted a comprehensive analysis of the capabilities which enable LMs to function as KBs, offered an overview of both LMs and KBs while exploring their intersection, and pointed out that Editability as an ability to ensure the accuracy and relevance of LMs, is one critical challenge that needs to be addressed when using LMs as KBs.

Apart from editing, other methods are also employed to modify the knowledge in LMs, such as knowledge-enhanced language modelling [95, 28, 82], knowledge adaptation [27, 29], and knowledge distillation [49, 45]. These methods are typically utilized in the pre-training or fine-tuning phase to enable LMs to acquire various forms of knowledge. However, these methods still learn some erroneous knowledge, which may lead to catastrophic forgetting if models continues training on erroneous knowledge. In contrast, knowledge editing focuses on the usage stage of LMs (after fine-tuning) and is more concerned with correcting incorrect knowledge in LMs without affect the unmodified data.

2.3. Knowledge Editing for Language Models

Several existing works have been proposed for Knowledge Editing. A natural strategy is to retrain and fine-tune the model based on the modified data. However, this approach is costly, and retraining cannot guarantee that erroneous data will be corrected. Zhu et al. [96] constructed a collection of supporting evidence for the modified facts and used it to fine-tune the model by minimizing a per-instance loss. Though it achieves high performance on the modified data, its performance on unmodified data is degraded significantly. Then, Zhu et al. [96] used a L_p norm-based method to constrain the weights of the language model and minimally interfere with the data that should not be modified. However, the highly non-linear nature of LMs makes such methods underperform in Reliability and Consistency [20].

Another class of methods develop hyper-network to learn a parameter shift for the base model. De Cao et al. [20] trained a hyper-network, KnowledgeEditor, to modify a fact and used Kullback-Leibler (KL) divergence-constrained optimization to alleviate the side effect on other data/knowledge that should not be changed. Mitchell et al. [53] pointed out KnowledgeEditor cannot edit on the very large LMs, and proposed a hyper-network with Gradient Decomposition, making editing tasks feasible on larger models. Hase et al. [37] focused on sequential editing, and proposed SLAG, an editor based on KnowledgeEditor. These methods have shown effectiveness in *single editing*.

Hyper Networks can view as a "probe" implicitly revealing the parameters associated with modifying the data. There also exists works explicitly locating parameters related to modifying a specific fact [52]. In addition, , Sinitsin et al. [74] used a MAML algorithm to search a set of model parameters that provided good performance for edits. Mitchell et al. [54] proposed a semi-parametric model editor that stored model edits in an external memory rather than directly in model parameters. However, since multiple edits are not a simple repetition of a single edit, the performance drops sharply with the number of edited increase. In this paper, we propose an editing model based on hyper-network for *multiple editing* tasks, achieving simultaneous editing in a divide-and-conquer way.

2.4. Dynamic Networks

Recently, the dynamic network has attracted great research interests, with the key idea being adaptive inference. Existing methods are typically designed from two perspectives: Dynamic architectures and Dynamic parameters. [36]

Dynamic architectures [56, 85, 15] are designed to perform inference on each input sample, considering the varying computational demands of different inputs. They save redundant computation for canonical samples ("Easy for Models"), while preserving the representation power necessary to recognize non-canonical samples ("Hard for Models")[2]

On the other hand, Dynamic parameters [30, 51, 9] can adapt network parameters to various inputs while keeping the architectures fixed, thereby enhancing the networks' representational capacity with only a minor increase in computational cost. Compared to dynamic architectures, they eliminate the need for designing specific architectures and training strategies, and careful hyper-parameter tuning.

The mixture of experts (MoE) [39], a classical algorithm for the dynamic network, has been widely used in NLP. For instance, Shazeer et al. [73] embedded MoE into Long Short-Term Memory (LSTM) network and proposed the sparely gated MoE. Fedus et al. [26] proposed Switch Transformer, which uses MoE to implement a sparse activation model, ensuring that the computational cost remains unchanged while allowing the network to have a large number of parameters. PATHWAYS [7] is a typical application of the MoE method, allowing the model to dynamically learn which network parts perform well at which tasks.

In this paper, we focus on the more practical batch editing problem, trying to edit the parametric knowledge encoded within LMs. In particular, we adapt the MoE idea to multiple editing tasks and propose the novel MoEditor to achieve divide-and-conquer editing. Furthermore, we propose a dynamic parameter model, ProEditor, to enhance the capabilities of MoEditor by providing a more robust and effective mechanism for managing the model parameters.

3. Preliminaries

3.1. Task Formulation

Assuming we have a pre-trained LM $f(\cdot;\theta)$ fine-tuned on a dataset $\langle x, y, a\rangle \in D$, and it maps the input x to the

[2]In this paper, we can consider the data that fails to be modified due to conflicts between them when modifying multiple pieces of data as "non-canonical samples", while the data that is successfully modified is "Canonical samples".

output y under a set of parameters θ. The task of knowledge editing is to make $f(\cdot;\theta)$ prefer the output of the input x to be an alternative prediction a instead of y. To evaluate the properties of the editing model, we have constructed three types of dataset $\langle x_i, y_i, a_i\rangle \in D_i, i \in \{u, o, p\}$ for knowledge editing. See Table 1 for specific examples.

- To achieve *generality* and *reliability*, the dataset D is divided into D_u and D_o . The subset D_u contains the edits, and the subset D_o contains the unmodified data. Notice, we only need to ensure that the output y_o remains the same for the input x_o in D_o after editing, a_o is useless in D_o.
- To test the consistency of Editor, we generated a semantically consistent input dataset D_p for the data in D_u.[3].

In the rest of the paper, we use x_u, x_p, x_o to represent the modified data, semantically consistent data, and other data, resp. We use N_u to represent the number of modified data, and N_d as the size of D_u, use N_p to represent the number of edits sample from D_p, in this paper, we set $N_u = N_p$, similar to MEND, for each edit, we select one semantically consistent input for evaluation.

As mentioned in Sec.2.3, the gradient-based method is a feasible way to apply the editing. The gradient is a property of neural networks and is an important mathematical signal that used to train neural networks, and gradients carry rich information about how LMs access the knowledge stored in θ [53, 20]. The research on gradient-based editing methods is helpful for us to explore the training methods of LMs and deepen our understanding of LMs mechanism. Therefore, our work primarily focuses on gradient-based methods.

More formally, denote a set of parameters for different layers in LMs as $\theta = \{w_1, w_2, ..., w_L\}$, L is the number of layers in LMs. We calculate the standard fine-tuning gradient $\nabla W = \{\nabla w_1, \nabla w_2, ..., \nabla w_L\}$ of a given correction a_u and the model output $f(x_u;\theta)$:

$$\nabla W = \frac{\partial L(f(x_u;\theta), a_u)}{\partial \theta}, \tag{1}$$

where $L(\cdot)$ is the loss function (e.g. The Cross Entropy Loss). Then, the edit model $\phi(\cdot)$ can view as a mapping process that maps the gradient ∇W to a suitable parameter shift $\theta^\star = \{\theta_1^\star, \theta_2^\star, ..., \theta_L^\star\}$, denote as $\nabla W \xrightarrow{\phi(\cdot)} \theta^\star$. Here, we introduce two commonly used methods for editing: Fine-tuning and Hyper-Networks.

3.2. Editing by Fine-tuning

Fine-tuning (FT) is a popular method of training models that involves initializing the model with pre-trained weights and updating all parameters (or only a small portion) through optimization functions such as Gradient Descent. The update rule is typically represented as $\theta^\star = \alpha\nabla W$, where α is the learning rate and ∇W is the gradient. However, since ∇W contains a lot of redundant information, which can lead to **catastrophic forgetting** when solving editing problems. In order to mitigate the Catastrophic forgetting problems, Zhu et al. [96], using explicit constraints on the LMs weights:

$$\begin{aligned} \text{minimize}_{\theta'} \frac{1}{N_d} \sum_{<x,y,a>\in D_u} L(f(x_u;\theta'), a_u) \\ \textit{subject to} \parallel \theta' - \theta \parallel < \delta, \end{aligned} \tag{2}$$

where δ is a small positive constant, N_d is the size of D_u, $\parallel \cdot \parallel$ denotes a L_p norm in the parameter space (e.g. L_2 norm).

However, De Cao et al. [21] points out that such constraints are insufficient. The L_p constraint only makes parameter changes as sparse as possible in the parameter space. However, due to the implicit of LMs, even a minimal change in parameters may produce completely different outputs for many data points, leading to catastrophic forgetting. Therefore, alternative approaches are being considered. (Sec.3.3)

3.3. Editing with Hyper-Networks

Hyper-Network, as described in Ha et al. [35], involves utilizing one network to generate the weights for another network. Researchers such as De Cao et al. [21] and Mitchell et al. [53] have recently employed hyper-network as a mapping function to edit language models. It allows for precise and flexible control of model parameters, enabling accurate identification of the necessary parameter shift.

As shown in Figure 2, for a frozen LM $f(\cdot;\theta)$ with L layers, the hyper-network methods training a light-weight model G (Editor) with weight θ_g to produce edits to LM's weights θ when provided with the standard fine-tuning gradient (Eq. (1)) of a given correction as input. Significantly, when modifying multiple layers, we need to train different editors $G_l, l \in [1, L]$ for each layer. Finally, by applying the output (parameter shift) θ_l^* of each editor to the corresponding modified layer, we can simultaneously edit while keeping the results of other data x_o unchanged.

MEND[53] is one way to realise knowledge editing with hyper-networks. We adopt the baseline model in this paper due to its efficiency and ability to edit a few data simultaneously.

The architecture of MEND is shown in Figure 3. The key of MEND is the Gradient Decomposition, which decomposes each gradient of layer l into its rank-one outer product form (cf. Eq. (5)), and uses the Editor $G_l(\cdot;\theta_g)$ to map the decomposing gradient into a new decomposition form that can modify the edits (cf. Eq. (6)), and finally calculate the parameter shift $\theta_l^\star$ for layer l (cf. Eq. (7)):

$$\theta_l^\star = G_l(x_u;\theta_g). \tag{3}$$

The detail for MEND is as follows, after calculating the gradient for x_u, the Editor $G_l(\cdot;\theta_g)$ decomposed the gradient for each layer $\nabla w_l \in \mathbb{R}^{hid_1 * hid_2}$ into $\mu_l \in \mathbb{R}^{d_1 * hid_1}$ and

[3]Semantically consistent input means that the sentences that have the same meaning as the edits but are expressed differently. e.g. For the modification: "Who is the UK PM? ", one possible semantically consistent input is: "Who is the prime minister of the UK?".

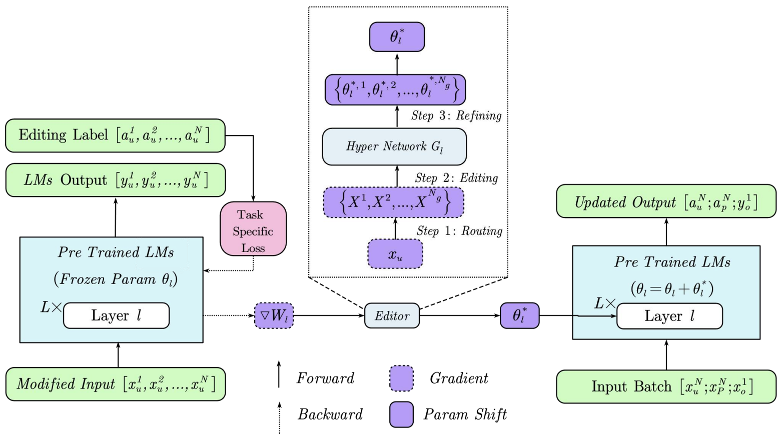
Editing Label $[a_u^1, a_u^2, ..., a_u^N]$
LMs Output $[y_u^1, y_u^2, ..., y_u^N]$
Task Specific Loss
Pre Trained LMs (Frozen Param θ_l)
$L\times$ Layer l
Modified Input $[x_u^1, x_u^2, ..., x_u^N]$
∇W_l
Editor
θ_l^*
$\{\theta_l^{*,1}, \theta_l^{*,2}, ..., \theta_l^{*,N_g}\}$
Step 3: Refining
Hyper Network G_l
Step 2: Editing
$\{X^1, X^2, ..., X^{N_g}\}$
Step 1: Routing
x_u
Updated Output $[a_u^N; a_p^N; y_o^1]$
Pre Trained LMs $(\theta_l = \theta_l + \theta_l^*)$
Input Batch $[x_u^N; x_P^N; x_o^1]$
Forward
Backward
Gradient
Param Shift

Figure 2: The editing process with Hyper Network. Given an LMs consisting of L layers, after calculating the fine-tuning gradients of a given edit batch with a task-specific loss function(e.g., Cross Entropy Loss). The Editor maps the gradient ∇W_l of layer l to the parameter shift $\theta^\star$. With the application of $\theta^\star$, LMs can effectively modify the results of the edits x_u^N and x_P^N while maintaining the output of data x_o^1 .

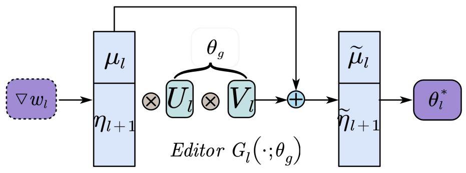

Figure 3: The baseline Editor $G_l(\cdot;\theta_g)$ architecture with parameters θ_g.

$\eta_{l+1} \in \mathbb{R}^{d_1 * hid_2}$, d_1 is the numbers of tokens for edits x_u, hid_1 and hid_2 is the hidden dimension of LMs $f(\cdot;\theta)$.

The decomposed is based on the chain rule, we define the inputs to layer l as μ_l and the pre-activation inputs to layer l+1 as $z_{l+1} = \theta_l \mu_l$, θ_l is the weight matrix to layer l, and η_{l+1} as the gradient of the loss L with respect to z_{l+1}. Eq. (5) shows that the gradient of the loss L with respect to θ_l is equal to $\eta_{l+1}\mu_l^T$ in matrix representation.

$$\frac{\partial L}{\partial \theta_l^{ij}} = \sum_k \frac{\partial L}{\partial z_{l+1}^k} \frac{\partial z_{l+1}^k}{\partial \theta_l^{ij}} = \frac{\partial L}{\partial z_{l+1}^i} \frac{\partial z_{l+1}^i}{\partial \theta_l^{ij}}, \tag{4}$$

where θ_l^{ij} represents the value in the i-th row and j-th column of the weight matrix, and z_{l+1}^k represents the pre-activation inputs in the k-th row. The first equality means the derivative of the loss L with respect to weight θ_l^{ij} is equal to the product of the derivative of loss L with respect to next-layer pre-activations z_{l+1}^i and the derivative of next-layer pre-activations z_{l+1}^i with respect to θ_l. The second equality is due to the pre-activation inputs of the k-th row z_{l+1}^k is only related to $\theta_l^{k\cdot}$, so when $k \neq i$ is satisfied, $\frac{\partial z_{l+1}^k}{\partial \theta_l^{ij}} = 0$. Noting that $z_{l+1}^i = \theta_l \mu_l = \sum_j \mu_l^j \theta_l^{ij}$, simply we can replace $\frac{\partial z_{l+1}^i}{\partial \theta_l^{ij}}$ with μ_l^j. Futhermore, we define η_{l+1} to be exactly $\frac{\partial z_{l+1}^i}{\partial \theta_l^{ij}}$. Then we have:

$$\frac{\partial L}{\partial \theta_l^{ij}} = \eta_{l+1}^i \mu_l^j, \tag{5}$$

in vector notation, we denote Eq. (5) as $\nabla w_l L = \eta_{l+1}\mu_l^T$.

Then we concat the μ_l and η_{l+1} as $[\mu_l; \eta_{l+1}] \in \mathbb{R}^{d_1 * d_2}$, and use two *learnable* low-rank metrics $U_l \in \mathbb{R}^{d_2 * ind}$ and $V_l \in \mathbb{R}^{ind * d_2}$ to mapping the $[\mu_l; \eta_{l+1}]$ to $[\tilde{\mu}_l; \tilde{\eta_{l+1}}]$:

$$[\tilde{\mu}_l; \tilde{\eta_{l+1}}] = [\mu_l; \eta_{l+1}] + ([\mu_l; \eta_{l+1}] * U_l * V_l) \tag{6}$$

where $d_2 = hid_1 + hid_2$,$ind = min(intr_{dim}, d_2)$, $intr_{dim}$ is a hyper-parameter, in this paper $intr_{dim}$ is 1920. Finally, the parameter shift on layer l is calculated by $[\tilde{\mu}_l; \tilde{\eta_{l+1}}]$:

$$\theta_l^\star = \tilde{\eta_{l+1}} \tilde{\mu}_l^T. \tag{7}$$

In comparison to FT, MEND employs two low-rank matrices to selectively preserve relevant information for edits (as shown in Eq. (6)), thereby minimizing interference from redundant information. However, when editing multiple modifications, MEND uses the same U_l and V_l to map the different edits at layer l (cf. Fig. (3)), ignoring the conflict between edits which could lead to editing failures.

We conclude this session by comparing FT and hyper-network based knowledge editing as follows:

1. The hyper-network based solutions help mitigate the issue of Catastrophic forgetting of FT;
2. MEND is affected by the conflict between data. Finding a suitable parameter change value for all edits at the same time is difficult, resulting in modification failure;
3. In this paper, our proposed method decomposes the complex multi-objective optimization problem into sub-problems through the divide and conquer strategy. It reduces the conflict between modified data by finding the diversity editing strategy and combining them to achieve better multiple edits.

4. The Proposed Approach

Motivated by the the work of Dynamic Neural Networks [36], we propose a novel divide-and-conquer framework to achieve simultaneous editing and reduce the probability of data conflicts. The edits modified by a single editor will be divided into multiple groups, and we use different editors to modify each data group separately. In order to achieve the goal of divide-and-conquer editing, we use three stages corresponding to the three steps within the editor shown in Figure 2 respectively:

1. The stage of data routing (Sec 4.1) aims to group the edits and decide which editor handles which edit(s).
2. The stage of editing (Sec 4.2) shows how to edit the input data with implicit and explicit editors.
3. The stage of Refining (Sec 4.3) determines how to refining the editor output $\theta^\star$ to better apply to the LMs.

In this section, we present the details of each of the three stages.

4.1. Data routing

The stage of data routing divides the edits x_u into N_g groups, with each data group being modified by the same editor. There are many ways to realize data routing.

A straightforward way is to randomly group edits. However, our experimental results show that it is not stable enough. In order to address this issue, we can train a network to determine which edits should be put in the same group. Although argmax(Softmax(·)) can help in theory, it is not derivable due to the discontinuity of argmax(), which leads to its inability to be trained. Instead, we adopt the Gated Network, which is widely used in Mixture of Experts [73], due to its efficiency. Specifically, we use a noisy top gating model Gate(·) [73] defined as follows:

$$\begin{aligned}\text{Gate}(\cdot) &= \text{Softmax}(f(x)\cdot W + \text{Noise}),\\ \text{Softplus} &= \lambda \cdot \text{Softplus}(f(x)\cdot W_{noise}),\end{aligned} \tag{8}$$

where x is the edits, $f(x)$ is the encoder of x by LMs, λ is a normal distribution, $W \in \mathbb{R}^{d\times N_g}$ and $W_{noise} \in \mathbb{R}^{d\times N_g}$ denote as the Gate network weight matrix and noise weight matrix, and d is the hidden dimension of the LMs. W and W_{noise} makes the model *trainable*. Note that W_{noise} improves the gated model's generalization and makes the grouping result more stable. The output of $Gate(\cdot)$ is the distribution of group index, the j-th group is $X^j = \{x^i\}_{i=1}^N$, when $\text{argmax}(\text{Gate}(x^i)) = j$.

4.2. Editing

The stage of editing determines how to use the divide-and-conquer framework to resolve the conflict of data, thereby achieving simultaneous editing. We designed two models as the Editor G_l.

MoEditor. Inspired by Mixture of Experts [39], we propose our first model Mixture of Editors (MoEditor), treating multiple editors as different experts explicitly. The architecture of MoEditor is shown in Figure 4. The MoEditor can be regarded as initializing N_g MENDs (cf. Eq. (3)) in parallel for each edited layer of LMs, the editors at layer l denote as $G_l(\cdot;\theta_g^i), i \in [1, N_g]$. The j-th editor's output at layer l is:

$$\theta_l^{\star,j} = G_l(X^j;\theta_g^j) = \eta_{\tilde{l+1}}^{j}\,\tilde{\mu}_l^{j,T}. \tag{9}$$

where X^j is the $j\text{-}th$ group of edits, $\{U_l^j, V_l^j\} \in \theta_g^j$ is the parameters of $j\text{-}th$ editor G_l, each G_l is calculated by Eq. (6). After editing, a gather function merges each editor's output (Sec.4.3).

With the MoEditor, the parameter shifts $\theta_l^{\star,j}$ are calculated by different groups of U and V, thus not only addressing the data conflict issue of MEND but also helping avoid the catastrophic forgetting caused by FT due to ∇W possibly containing a lot of information that is not needed to modify the current edits.

However, MoEditor has some limitations. Firstly, it requires extensive memory. For an LM that requires modification of L layers, MoEditor requires $L \times N_g$ editors, while MEND only requires L editors. Secondly, it is sensitive to parameters. The number of groups can influence the stability of the results. Finally, fusing multiple editors' results might cause a secondary conflict, leading to performance loss.

ProEditor. To solve these problems, we propose a dynamic parameter model, Proactive Editor (or simply ProEditor), which is an implicit editor model that uses only one editor to approximate multiple editors by a generator editor's parameters. Compared with the MoEditor, edits for the Proactive Editor are not to choose an editor but to construct it for greater adaptability. This implicit strategy can reduce the multiplication of memory space caused by the explicit method of divide-and-conquer and make the relationship between the editor and the target modified data more flexible, enabling the editor to modify different data adaptively.

Specifically, we use an intrinsic hyper network ψ to generate weights in the ProEditor for each group of edits (cf. Fig. (5)). We use the feed-forward neural Network(FFN) as the structure of ψ and take the decomposed gradient $[\mu_l; \eta_{l+1}]$ as the FFN input.

$$\theta_l^\star = G_l(x_u; \psi(x_u)), \tag{10}$$

where $\psi(x_u)$ is the parameter generator of the editor G_l. Notable, the ProEditor does not care about the group number N_g because it can generate parameters for each edit or token.

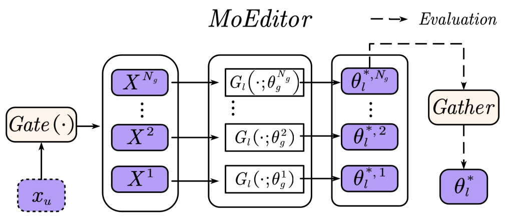

Figure 4: The MoEditor architecture.

As we can see, the ProEditor only needs an additional FFN to achieve divide and conquer, making it more efficient than MoEditor.

Naturally, Because $[\mu_l;\eta_{l+1}]$ is the gradient decomposition representation of each token, there is two granularity to generate parameters: token and sentence levels. In downstream applications, users can decide which option works better. In general, the sentence level allows to have even fewer parameters.

Following, we refine Eq. (6) under these two settings. As shown in Figure 5, at the token level, $\psi(\cdot)$ generates parameters for each token; the mapping process is:

$$[\tilde{\mu_l};\tilde{\eta_{l+1}}] = \delta + (\delta \cdot \psi(x_u) * U_l * V_l), \quad (11)$$

$$\psi(x_u) = \text{FFN}(\delta) = [P_1, P_2, ..., P_{d_1}],$$

where $\cdot$ is Hadamard product, $\psi(x_u) \in \mathbb{R}^{d_1 * d_2}$ and we denote $\delta \in \mathbb{R}^{d_1 * d_2}$ as $[\mu_l;\eta_{l+1}]$, $P_i \in \mathbb{R}^{d_2}$ is the generated parameters. At the sentence level, tokens under the same sentence in the same group will share common parameters:

$$[\tilde{\mu_l};\tilde{\eta_{l+1}}] = \delta + ([\delta_1, \delta_2, ..., \delta_{N_g}] \cdot \psi(x_u) * U_l * V_l), \quad (12)$$

$$\psi(x_u) = \text{FFN}([\delta_1, \delta_2, ..., \delta_{N_g}]) = [P_1, P_2, ..., P_{N_g}],$$

where $\psi(X^j) \in \mathbb{R}^{N_g * d_2}$, and $[\delta_1, \delta_2, ..., \delta_{N_g}]$ is the decomposition gradient for N_g groups. The finally parameter shift $\theta_l^\star$ at layer l is calculated by Eq. (7).

4.3. Refining

The refining stage determines how to apply the editor's outputs to the LMs during inference (only).

For the MoEditor, as the gradient is a property of neural networks, it is an important mathematical signal to judge the importance of the parameters. The larger the value is, the more important the parameters are. We prioritize parameters with larger absolute values and propose a selection-based fusion method to combine the results of multiple editors.

The selection-based fusion method is defined as follows:

$$\varphi(x) = \text{MAX}(\rho(\widehat{\text{Pos}(x)}), \rho(\widehat{\text{Neg}(x)})), \quad (13)$$

where $\rho(x)$ can be the sum or average function of the inputs, $\hat{x}$ means the absolute value of x, Pos(x) and Neg(x) are the sets of positive and negative values, respectively. The final output for MoEditor is:

$$\theta_l^\star = \varphi(\{\theta_l^{\star,j}\}_{j=1}^{N_g}), \quad (14)$$

where φ is the selection-based fusion function; $\theta_l^{\star,j}$ is the output of the j-th editor at layer l.

For the ProEditor, we find that not all parameters in the final output need to be updated, so we use the Fisher Information [79] to filter out some unimportant parameters to minimize the modification of parameters in LMs. The Fisher information estimates how much information a variable carries about a distribution parameter. In other words, we use Fisher information to select the important parameters in each set of modifications and update them only for this part. Formally, we derive the Fisher information for the parameter $w \in \theta_l^\star$ as follows:

$$F(w) = \frac{1}{N}\sum_{i=1}^{N}(\frac{\partial logp(a_u^i|x_u^i;w)}{\partial w})^2, \quad (15)$$

where N is the number of edits and w is the edited parameters, x_u^i, a_u^i are the i-th edits input and the alter label. $p(\cdot)$ is the output of LMs for the edits. The final output for Proactive Editor $\theta_l^\star$ is: $\theta_l^\star * Sign(F(w) > \tau)$ where $Sign(\cdot)$ is the the symbolic function, $\theta_l^\star$ is the parameter shift of layer l, τ is the threshold, and we modify the parameter w once $F(w) > \tau$. Finally, the parameter shift for LMs is: $\theta^\star = \{\theta_1^\star, \theta_2^\star, ..., \theta_L^\star\}$.

4.4. Training objective

To maintain Consistency and Generality, we calculate the cross-entropy loss of the edits. We sample N_u edit pairs $< x_u, y_u, a_u >$ from D_u, and extract N_p $< x_p, y_p, a_p >$ pairs associated with x_u from D_p which is semantically equivalent inputs with edits in D_u.

Then we use $< x_u, y_u, a_u >$ to calculate the parameter shift $\theta^\star$, and use $< x_p, y_p, a_p >$ to calculate the edit loss.

$$L_{edit} = -\text{logP}(a_p|f(x_p;\theta+\theta^\star)), \quad (16)$$

where logp$(\cdot)$ is the cross-entropy loss function.

To maintain Reliability, we use Kullback-Leibler (KL) divergence from the updated model to the original one. We sample $< x_o, y_o >$ from D_o, and the KL pushes the updated model to predict output distributions identical to the original one:

$$L_{rel} = \text{KL}(P(f(x_o;\theta)||P(f(x_o;\theta+\theta^\star)), \quad (17)$$

where $P(f(x;\theta))$ is the output of Language model with parameters θ for the input x. The KL divergence helps us optimize parameters by quantifying the differences between representations and enables us to avoid uncertainties arising from regularization constraints.

It is worth noting that for MoEditor, we optimize the results of each Editor separately during training[4]. The total training loss is defined as:

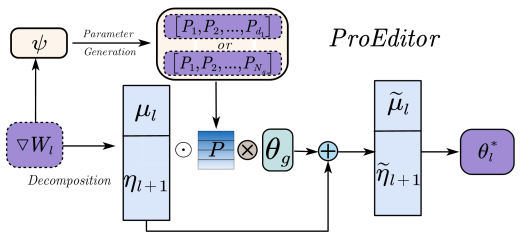

Figure 5: The Proactive Editor(ProEditor) architecture, ∇W_l is the gradient for x_u which is calculated by Eq.(1).

[4] We tried to merge them during training, but the results were poor.

Table 1
Examples of the FEVER and ZsRE. We denote *Output* as the output of LMs, and the *Target* is the prediction we prefer.

Dataset	*Type*	*Input*	*Output*	*Target*
FEVER	x_o	IMDb is not a website.	REFUTES	-
	x_u	Apple Inc. marketed the IPhone 4.	REFUTES	SUPPORTS
	x_p	Apple Inc. sold the IPhone 4.	REFUTES	SUPPORTS
ZsRE	x_o	The continent of Kirkwood Islands is what?	Antarctica	-
	x_u	When was Ronaldo Pompeu born?	21 December 1981	8 April 1990
	x_p	What time was Ronaldo Pompeu born?	21 December 1981	8 April 1990

$$\text{Loss} = \sum_{i=1}^{N_g} L_{edit}^i + \sum_{i=1}^{N_g} L_{rel}^i, \tag{18}$$

where N_g is the number of Editors, and we use Adam optimizer [43] to optimize the Loss. For ProEditor, we jointly optimize ψ and the Editor. Since there is only one Editor, the loss function is Loss $= L_{edit} + L_{rel}$.

The use of a single $\theta^\star$ in Eq. (16) to edit all modified data can lead to poor generalization of the editing method. Our method, described by Eq. (18), decomposes the optimization problem into N_g groups. This transformation of a complex problem into multiple simple problems allows for better implementation of multiple edits.

5. Experiment

5.1. Benchmark Datasets and Evaluation.

We evaluate our models on the FEVER fact-checking dataset [77] and the ZsRE question-answering dataset [46]. See Table 1 for the example of two datasets.

Fact-checking benchmark dataset (FEVER) contains 104,966 training instances and 10,444 validation instances respectively. Each instance is a True/False factual claim. The input $x_u \in D_u$ is a batch of facts. $x_p \in D_p$ includes the semantically equivalent inputs generated by De Cao et al. [20]. Locality inputs $x_o \in D_o$ are randomly sampled facts distinct from the edit example.

Question-answering dataset (ZsRE) contains 244,173 training and 27,644 validation instances respectively. Every input $x_u \in D_u$ is a question about an entity, and y_u is sampled from the top-ranked predictions based on a BART-base model trained on QA. $x_p \in D_p$ is generated by De Cao et al. [20]. $x_o \in D_o$ are randomly sampled facts distinct from the edit example.

Evaluation metrics. We measure the *consistency* of a model editor using edit success (ES) [53].

$$\text{ES} = \mathbb{E}_{(x_p,y_p)\sim D_p}[\mathbb{1}_{f(x_p;\theta+\theta^\star)=a_p}]. \tag{19}$$

To assess *reliability*, we use drawdown (DD) [53][5], which is defined as the performance degradation of the edited model on the rest of the dataset:

$$\text{DD} = \mathbb{E}_{(x_o,y_o)\sim D_o}[\mathbb{1}_{f(x_o;\theta+\theta^\star)=f(x_o;\theta)}] \tag{20}$$

Finally, we employ the Language Model Performance (LMP) of the two aforementioned scores.

$$\text{LMP} = \text{ES} - \text{DD} \tag{21}$$

Specifically, when testing model performance, we randomly select N_u edits from D_u and D_p for testing and repeat the process N_d times (N_d is the size of D_u), and the mean of the results is taken as the final result.

5.2. Implementations

We use the MEND with sharing as the Basic Editor, and the fine-tuning Bert-base [24] model for the FEVER fact-checking and BART-base [47] model for ZsRE question-answering come from the checkpoints released by De Cao et al. [20]. Both Editors are trained and evaluated on applying N_u edits, with $N_u \in \{25, 75, 125\}$, with a single extra example to compute drawdown, regardless of the number of edits. Following Mitchell et al. [53], we edit the MLP weight matrices in the last two transformer blocks of the encoder and decoder for BART and the last three transformer blocks for BERT.

For all algorithms, we use early stopping to end training early if the loss Eq.(18) does not decrease for 20000 steps on a subset of 500 validation examples, with a maximum number of training steps of 500,000. The learning rate for ProEditor is 1e-4. We use the same hyper-parameter and experimental settings as MEND[53] for other methods. All runs are trained entirely on a single NVIDIA RTX A100 GPU.

5.2.1. State-of-the art models

We compare our models with the following four most relevant editors that have multi-editing capabilities:

- **MEND** [53] training a hyper-network to learn a rank-1 decomposition of the gradient, which can edit multiple edits simultaneously.
- **FT** [96] fine-tuning the LMs on the edits.
- **FT+KL** [53] enforcing a KL constraint on the outputs of LMs when fine-tuning the LMs on the edits.

[5]Following KnowledgeEditor, We do not compute the drawdown metrics on all data, which would be very computationally demanding.

Table 2
Simultaneous editing results on ZsRE with N_u=$\{25, 75, 125\}$. For MoEditor, the N_g corresponding to each N_u is $\{3, 5, 5\}$.

	25			75			125		
	ES↑	DD↓	LMP↑	ES↑	DD↓	LMP↑	ES↑	DD↓	LMP↑
MEND	0.890	0.011	0.879	0.780	0.011	0.769	0.670	0.012	0.658
ENN	0.350	0.005	0.345	0.160	0.005	0.155	0.110	0.006	0.104
FT	0.923	0.025	0.898	0.908	0.072	0.836	0.893	0.093	0.800
FT+KL	0.639	0.002	0.637	0.393	0.005	0.388	0.330	0.010	0.320
RaEditor	0.917	0.014	0.903	0.852	0.021	0.831	0.789	0.029	0.759
MoEditor	0.945	0.021	0.924	0.856	0.041	0.815	0.794	0.045	0.749
ProEditor	0.961	0.009	**0.950**	0.857	0.014	**0.843**	0.832	0.016	**0.816**

Table 3
Simultaneous editing results on FEVER with N_u=$\{25, 75, 125\}$. For MoEditor, the N_g corresponding to each N_u is $\{3, 7, 7\}$.

	25			75			125		
	ES↑	DD↓	LMP↑	ES↑	DD↓	LMP↑	ES↑	DD↓	LMP↑
MEND	0.890	0.017	0.873	0.560	0.030	0.530	0.553	0.001	0.551
ENN	0.590	0.020	0.570	0.540	0.020	0.520	0.651	0.002	0.649
FT	0.906	0.038	0.868	0.931	0.230	0.701	0.910	0.192	0.709
FT+KL	0.873	0.004	0.869	0.680	0.008	0.672	0.643	0.003	0.641
RaEditor	0.802	0.165	0.638	0.802	0.077	0.725	0.638	0.012	0.626
MoEditor	0.913	0.031	0.882	0.804	0.059	0.745	0.779	0.056	0.723
ProEditor	0.955	0.069	**0.886**	0.907	0.056	**0.851**	0.843	0.064	**0.778**

- **ENN** [74] using gradient descent to update the model's predictions for edits based on the MAML algorithm.

To compare with the state-of-the-art models mentioned above, we use the random group model **RaEditor** and the dynamic structural model **MoEditor** with the functions $Gate()$ and $\rho(x)$ that aggregate the outputs of each expert. Additionally, we use the dynamic parametric model **ProEditor** with the $\tau = 0.7$ in Fisher information.

5.3. Experimental Results

5.3.1. Simultaneous Editing results

This section introduces the simultaneous editing results on two standard benchmark datasets.

Table 2 and Table 3 show the multiple edit results of different methods. The FT outperforms the other baseline methods in ES and LMP, but its high DD results in poor overall performance (i.e. LMP) compared to our two models. This is due to severe overfitting on modified data, which causes catastrophic performance degradation on unmodified data. Adding KL helps reduce DD, but the success rate (ES) significantly reduces.

As the number of edits increases, the performance of the editor deteriorates to varying degrees, which aligns with the common understanding that the more edits, the more conflicts between them, making successful editing more difficult. Moreover, methods like MEND and ENN show more significant performance degradation, which can be attributed to their neglect of the relationship between edits during simultaneous editing which results in modifications being made in the same way for all edits which further leads to the modification result being affected by the data with a large gradient value and different update direction compared to other data.

Our proposed models exhibit significant performance improvements compared to the baselines across all three edit number settings. This suggests that our models can dynamically handle conflicting data, and reduce their impact on each other. However, the RaEditor and MoEditor exhibit higher performance than the baseline only in some cases, indicating their potential instability. In contrast, the ProEditor outperforms both the RaEditor and MoEditor, demonstrating that it can address the issues of parameter sensitivity and secondary conflicts in the MoEditor.

5.3.2. Editing the large-scale LMs

In this section, we aim to demonstrate the effectiveness of our editing method on the large-scale publicly-available Transformer model, T5-XL, which has 2.8 billion parameters and is fine-tuned on the NQ dataset [70]. We evaluate our approach on the ZsRE dataset and present our results in Table 4. Our method outperforms MEND on T5-XL, while other methods like FT face memory constraints and cannot modify 25 edits simultaneously. Interestingly, both MEND and ProEditor only modify up to 25 edits at a time, highlighting the limitations of gradient-based editing methods when it comes to large-scale models with a high number of modifications.

Table 4
Simultaneous editing T5-xl (2.8B) results on ZsRE with N_u=25.

	ES↑	DD↓	LMP↑
MEND	0.648	0.004	0.644
ProEditor	0.705	0.002	0.703

5.4. Editing analysis

In this section, we examine the issue of data conflicts that can arise during modifications, and demonstrate how our proposed diversity strategy effectively resolves this issue.

5.4.1. Data conflict analysis

To measure the conflict between edits, we consider both the angle and amplitude of the data gradient. Specifically, we calculate the cosine value between the gradients of a group of edits to determine the degree of conflicts between them. If the cosine value is less than 0, we identify a directional conflict between two pieces of data. The Gradient Conflict Ratio (GCR) for the group is the ratio of the number of conflicting data pairs to the total number of data pairs D_p in the group, denoted as $D_p = <g_0, g_1>_1, <g_0, g_2>_2, ..., <g*, g*>_C$.

$$\text{GCR} = \frac{\sum_{m=1}^{C} Sign(-cos(<g_i, g_j>_m))}{C}, \tag{22}$$

where $C = \frac{N!}{2!(N-2)!}$ means taking any two combinations of data from N data. $Sign(\cdot)$ is the the symbolic function, and $cos(<g_i, g_j>)$ is the cosine of g_i and g_j. When the gradient direction conflicts, the cosine value is less than 0, we take its opposite value as the conflict fraction.

The gradient magnitude similarity (GMS) [92] measures the magnitude of the gradient values.

$$\text{GMS} = \frac{\sum_{m=1}^{C} \gamma(<g_i, g_j>)}{C}, \tag{23}$$

where $\gamma(<g_i, g_j>) = \frac{2||g_i||_2||g_j||_2}{||g_i||_2^2+||g_j||_2^2} \in [0, 1]$; the larger the γ, the closer the gradient values are.

We analyze data conflicts by counting the conflicting values in each layer, as shown in Table 5. On both datasets, the *GCR* values are around 40%, indicating that there is around 40% conflict of the data, while the *GMS* values are above 70%, indicating that the values of the gradient for most of the edits are closer. In other words, for each of the two data sets, each data has its characteristics. Using the same way to modify these data simultaneously will ignore the unique characteristics and thus inevitably lead to modification failures. To overcome this issue, we dynamically group the data, allowing editors to learn diverse editing strategies that consider the differences between different data as much as possible.

5.4.2. Partition analysis

This section presents our analysis of how to address the problem of data conflict using the divide-and-conquer approach. We let different editors learn diverse editing strategies and employ parameter sparsity to combine the results of multiple editors. This approach aims to mitigate the performance degradation caused by conflicts, improving the edited data's overall quality.

Table 5
The conflict rate and gradient amplitude similarity between edits. We edit the four feed-forward neural network (FFN) layers (L1-L8) in each of the last two Transformer encoder and decoder layers of the Bart model, as well as the three FFN layers (L1-L6) in the last three Transformer layers of the Bert model. L1-L8 denotes these layers in Bart and L1-L6 in Bert.

	75-FEVER		125-FEVER		75-ZsRE		125-ZsRE	
layer	GCR	GMS	GCR	GMS	GCR	GMS	GCR	GMS
L1	0.42	0.74	0.42	0.77	0.47	0.83	0.46	0.85
L2	0.43	0.76	0.40	0.77	0.49	0.83	0.49	0.85
L3	0.38	0.78	0.37	0.79	0.37	0.77	0.41	0.80
L4	0.38	0.81	0.37	0.80	0.48	0.85	0.49	0.86
L5	0.36	0.81	0.37	0.81	0.47	0.86	0.47	0.87
L6	0.34	0.82	0.32	0.80	0.44	0.74	0.46	0.76
L7	-	-	-	-	0.3	0.91	0.27	0.92
L8	-	-	-	-	0.39	0.85	0.38	0.86

Table 6 reveals that each sub-editor E_i achieves great editing results on their respective group of N_u records (We use random grouping and ensure that each sub-editor modified different data.), suggesting that diverse editing strategies are being learned. However, merging the outputs of the sub-editors (cf. Eq. (14)) leads to a slight decrease in overall performance, which becomes more pronounced as N_u increases, indicating the presence of secondary conflicts during merging. Nevertheless, the merged output still outperforms the baseline MEND model. In contrast, ProEditor leverages dynamic parameters to prevent secondary conflicts arising from merging and delivers better performance, demonstrating the effectiveness of the divide-and-conquer approach.

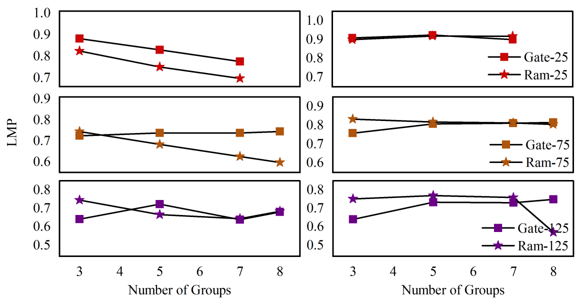

Figure 6: The results of different grouping numbers with MoEditor, Left: Result on FEVER, Right: Result on ZsRE.

5.5. Further Analysis

5.5.1. Model parameter analysis

In this section, we analyze the parameters of the two divide-and-conquer models and obtain the optimal number of groups in MoEditor as well as the modification granularity in ProEditor through comparative experiments.

Table 6
Performance Analysis using Divide-and-Conquer Approach. We employ five sub-editors (E1-E5) on N_u edits. Each sub-editor E_i modified $N_u/5$ edits, and we ensure they edit different data. We denote the output merged by five sub-editors as *RaEditor*, and denote the results of MEND as *MEND*. Both RaEditor and MEND modify N_u edits.

				ZsRE				
N_u	E1	E2	E3	E4	E5	*RaEditor*	*MEND*	*ProEditor*
25	0.960	0.960	0.960	0.957	0.964	0.918	0.879	0.950
75	0.883	0.901	0.894	0.895	0.891	0.813	0.769	0.843
125	0.872	0.866	0.867	0.866	0.871	0.770	0.658	0.816
				FEVER				
25	0.959	0.956	0.961	0.955	0.963	0.620	0.873	0.886
75	0.844	0.831	0.846	0.829	0.815	0.637	0.530	0.851
125	0.826	0.851	0.828	0.822	0.843	0.517	0.551	0.778

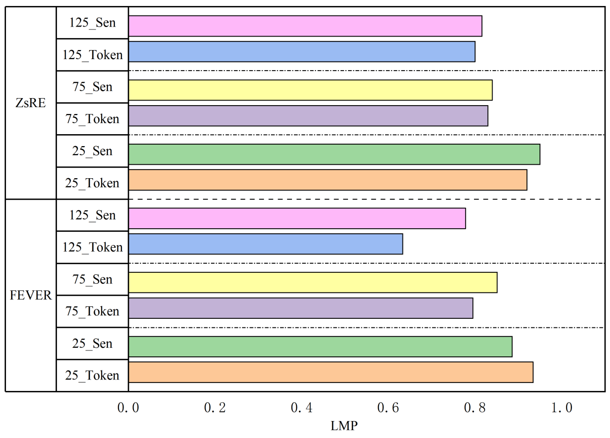

Figure 7: Results at different levels using ProEditor.

Figure 6 illustrates the results of our comparative experiments on MoEditor, where we aim to determine the optimal number of sub-editors. Surprisingly, the performance does not necessarily improve as the number of sub-editors increases. Instead, we observe that three to five sub-editors performed best on the two tasks we have tested. Moreover, having more sub-editors will require more computational resources and double the computation needed, which motivates us to propose the ProEditor with dynamic parameter selection.

For ProEditor, the results presented in Figure 7 indicate that sentence-level granularity performs better than token-level in most cases. This is because the number of tokens increases significantly with the number of sentences, which can lead to many parameters in the model that are difficult to train effectively. On the other hand, sentence-level granularity allows the model to generate specific parameters for each sentence, effectively designing the most suitable modification strategy for each sentence.

Finally, we compare the number of parameters required by different Editors. As shown in Table 7, ProEditor requires similar parameters as MEND, and both require fewer parameters than FT. As far as efficiency is concerned, MoEditor and FT require more time, while ProEditor and MEND have similar time requirements, taking approximately 12 hours and one day on a single A100 GPU for 25 and 75 modifications, respectively.

Table 7
Parameter Analysis. h_1 and h_2 are the dimensions of the edited parameter matrix (Both h_1 and h_2 are larger than 10^2), and N_g is the number of groups in ProEditor output (the maximum for N_g is modification number N_u)

Editor	Editor Parameters
FT	$O((h_1 \times h_2)^2)$
MEND	$O((h_1 + h_2)^2)$
ProEditor	$O((h_1 + h_2)^2 + N_g(h_1 + h_2))$

5.5.2. *Impact of Refining starge*

In this section, we analyze different merging strategies in the Refining stage. In the MoEditor model, editors learn various modification patterns. To reduce secondary conflicts caused by merging multiple results, we conduct experiments to test four different fusion methods:

- Add: Sum the outputs of multiple editors and directly update the sum into the language model to apply the modification.
- Max: Select the maximum absolute value of each element in each output matrix as the final output while preserving its positive or negative sign, then add them as the final result.
- Mean of Sign sum (SSM): Use Eq. (13) to calculate the final result, with the $\rho = \frac{\sum_{i=1}^{N_g} x_i}{N_g}$.
- Sum of Sign sum (SSS): Use Eq. (13) to calculate the final result with the $\rho = \sum_{i=1}^{N_g} x_i$.

Figure 8 illustrates that only Max or SSM fusion methods yield undesirable results. Our analysis suggests that both methods result in a final output that is too one-sided, causing the loss of some information. Conversely, both the Add and SSS methods add up the results, while SSS sums up the values with the same sign, thus retaining the modified information to the maximum extent and achieving superior performance.

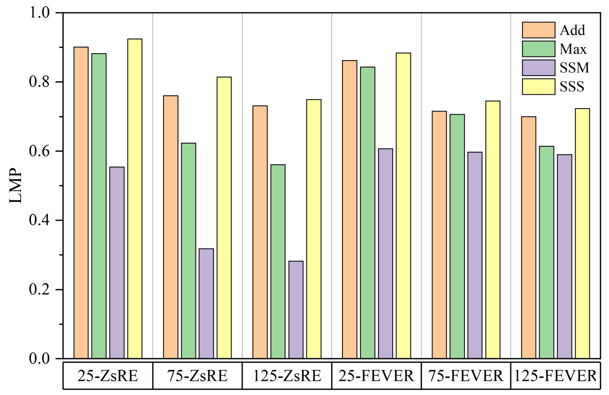

Figure 8: Impact of fusion methods with MoEditor. We test on both FEVER and ZsRE datasets.

Inspired by the fusion results on MoEditor, we investigate the necessity of the parameters in the output of ProEditor during editing. We assume that some parameter shifts may be meaningless or detrimental to the editing performance. To investigate this, we conduct experiments using MEND to test the performance changes under different Fisher thresholds τ in Eq. (15) on two datasets under 25 data conditions, as shown in Figure 9. The results indicate that a threshold of 0.4 achieves performance close to using all parameters, suggesting that we can achieve better editing performance by using only 40% of the parameters. In particular, the blue node in Figure 9 indicates that when 90% of the parameters are used, the result is better than using all the parameters, which suggests that some parameters in the shift $\theta^\star$ are harmful to performance.

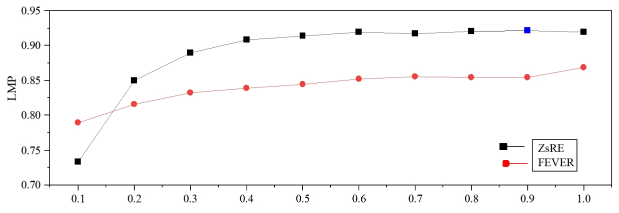

Figure 9: The performance change of MEND under different Fisher thresholds. 0.1 indicates that only parameters with a Fisher value of the top 10% are used, and 1.0 indicates that all parameters are used.

6. Conclusions & Future work

This paper proposes a novel divide-and-conquer framework for simultaneous knowledge editing to address data conflicts through dynamic computation. Our framework consists of a dynamic architecture model and a dynamic parameter model, and they learn diverse modification strategies for different editors. We demonstrate that diverse editing strategies exist for the same data, and our divide-and-conquer method can effectively learn the optimal combination of these strategies. Extensive experiments conducted on two benchmark datasets under various modifying conditions show that our proposed method outperforms four state-of-the-art models by a significant margin.

Our approach has the advantage of addressing data conflicts with dynamic inference. In the future, there are three potential directions to extend our work. Firstly, although our divide-and-conquer framework has improved the editing limit of the baseline editor (MEND), the limit still exists, such as editing on a large number of edits and the more large-scale LMs. A stronger and alternative baseline or a more robust divide-and-conquer algorithm is the key to breaking through the limit. Secondly, Gradient-based editing methods still have some limitations: (i) the requirement for model open-sourcing to obtain the gradient information of parameters and (ii) the need for stronger computing power to run a large number of modifications on large-scale language models. These two factors make it challenging to use gradient-based editing methods for large-scale models with over 10 billion parameters (e.g. T5-XXL) or not open-sourced models (e.g. GPT3 or GPT3.5) to perform large-scale knowledge editing tasks. Finally, we are interested in model interpretability, further explaining the internal mechanism of the language model through knowledge editing and feeding back to the editorial model.

Acknowledgements

This work has been supported by the National Natural Science Foundation of China (No.61936012), by the National Key Research and Development Program of China (No.2020AAA0106100), by the National Natural Science Foundation of China (No.62076155), and by the Chang Jiang Scholars Program (J2019032).

References

[1] Abu-Salih, B., Al-Tawil, M., Aljarah, I., Faris, H., Wongthongtham, P., 2021. Relational learning analysis of social politics using knowledge graph embedding. Data Mining and Knowledge Discovery , 1497–1536.

[2] AlKhamissi, B., Li, M., Celikyilmaz, A., Diab, M., Ghazvininejad, M., 2022. A review on language models as knowledge bases. arXiv preprint arXiv:2204.06031 .

[3] Althar, R.R., Samanta, D., 2021. The realist approach for evaluation of computational intelligence in software engineering. Innov. Syst. Softw. Eng. 17, 17–27.

[4] Auer, S., Kovtun, V., Prinz, M., Kasprzik, A., Stocker, M., Vidal, M.E., 2018. Towards a Knowledge Graph for Science, in: Proc. of the 8th International Conference on Web Intelligence, Mining and Semantics (WIMS 2018), pp. 1327–1328.

[5] Bader, S.R., Grangel-González, I., Nanjappa, P., Vidal, M.E., Maleshkova, M., 2020. A Knowledge Graph for Industry 4.0, in: Proceedings of the 17th Extended Semantic Web Conference (ESWC 2020), pp. 465–480.

[6] Balažević, I., Allen, C., Hospedales, T., 2019. Tucker: Tensor factorization for knowledge graph completion, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 5185–5194.

[7] Barham, P., Chowdhery, A., Dean, J., Ghemawat, S., Hand, S., Hurt, D., Isard, M., Lim, H., Pang, R., Roy, S., et al., 2022. Pathways: Asynchronous distributed dataflow for ml. Proceedings of Machine Learning and Systems 4, 430–449.

[8] Belinkov, Y., Glass, J., 2019. Analysis methods in neural language processing: A survey. Transactions of the Association for Computational Linguistics 7, 49–72.

[9] Bello, I., . Lambdanetworks: Modeling long-range interactions without attention, in: International Conference on Learning Representations.

[10] Bordia, S., Bowman, S., 2019. Identifying and reducing gender bias in word-level language models, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop, pp. 7–15.

[11] Botha, L., Meyer, T., Peñaloza, R., 2021. The probabilistic description logic. Theory and Practice of Logic Programming 21, 404 – 427.

[12] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020. Language models are few-shot learners. Advances in neural information processing systems 33, 1877–1901.

[13] Buchgeher, G., Gabauer, D., Gil, J.M., Ehrlinger, L., 2021. Knowledge graphs in manufacturing and production: A systematic literature review. IEEE Access 9.

[14] Cai, B., Xiang, Y., Gao, L., Zhang, H., Li, Y., Li, J., 2022. Temporal knowledge graph completion: a survey. arXiv preprint arXiv:2201.08236 .

[15] Cai, S., Shu, Y., Wang, W., 2021. Dynamic routing networks, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 3588–3597.

[16] Chen, J., Wang, X., Xu, X., 2022. Gc-lstm: graph convolution embedded lstm for dynamic network link prediction. Applied Intelligence 52, 7513–7528.

[17] Chen, M., Zhang, W., Zhang, W., Chen, Q., Chen, H., 2019. Meta relational learning for few-shot link prediction in knowledge graphs, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 4217–4226.

[18] Cheng, D., Yang, F., Wang, X., Zhang, Y., Zhang, L., 2020. Knowledge graph-based event embedding framework for financial quantitative investments., in: SIGIR, pp. 2221–2230.

[19] Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., Wei, F., 2022. Knowledge neurons in pretrained transformers, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 8493–8502.

[20] De Cao, N., Aziz, W., Titov, I., 2021a. Editing factual knowledge in language models, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic. pp. 6491–6506. URL: https://aclanthology.org/2021.emnlp-main.522, doi:10.18653/v1/2021.emnlp-main.522.

[21] De Cao, N., Aziz, W., Titov, I., 2021b. Editing factual knowledge in language models, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6491–6506.

[22] De Cao, N., Schlichtkrull, M.S., Aziz, W., Titov, I., 2020. How do decisions emerge across layers in neural models? interpretation with differentiable masking, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 3243–3255.

[23] Deng, S., Zhang, N., Zhang, W., Chen, J., Pan, J.Z., Chen, H., 2019. Knowledge-Driven Stock Trend Prediction and Explanation via Temporal Convolutional Network, in: Proc. of the World Wide Web Conference (WWW 2019), pp. 678–685.

[24] Devlin, J., Chang, M.W., Lee, K., Toutanova, K., 2019. Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186.

[25] Edelstein, E., Pan, J.Z., Soares, R., Wyner, A., 2020. Knowledge-driven intelligent survey systems towards open science. New Generation Computing , 397–421.

[26] Fedus, W., Zoph, B., Shazeer, N., 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.

[27] Fei, H., Ren, Y., Ji, D., 2020. Retrofitting structure-aware transformer language model for end tasks, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 2151–2161.

[28] Fei, H., Ren, Y., Zhang, Y., Ji, D., Liang, X., 2021. Enriching contextualized language model from knowledge graph for biomedical information extraction. Briefings in bioinformatics 22, bbaa110.

[29] Fei, H., Wu, S., Li, J., Li, B., Li, F., Qin, L., Zhang, M., Zhang, M., Chua, T.S., 2022. Lasuie: Unifying information extraction with latent adaptive structure-aware generative language model. Advances in Neural Information Processing Systems 35, 15460–15475.

[30] Gao, H., Zhu, X., Lin, S., Dai, J., . Deformable kernels: Adapting effective receptive fields for object deformation, in: International Conference on Learning Representations.

[31] Gehman, S., Gururangan, S., Sap, M., Choi, Y., Smith, N.A., 2020. Realtoxicityprompts: Evaluating neural toxic degeneration in language models, in: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 3356–3369.

[32] Goel, R., Kazemi, S.M., Brubaker, M., Poupart, P., 2020. Diachronic embedding for temporal knowledge graph completion, in: Proceedings of the AAAI conference on artificial intelligence, pp. 3988–3995.

[33] Gu, Y., Zhou, T., Cheng, G., Li, Z., Pan, J.Z., Qu, Y., 2019. Relevance Search over Schema-Rich Knowledge Graphs, in: Proc. of the 12th ACM International WSDM Conference (WSDM2019), pp. 114–122.

[34] Guha, R., McCool, R., Miller, E., 2003. Semantic search, in: WWW '03: Proceedings of the 12th international conference on World Wide Web, pp. 700–709.

[35] Ha, D., Dai, A.M., Le, Q.V., 2017. Hypernetworks, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=rkpACe1lx.

[36] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., Wang, Y., 2021. Dynamic neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence .

[37] Hase, P., Diab, M., Celikyilmaz, A., Li, X., Kozareva, Z., Stoyanov, V., Bansal, M., Iyer, S., 2023. Methods for measuring, updating, and visualizing factual beliefs in language models, in: Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Dubrovnik, Croatia. pp. 2714–2731. URL: https://aclanthology.org/2023.eacl-main.199.

[38] Holger, K., Daniel, O., Phil, T., Evan, W., Z., P.J., Michael, U., 2006. A Semantic Web Primer for Object-Oriented Software Developers. W3C Working Group Note 9 March 2006. W3C.

[39] Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E., 1991. Adaptive mixtures of local experts. Neural computation 3, 79–87.

[40] Jang, J., Ye, S., Yang, S., Shin, J., Han, J., KIM, G., Choi, S.J., Seo, M., 2022. Towards continual knowledge learning of language models, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=vfsRB5MImo9.

[41] Kazemi, S.M., Poole, D., 2018. Simple embedding for link prediction in knowledge graphs. Advances in neural information processing systems 31.

[42] Kelley, A., Garijo, D., 2021. A framework for creating knowledge graphs of scientific software metadata. Quant. Sci. Stud. 2, 1423–1446.

[43] Kingma, D.P., Ba, J., 2015. Adam: A method for stochastic optimization. CoRR abs/1412.6980.

[44] Kulshreshtha, A., Adiwardana, D.D.F., So, D.R., Nemade, G., Hall, J., Fiedel, N., Le, Q.V., Thoppilan, R., Luong, T., Lu, Y., et al., 2020. Towards a human-like open-domain chatbot .

[45] Kuncoro, A., Dyer, C., Rimell, L., Clark, S., Blunsom, P., 2019. Scalable syntax-aware language models using knowledge distillation, in: Proceedings of the 57th Annual Meeting of the Association for

Computational Linguistics, pp. 3472–3484.

[46] Levy, O., Seo, M., Choi, E., Zettlemoyer, L., 2017. Zero-shot relation extraction via reading comprehension, in: Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), Association for Computational Linguistics, Vancouver, Canada. pp. 333–342. URL: https://aclanthology.org/K17-1034, doi:10.18653/v1/K17-1034.

[47] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 7871–7880.

[48] Lin, S., Hilton, J., Evans, O., 2022. Truthfulqa: Measuring how models mimic human falsehoods, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 3214–3252.

[49] Liu, C., Tao, C., Feng, J., Zhao, D., 2022. Multi-granularity structural knowledge distillation for language model compression, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1001–1011.

[50] Liu, J., Wang, C., Li, C., Li, N., Deng, J., Pan, J.Z., 2021. DTN: Deep triple network for topic specific fake news detection. J. Web Semant. 70.

[51] Ma, N., Zhang, X., Huang, J., Sun, J., 2020. Weightnet: Revisiting the design space of weight networks, in: European Conference on Computer Vision, Springer. pp. 776–792.

[52] Meng, K., Bau, D., Andonian, A., Belinkov, Y., 2022. Locating and editing factual associations in gpt. Advances in Neural Information Processing Systems 35, 17359–17372.

[53] Mitchell, E., Lin, C., Bosselut, A., Finn, C., Manning, C.D., 2021. Fast model editing at scale, in: International Conference on Learning Representations.

[54] Mitchell, E., Lin, C., Bosselut, A., Manning, C.D., Finn, C., 2022. Memory-based model editing at scale, in: International Conference on Machine Learning, PMLR. pp. 15817–15831.

[55] Mohamed, S.K., Nováček, V., Nounu, A., 2020. Discovering protein drug targets using knowledge graph embeddings. Bioinformatics 36, 603–610.

[56] Mullapudi, R.T., Mark, W.R., Shazeer, N., Fatahalian, K., 2018. Hydranets: Specialized dynamic architectures for efficient inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8080–8089.

[57] Nguyen, D.Q., Vu, T., Nguyen, T.D., Nguyen, D.Q., Phung, D.Q., 2019. A capsule network-based embedding model for knowledge graph completion and search personalization., in: NAACL-HLT (1), NAACL-HLT. pp. 2180–2189.

[58] Pan, J., Calvanese, D., Eiter, T., Horrocks, I., Kifer, M., Lin, F., Zhao, Y., 2017a. Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering. Springer.

[59] Pan, J.Z., Edelstein, E., Bansky, P., Wyner, A., 2021. A Knowledge Graph Based Approach to Social Science Surveys. Data Intell. 3, 477–506.

[60] Pan, J.Z., Pavlova, S., Li, C., Li, N., Li, Y., Liu, J., 2018. Content based Fake News Detection Using Knowledge Graphs, in: Proc. of the International Semantic Web Conference (ISWC2018), pp. 669–683.

[61] Pan, J.Z., Staab, S., Aßmann, U., Ebert, J., Zhao, Y. (Eds.), 2013. Ontology-Driven Software Development. Springer.

[62] Pan, J.Z., Stamou, G., Tzouvaras, V., Horrocks, I., 2005. f-SWRL: A Fuzzy Extension of SWRL, in: Proc. of ICANN.

[63] Pan, J.Z., Taylor, S., Thomas, E., 2009. Reducing Ambiguity in Tagging Systems with Folksonomy Search Expansion, in: the Proc. of the 6th European Semantic Web Conference (ESWC2009).

[64] Pan, J.Z., Vetere, G., Gomez-Perez, J., Wu, H. (Eds.), 2017b. Exploiting Linked Data and Knowledge Graphs for Large Organisations. Springer.

[65] Pan, J.Z., Zhao, Y. (Eds.), 2014. Semantic Web Enabled Software Engineering. IOS Press.

[66] Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., Miller, A., 2019. Language models as knowledge bases?, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 2463–2473.

[67] Phil, T., Z., P.J., Daniel, O., Evan, W., Michael, U., Elisa, K., 2006. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. W3C Working Draft Working Group Note 2006/02/11.

[68] Qi, G., Pan, J.Z., Ji, Q., 2007. A Possibilistic Extension of Description Logics, in: Proc. of International Workshop on Description Logics (DL2007).

[69] Ribeiro, M.T., Singh, S., Guestrin, C., 2016. Model-agnostic interpretability of machine learning. arXiv preprint arXiv:1606.05386 .

[70] Roberts, A., Raffel, C., Shazeer, N., 2020. How much knowledge can you pack into the parameters of a language model?, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 5418–5426.

[71] Rospocher, M., van Erp, M., Vossen, P., Fokkens, A., Aldabe, I., Rigau, G., Soroa, A., Ploeger, T., Bogaard, T., 2016. Building event-centric knowledge graphs from news. J. Web Semant. 37-38.

[72] Sensoy, M., Fokoue, A., Pan, J.Z., Norman, T., Tang, Y., Oren, N., Sycara, K., 2013. Reasoning about Uncertain Information and Conflict Resolution through Trust Revision, in: Proc. of AAMAS.

[73] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q.V., Hinton, G.E., Dean, J., 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. Learning .

[74] Sinitsin, A., Plokhotnyuk, V., Pyrkin, D., Popov, S., Babenko, A., 2020. Editable neural networks, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=HJedXaEtvS.

[75] Stoilos, G., Stamou, G.B., Pan, J.Z., 2006. Handling Imprecise Knowledge with Fuzzy Description Logic, in: Proc. of International Workshop on Description Logics.

[76] Thorne, J., Vlachos, A., Christodoulopoulos, C., Mittal, A., 2018a. Fever: a large-scale dataset for fact extraction and verification, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 809–819.

[77] Thorne, J., Vlachos, A., Christodoulopoulos, C., Mittal, A., 2018b. FEVER: a large-scale dataset for fact extraction and VERification, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Association for Computational Linguistics, New Orleans, Louisiana. pp. 809–819. URL: https://aclanthology.org/N18-1074, doi:10.18653/v1/N18-1074.

[78] Tripodi, I.J., Callahan, T.J., Westfall, J.T., Meitzer, N.S., Dowell, R.D., Hunter, L.E., 2020. Applying knowledge-driven mechanistic inference to toxicogenomics. Toxicology in Vitro .

[79] Tu, M., Berisha, V., Cao, Y., Seo, J.s., 2016. Reducing the model order of deep neural networks using information theory, in: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE. pp. 93–98.

[80] Voita, E., Sennrich, R., Titov, I., 2019. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 4396–4406.

[81] Wang, C., Liu, P., Zhang, Y., 2021. Can generative pre-trained language models serve as knowledge bases for closed-book qa?, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp. 3241–3251.

[82] Wang, J., Huang, W., Qiu, M., Shi, Q., Wang, H., Li, X., Gao, M., 2022a. Knowledge prompting in pre-trained language model for natural language understanding, in: Goldberg, Y., Kozareva, Z., Zhang,

Y. (Eds.), Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, Association for Computational Linguistics. pp. 3164–3177. URL: https://aclanthology.org/2022.emnlp-main.207.

[83] Wang, L., Zhao, W., Wei, Z., Liu, J., 2022b. Simkgc: Simple contrastive knowledge graph completion with pre-trained language models, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 4281–4294.

[84] Wang, X., He, X., Cao, Y., Liu, M., Chua, T.S., 2019. KGAT: Knowledge Graph Attention Network for Recommendation, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2019), pp. 950–958.

[85] Wang, X., Yu, F., Dunlap, L., Ma, Y.A., Wang, R., Mirhoseini, A., Darrell, T., Gonzalez, J.E., 2020. Deep mixture of experts via shallow embedding, in: Uncertainty in artificial intelligence, PMLR. pp. 552–562.

[86] Wu, H., Wang, M., Zeng, Q., Chen, W., Nind, T., Jefferson, E.R., Bennie, M., Black, C., Pan, J.Z., Sudlow, C., Robertson, D., 2020. Knowledge Driven Phenotyping, in: Proc. of Medical Informatics Europe (MIE 2020), pp. 1327–1328.

[87] Xian, Y., Fu, Z., Muthukrishnan, S., de Melo, G., Zhang, Y., 2019. Reinforcement knowledge graph reasoning for explainable recommendation, in: Proceedings of SIGIR, pp. 285–294.

[88] Xie, C., Yu, B., Zeng, Z., Yang, Y., Liu, Q., 2021. Multilayer internet-of-things middleware based on knowledge graph. IEEE Internet Things J. 8, 2635–2648.

[89] Xu, H., Giunchiglia, F., 2015. Sko types: an entity-based scientific knowledge objects metadata schema. Journal of Knowledge Management 19, 60–70.

[90] Yang, Y., Huang, C., Xia, L., Li, C., 2022. Knowledge graph contrastive learning for recommendation, in: Proceedings of SIGIR, pp. 1434–1443.

[91] Yao, L., Mao, C., Luo, Y., 2019. Kg-bert: Bert for knowledge graph completion. arXiv preprint arXiv:1909.03193 .

[92] Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., Finn, C., 2020. Gradient surgery for multi-task learning. Advances in Neural Information Processing Systems 33, 5824–5836.

[93] Zeng, X., Tu1, X., Liu, Y., Fu, X., Su, Y., 2022. Toward better drug discovery with knowledge graph. Current Opinion in Structural Biology 72, 114–126.

[94] Zhang, R., Hristovski, D., Schutte, D., Kastrin, A., Fiszman, M., Kilicoglu, H., 2021. Drug repurposing for covid-19 via knowledge graph completion. Journal of Biomedical Informatics 115.

[95] Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M., Liu, Q., 2019. Ernie: Enhanced language representation with informative entities, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 1441–1451.

[96] Zhu, C., Rawat, A.S., Zaheer, M., Bhojanapalli, S., Li, D., Yu, F., Kumar, S., 2020. Modifying memories in transformer models. arXiv:2012.00363.

[97] Zhu, X., Ao, X., Qin, Z., Chang, Y., Liu, Y., He, Q., Li, J., 2021. Intelligent financial fraud detection practices in post-pandemic era. The Innovation 2.