Knowledge Editing for Multi-Hop Question Answering Using Semantic Analysis

Dominic Simon , Rickard Ewetz
University of Florida
{dominic.simon, rewetz}@ufl.edu,

Abstract

Large Language Models (LLMs) require lightweight avenues of updating stored information that has fallen out of date. Knowledge Editing (KE) approaches have been successful in updating model knowledge for simple factual queries but struggle with handling tasks that require compositional reasoning such as multi-hop question answering (MQA). We observe that existing knowledge editors leverage decompositional techniques that result in illogical reasoning processes. In this paper, we propose a knowledge editor for MQA based on semantic analysis called CHECK. Our framework is based on insights from an analogy between compilers and reasoning using LLMs. Similar to how source code is first compiled before being executed, we propose to semantically analyze reasoning chains before executing the chains to answer questions. Reasoning chains with semantic errors are revised to ensure consistency through logic optimization and re-prompting the LLM model at a higher temperature. We evaluate the effectiveness of CHECK against five state-of-the-art frameworks on four datasets and achieve an average 22.8% improved MQA accuracy.

1 Introduction

Large Language Models (LLM) are trained on extensive amounts of data, enabling them to grasp the statistical patterns of natural language and broad factual knowledge [Brown *et al.*, 2020]. The factual knowledge is utilized when LLMs are integrated into applications such as chatbots, translators, and question-answering systems [Zhu *et al.*, 2024]. It is unavoidable that the factual knowledge stored within LLMs becomes outdated over time. Retraining the LLMs from scratch to learn new factual data is both economically expensive [Li *et al.*, 2023] and introduces an undue burden on the environment [Faiz *et al.*, 2024]. The concept of knowledge editing (KE) has emerged as a promising solution to bypass the need for retraining LLMs from scratch. Knowledge editing approaches commonly fall into two categories: parameter-based approaches that inject edits directly into model parameters [Meng *et al.*, 2022; Meng *et al.*, 2023; Yu *et al.*, 2023; Gupta *et al.*, 2023; Hase *et al.*, 2023] and memory-based methods that introduce additional parameters for edit injection [Mitchell *et al.*, 2022; Madaan *et al.*, 2022; Yu *et al.*, 2024; Wang *et al.*, 2024a]. Both of these solution strategies have demonstrated success for basic questioning answering problems [Meng *et al.*, 2023; Mitchell *et al.*, 2022]. However, the problem becomes immensely more challenging when the questions involve compositional reasoning, such as multi-hop question answering (MQA), where the intermediate knowledge between a hop could have been edited. For example, *What is the country of citizenship of the author of Harry Potter*, where an edit could have changed the *country of citizenship* of *JK Rowling* from *United Kingdom* to *United States*. This emerging challenge has recently spurred further investigations into knowledge editing for MQA problems [Chen *et al.*, 2024; Shi *et al.*, 2024].

State-of-the-art knowledge editors for MQA rely on decomposing the multi-hop problems into multiple single-hop parts [Zhong *et al.*, 2023; Gu *et al.*, 2024; Wang *et al.*, 2024b]. The decomposition allows the knowledge editors to compare the intermediate facts with edits stored in a memory bank. The decomposition is performed using an LLM through long in-context examples. However, this approach is prone to leveraging illogical reasoning processes and accidentally utilizing non-relevant edited facts. An intuitive approach to improving existing solutions would be to leverage explicit question decomposition. Nevertheless, explicitly decomposing multi-hop questions into single-hop questions is not straightforward because it may introduce errors from the loss of context, nuances, and hallucinations.

In this paper, we propose a framework for knowledge editing based on semantic analysis called CHECK. The framework is based on insights from an analogy between compilers and reasoning using LLMs. Source code is first required to pass semantic analysis tests such as type checking before being compiled into a binary that can be executed. Inspired by this approach, we propose to semantically *type check* the reasoning chains generated by LLMs for solving MQA problems. The main contributions of this paper are summarized, as follows:

- We propose the concept of semantically analyzing the reasoning process of knowledge editors. Each hop in a multi-hop question is assigned a type in the form of person, place, or thing. Next, each of the input and output types within each hop of a reasoning chain are checked for consistency.

- Semantic inconsistencies are resolved by formulating optimization problems to repair the reasoning chains by rearranging the extracted relationships or re-prompting the LLM for a new reasoning chain at a higher temperature.
- Experimental evaluation on the MQuAKE dataset demonstrates that CHECK achieves an average 22.8% greater accuracy than other similar approaches across three open-source LLMs.

The remainder of the paper is broken into the following sections: preliminary knowledge is discussed in Section 2, the motivation behind using type checking is given in Section 3, the methodology of the CHECK framework is explained in Section 4, experimental results across 4 datasets and 3 LLMs are provided in Section 5, and the conclusion is in Section 6.

2 Preliminaries

The problem formulation of knowledge editing is provided in Section 2.1. Related works are discussed in Section 2.2.

2.1 Problem Formulation

This paper addresses the problem of Multi-hop Question Answering (MQA) under Knowledge Editing. A single factual association can be viewed as a subject s, relation r, object o triple $t = (s, r, o)$, where *Akira Toriyama was born in Japan* can be converted to $(Akira\ Toriyama,\ born\ in,\ Japan)$. Editing a factual relation is updating o to become a new entity o' so that the edited triple becomes $t' = (s, r, o')$. A factual association can be expressed in the form of a question $q = (s, r) \rightarrow o$, where o is unknown until the question is answered. Multi-hop questions $\mathcal{Q}$ contain a chain of relations $\mathcal{Q} = (r_0, r_1, ..., r_n)$ that can be viewed as a set of subquestions $((s, r_0) \rightarrow o_1, (o_1, r_1) \rightarrow o_2, ..., (o_{n-1}, r_{n-1}) \rightarrow o_n)$ that must be iteratively solved to uncover the obscured entities until the final answer is found. For example, the multi-hop question *Where is the birthplace of the creator of Dragonball?* contains the relations $(creator,\ birthplace)$, which translate into the subquestions $((Dragonball,\ creator) \rightarrow Toriyama,\ (Toriyama,\ birthplace) \rightarrow Japan)$. Answering any of the subquestions q of a multi-hop question $\mathcal{Q}$ with edited information t' will cause the subsequent subquestion answers o'_n to deviate from the original answer path, such that $\mathcal{Q} = ((s, r_0) \rightarrow o'_1, (o'_1, r_1) \rightarrow o'_2))$. Generating a correct subquestion path to traverse and determining whether a subquestion requires an edited answer are the two main challenges of MQA under KE.

2.2 Related Works

In this section, we review studies on knowledge editing for LLMs. Early investigation on KE using parameter-based approaches include [Meng *et al.*, 2022; Mitchell *et al.*, 2022; Meng *et al.*, 2023]. However, solution strategies that directly modify model parameters face issues such as catastrophic forgetting [Gupta *et al.*, 2024], one-way associations [Meng *et al.*, 2022; Meng *et al.*, 2023], and long training times [Yu *et al.*, 2024]. Other investigations have focused on augmenting the LLM with external knowledge graphs [Cheng *et al.*, 2024; Shi *et al.*, 2024; Chen *et al.*, 2024]. However, such solutions are constrained to applications where such graphs are available [Baldazzi *et al.*, 2023]. Embedding-based editors store edits in an embedding space for compact retrieval [Zhong *et al.*, 2023; Gu *et al.*, 2024]. MeLLo [Zhong *et al.*, 2023] uses the dense retrieval model Contriever [Izacard *et al.*, 2021] to store factual edit sentences in an embedding space. Next, an in-context learning prompt is used to break the initial question into subquestions and check the subquestion answer against the most similar embedded answer for factual conflicts. PokeMQA [Gu *et al.*, 2024] uses a similar prompting scheme, but it removes the burden of determining conflict from the LLM and trains a two-level conflict disambiguation network to determine whether the subquestion answer and retrieved embedding contain conflicting information.

Method	Question Decomposition	Verification of Decomposition
GMeLLo	✓	$\emptyset$
MeLLo	✓	$\emptyset$
DeepEdit	✓	$\emptyset$
PokeMQA	✓	$\emptyset$
CHECK	✓	✓

Table 1: Knowledge Editing steps included by state-of-the-art frameworks [Chen *et al.*, 2024; Zhong *et al.*, 2023; Wang *et al.*, 2024b; Gu *et al.*, 2024].

The knowledge editing steps included by state-of-the-art frameworks are shown in Table 1. Neither the above knowledge editors nor similar ones have any way to ensure that the generated subquestions are being answered in an order reflecting the original multi-hop question. This results in misordered chains of subquestions, leading to a question-answering flow that will never arrive at the correct answer. The proposed CHECK framework resolves this issue by type checking the subquestion reasoning process using semantic analysis.

3 Semantic Analysis

Our proposed knowledge editing solution is based on insights from an analogy between compilers and reasoning using LLMs. Source code is converted into a binary executable through a compilation process consisting of preprocessing, semantic parsing, assembly conversion, and linking. Next, the binary can be executed to compute an output. Semantic parsing involves type checking to ensure each function call has arguments that match the function definitions. For example, checking that a function expecting an argument of type double is not passed an argument of type char. The compilation process eliminates syntactic and semantic errors, which reduces debugging of the executable to value errors. We propose to adapt this method of semantic analysis to knowledge editing for MQA by type checking the reasoning chains and knowledge edits. The type checking will ensure that the reasoning processes are logical and will assist in eliminating hal-

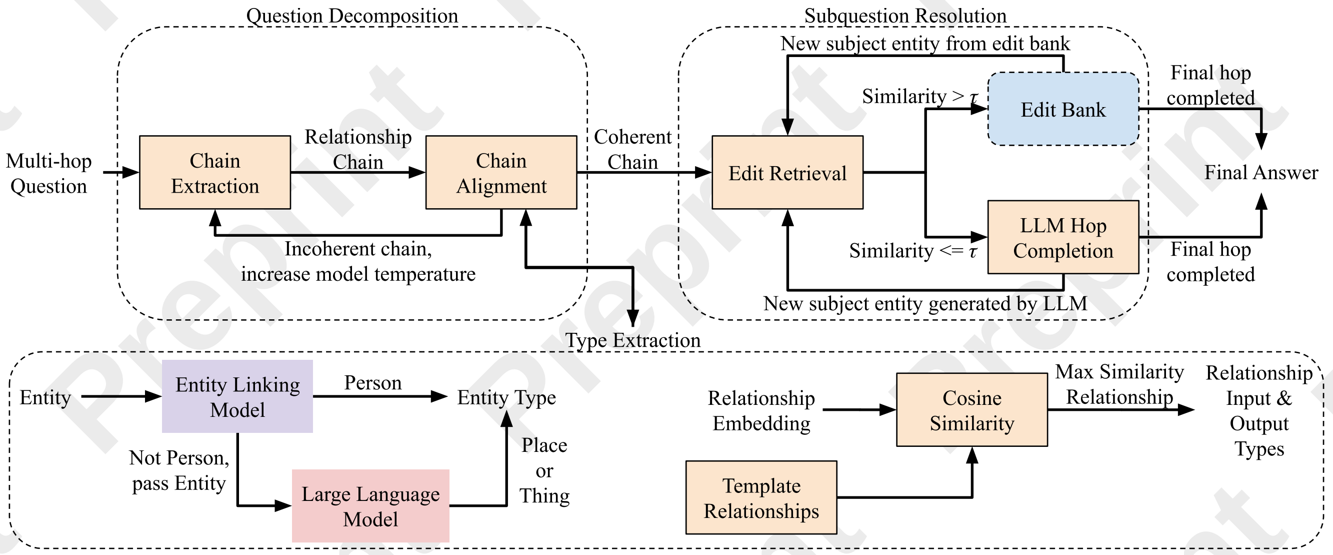

Figure 1: The flow of the proposed CHECK knowledge editor. CHECK processes the initial multi-hop question by decomposing it into a chain of relationships. Misaligned chains are realigned through type checking and model temperature increases. Next, each entity and relationship pair is checked to see if an edit is necessary. If an edit is required, the object corresponding to the edit triple is provided as the next entity. Otherwise, the LLM provides the next entity.

lucinations from LLM responses.

In this paper, we propose to categorize all entities as $\mathrm{persons}$, places, or things. Optionally, more fine-grained type categories can be used. Consequently, relationships can be viewed as functions that expect inputs and outputs of $\mathrm{persons}$, places, or things. In the sentence *Akira Toriyama was born in Japan*, the relationship *born in* expects an entity of type person (*Toriyama*) as input and place (*Japan*) as output. We propose to decompose multi-hop questions into single-hop questions and iteratively resolve each relationship. If the question is decomposed correctly, the output types of one relationship are expected to overlap the input types of the next relationship. If there is no overlap between the input and output of neighboring relationships, then the semantic analysis has revealed an error that is required to be corrected before the relationship chain is evaluated to answer the MQA.

4 Methodology

In this section, we present the methodology of the CHECK framework. The input to the CHECK framework is a multi-hop question and a set of factual edits. The output is an answer to the questions. The CHECK framework consists of a type extraction step, a multi-hop question decomposition step, and a subquestion resolution step. The flow of the framework is illustrated in Figure 1. The type extraction step involves developing functions and a library for extracting the type of entities and relationships, respectively. The details of the type extraction is provided in Section 4.1. The multi-hop question decomposition step involves decomposing the multi-hop question into a chain of relationships that represent each hop and the initial multi-hop question entity. The relationship chain is then checked for type alignment and realigned if necessary. The details of question decomposition are explained in Section 4.2. The final step is to iteratively traverse the relationship chain until the answer entity is found in the subquestion resolution step. Within each iteration, the entity and relationship are compared against edits to determine whether it is necessary to insert edited information. The details of the relationship chain traversal are provided in Section 4.3.

4.1 Type Extraction

In this section, we describe how the types for both entities and relationships are extracted. The entity types are extracted using a combination of entity linking models and LLMs during MQA. In contrast, we pre-characterize a library of input and output types for the relationships. During MQA, the library is queried to obtain the input and output type of each relationship. The approach to generating types for entities and relationships is different because the number of different entities is very large and cannot be enumerated ahead of time. On the other hand, there is only a limited number of relationship that connect persons, places, and things. Therefore, it is possible to pre-characterize the different types of relationships into a template library, for quick and reliable access at runtime.

Entity Type Extraction: The objective of entity type extraction is to determine if an entity is of type person, place, or thing. We first pass the entity to an entity linking model, which can accurately decide whether the entity is a person or not. CHECK uses the ReFinED [Ayoola *et al.*, 2022] entity linking model for its short inference times and accurate entity linking. If the entity is not a person, then the entity is passed to the language model $\mathcal{F}$ to determine whether the entity is a place or thing. The prompt and in-context examples are provided in Section 7 of the Appendix.

Relationship Type Extraction: The relationships from given edits are extracted to build a relationship template library, where template relationships r_t are encoded using a dense retrieval model and act as keys to access their input

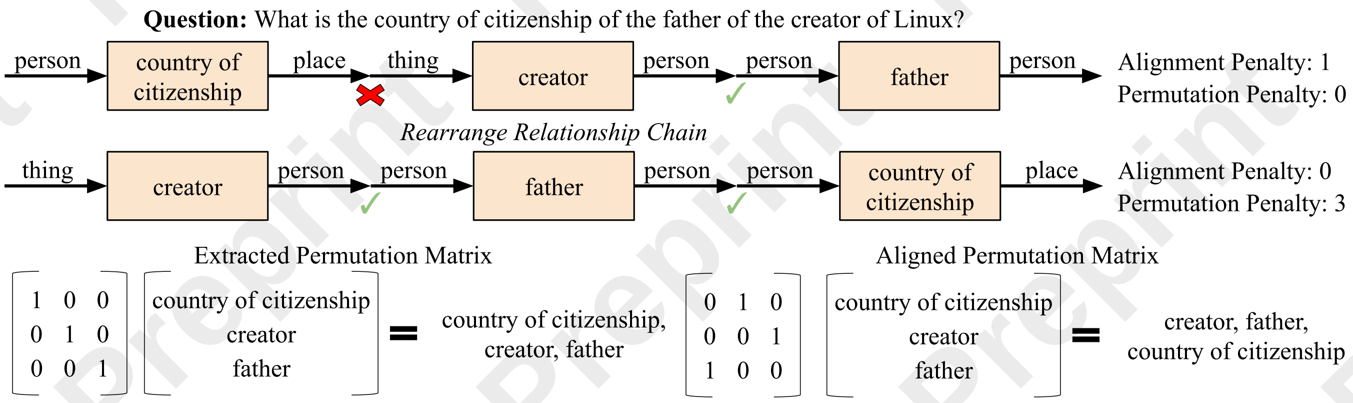
Question: What is the country of citizenship of the father of the creator of Linux?
person
country of citizenship
place
thing
creator
person
person
father
person
Alignment Penalty: 1
Permutation Penalty: 0
Rearrange Relationship Chain
thing
creator
person
person
father
person
person
country of citizenship
place
Alignment Penalty: 0
Permutation Penalty: 3
Extracted Permutation Matrix
country of citizenship, creator, father
Aligned Permutation Matrix
creator, father, country of citizenship

Figure 2: Example of relationship chain repair on a misaligned chain. Initially, there is an alignment penalty of 1. After two permutation steps, the relationship chain is realigned.

and output types. During MQA, the template relationships r_t are compared against embedded multi-hop question relationships r using cosine similarity. Question relationships r take on the input and output types of the most similar r_t. Edits are expected to be provided as triples, so r_t can be taken directly from the edits. Relationships can have multiple input and output types, but LLMs struggle to correctly assign multiple types at once. To generate an accurate template library, the relationship types are manually labeled, which is feasible due to the limited number of relationships.

4.2 Question Decomposition

In this section, we explain how a relationship chain is extracted from a multi-hop question. CHECK receives the multi-hop question and extracts a chain of hops that will be traversed to find the answer to the multi-hop question. Then, the input and output entity types of each relationship in the chain are checked to ensure the types are aligned. If the chain is misaligned, it is reconstructed to create a new relationship chain that is aligned.

Chain Extraction: The multi-hop question $\mathcal{Q}$ is passed to the LLM $\mathcal{F}$ along with an in-context learning prompt found, in Section 7 of the Appendix, in order to extract the relationship chain $\mathcal{R}$. Relationship chain extraction is outlined as follows:

$$\mathcal{R} = (r_0, r_1, ..., r_n) = \mathcal{F}(\mathcal{Q}), \tag{1}$$

where $\mathcal{R}$ is a chain of relationships $(r_0, r_1, ..., r_n)$. The initial entity o_n from $\mathcal{Q}$ is extracted using an entity linking model. The relationship chain $\mathcal{R}$ can be iteratively traversed backwards to generate triples (o_n, r_n, o_{n-1}) until o_{n-1} is the final answer o_0 to $\mathcal{Q}$. The traversal process can be viewed as a series of function calls where r_n is a function that takes o_n as input and outputs o_{n-1} such that $o_0 = r_1(r_2(r_n(o_n)))$. Relationship chain traversal is described in-depth in Section 4.3.

Chain Alignment: The extracted relationship chains $\mathcal{R}$ have been observed to contain misaligned relationships r. Misalignment occurs when the ordering of r within $\mathcal{R}$ does not match the ordering of relationships within the original multi-hop question. To check the alignment of the extracted chain, all r in $\mathcal{R}$ are given input $\mathcal{T}_{in}$ and output $\mathcal{T}_{out}$ types, as described in Section 4.1. The relationships types are combined to form a chain $\mathcal{C}$ of types corresponding to $\mathcal{R}$, such that:

$$\mathcal{C} = [(\mathcal{T}_{in}^0, \mathcal{T}_{out}^0), (\mathcal{T}_{in}^1, \mathcal{T}_{out}^1), ..., (\mathcal{T}_{in}^n, \mathcal{T}_{out}^n)]. \tag{2}$$

The alignment penalty $\mathcal{A}$ for a type chain $\mathcal{C}$ is determined by the number of input and output type pairs n that are misaligned. The alignment of $\mathcal{C}$ is described as follows:

$$\begin{aligned} \mathcal{A} &= \sum_{i=1}^{n} m_i, \\ \text{s.t. } m_i &= \begin{cases} 0 & \mathcal{T}_i^{out} == \mathcal{T}_{i+1}^{in} \\ 1 & \mathcal{T}_i^{out} \neq \mathcal{T}_{i+1}^{in}, \end{cases} \end{aligned} \tag{3}$$

where a misalignment $\mathcal{T}_i^{out} \neq \mathcal{T}_{i+1}^{in}$ carries a penalty of 1, while an alignment $\mathcal{T}_i^{out} == \mathcal{T}_{i+1}^{in}$ carries no penalty. The penalty values m_i are summed to get $\mathcal{A}$. If $\mathcal{A} == 0$, $\mathcal{C}$ is properly aligned and is passed to the subquestion resolution step. If $\mathcal{A} > 0$, CHECK attempts to find an aligned $\mathcal{C}$.

Relationship Chain Repair: When a relationship chain $\mathcal{R}$ is not aligned, we have observed that $\mathcal{R}$ often contains the correct relationships, only in an incorrect order. Therefore, there exists an opportunity to correct the misaligned chains by permuting the relationships to find an aligned chain. The relationship chain repair step starts by generating all permutations of the relationship type chain $\mathcal{C}$. The repair is successful if one of the permutations c has an alignment penalty of zero $\mathcal{A}(c) = 0$. However, there may be multiple permuted chains that have an alignment penalty of zero. If multiple c exist where $A(c) = 0$, we select c that has the smallest permutation cost λ, the c that required the smallest number of permutations to eliminate the alignment penalty.

To find λ for each c, the permutation matrix $\mathcal{P}$ used to generate c is examined. Generating c using $\mathcal{P}$ can be described as $\mathcal{P} \cdot \mathcal{R} = c$, and expanded as:

$$\begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1 \end{pmatrix} \begin{pmatrix} \mathcal{C}_0 \\ \mathcal{C}_1 \\ ... \\ \mathcal{C}_n \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ ... \\ c_n \end{pmatrix}, \tag{4}$$

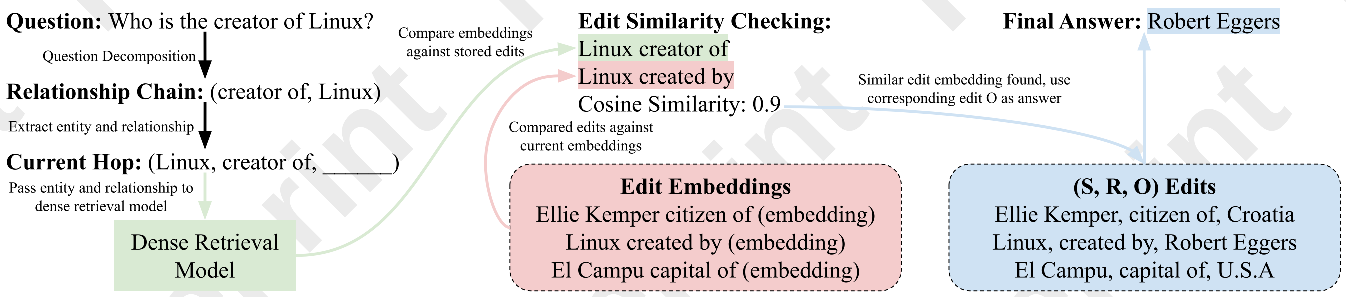
Question: Who is the creator of Linux?
Question Decomposition
Relationship Chain: (creator of, Linux)
Extract entity and relationship
Current Hop: (Linux, creator of, ______)
Pass entity and relationship to dense retrieval model
Dense Retrieval Model
Compare embeddings against stored edits
Edit Similarity Checking:
Linux creator of
Linux created by
Cosine Similarity: 0.9
Compared edits against current embeddings
Similar edit embedding found, use corresponding edit O as answer
Final Answer: Robert Eggers
Edit Embeddings
Ellie Kemper citizen of (embedding)
Linux created by (embedding)
El Campu capital of (embedding)
(S, R, O) Edits
Ellie Kemper, citizen of, Croatia
Linux, created by, Robert Eggers
El Campu, capital of, U.S.A

Figure 3: Example of CHECK answering the question *Who was the creator of Linux?* The subject and relationship are extracted during question decomposition and rearranged into a single triple with the object unknown. The entity and relationship are embedded using a dense retrieval model and compared against the stored edit embeddings. A similar edit is found, so the object of the corresponding triple is used as the next entity / final answer.

where $\mathcal{C}_n$ and c_n correspond to the input / output type pairs within the the original type chain $\mathcal{C}$ and its permutations c.

The rows in $\mathcal{P}$ can be reordered to obtain a different permutation c, as long as each row and column has exactly 1 non-zero value. CHECK attempts to find a c where $\mathcal{A}(c) = 0$ and the permutation penalty λ of $\mathcal{P}$ is minimal. The penalty λ of the permutation matrix $\mathcal{P}$ is computed as follows:

$$\lambda = N - \sum_{i=1}^{N} \mathcal{P}_{i,i}, \tag{5}$$

where $\mathcal{P}_{r,c}$ denotes the element in row r and column c of $\mathcal{P}$. The summation computes the number of elements that are on the diagonal, which corresponds to the number of elements in their "unpermuted" position. Subtracting the sum from the number of diagonal elements N provides the number of rows that have been permuted, also known as the permutation cost λ for the current permutation c. The initial permutation matrix $\mathcal{P}$ will have the value 1 on all diagonal matrix elements $\mathcal{P}_{i,i}$, so the λ for $\mathcal{C}$ will be 0, while λ for any permutation c will be non-zero.

Equation 3 is applied to all generated c. If at least 1 permutation c where $\mathcal{A}(c) = 0$ exists, the c where $A(c) = 0$ with the lowest λ is used as the new $\mathcal{C}$ and the corresponding permutation of $\mathcal{R}$ is passed to the subquestion resolution step.

If an aligned c does not exist, then question decomposition is restarted and a higher LLM temperature is used during chain extraction. We use a temperature scale of 0.0 to 1.0 on increments of 0.1. After 1.0, the model responses tend to become too varied to be useful. The 0.1 increment gives a good trade-off between exploring many options and while still having a reasonable run time. If no aligned c is found, the c with the least $\mathcal{A}$ is used during subquestion resolution.

4.3 Subquestion Resolution

The input to the subquestion resolution step is an entity o_n and relationship r. The goal is to generate the next entity o_{n-1}. Starting with the initial entity and relationship, the current o_n and r are compared against stored edits. If an edit that is sufficiently similar to one of the inputs exists, the next entity o_{n-1} will be an edited object o' from the corresponding edit triple t'. Otherwise, the LLM is prompted to generate and answer a question based on the current o_n and r to find o_{n-1}. This process is repeated until the final multi-hop answer has been obtained. Each entity and relationship is embedded using a dense retrieval model and is compared against all edit embeddings using cosine similarity. An example of subquestion resolution is provided in Figure 3.

Edit Storage: Prior to MQA, CHECK receives edits as triples and stores them as strings and embeddings. Edit triples t' are stored as comma-separated lists. The subject s of t' is passed to an entity linking model to get the true name s^* of the entity. The edit triple t' is then updated with s^* such that $t' = (s^*, r, o')$.

Edits t' are also stored in an embedding space. The subject s and relationship r of t' are combined into one string and passed to an embedding model to get an edit embedding. Following previous works, we use the Contriever [Izacard *et al.*, 2021] dense retrieval model. These embeddings are then used during relationship chain traversal to aid in determining whether an edit needs to be made.

Edit Retrieval: The initial entity o_n is passed to an entity linking model to generate its true name o_n^*. The initial entity true name o_n^* is compared against all true edit names s^* previously inserted into CHECK. Next, the list of edit triples $\mathcal{L}_t$ is searched for corresponding edits as follows:

$$\mathcal{E}_{search} = \begin{cases} \mathcal{E}_{s^*}, & \text{if } o_n^* \in \mathcal{L}_t, \\ \mathcal{E}, & \text{if } o_n^* \notin \mathcal{L}_t, \end{cases} \tag{6}$$

where the set of semantic embeddings that will be checked for necessary edits $\mathcal{E}_{search}$ is narrowed to the embeddings $\mathcal{E}_{s^*}$ where $s^* \in \mathcal{L}$ and $o_n^* == s^*$. The entity linking model can generate false positive and false negative outputs, so even if no match is found, all edited semantic embeddings $\varepsilon_e \in \mathcal{E}$ are checked to ensure that an edit is not missed.

Once $\mathcal{E}_{search}$ has been found, the initial entity o_n and relationship r_n of $\mathcal{Q}$ are extracted from $\mathcal{R}$ and passed to a dense retrieval model to generate semantic embeddings ε_c for the current hop. Then, $\varepsilon_e \in \mathcal{E}$ are compared against ε_c using cosine similarity $cos()$. If the highest similarity embedding is above a threshold τ, then the o' from the corresponding edit triple t' becomes the new o_{n-1}. If the highest similarity is below τ, then o_n and r_n move to the triple completion sub-step. Semantic embedding matching can be described algorithmi-

Dataset [Zhong *et al.*, 2023]	MQuAKE-CF-3k		MQuAKE-2002		MQuAKE-Hard		MQuAKE-T	
Accuracy Type	Case	Question	Case	Question	Case	Question	Case	Question
Model	GPT-J [Wang and Komatsuzaki, 2021]							Size: 6B
GMeLLo-QA [Chen *et al.*, 2024]	10.60	6.04	10.39	6.14	8.86	4.35	21.95	10.67
MeLLo [Zhong *et al.*, 2023]	14.97	6.89	17.18	8.13	6.76	2.64	32.82	18.49
DeepEdit [Wang *et al.*, 2024b]	19.03	13.44	27.17	19.55	6.53	3.96	55.84	41.86
PokeMQA [Gu *et al.*, 2024]	15.70	6.97	19.98	8.72	11.66	5.59	59.37	31.00
CHECK	**42.27**	**29.57**	**56.59**	**40.86**	**35.90**	**23.85**	**78.69**	**55.82**
Model	Vicuna [Chiang *et al.*, 2023]							Size: 7B
GMeLLo-QA [Chen *et al.*, 2024]	11.23	6.44	10.84	6.41	5.59	2.41	28.53	14.38
MeLLo [Zhong *et al.*, 2023]	9.93	5.08	9.84	5.13	1.86	0.85	68.52	50.18
DeepEdit [Wang *et al.*, 2024b]	13.87	8.38	20.63	12.52	0.93	0.54	34.05	19.04
PokeMQA [Gu *et al.*, 2024]	30.97	18.18	40.51	25.66	30.77	15.70	68.68	48.11
CHECK	**47.57**	**30.93**	**63.74**	**41.99**	**48.72**	**29.68**	**81.64**	**55.84**
Model	Falcon [Almazrouei *et al.*, 2023]							Size: 7B
GMeLLo-QA [Chen *et al.*, 2024]	7.77	4.27	6.50	3.63	5.36	3.34	16.38	7.57
MeLLo [Zhong *et al.*, 2023]	4.01	7.30	10.14	5.56	1.63	0.85	52.94	36.42
DeepEdit [Wang *et al.*, 2024b]	13.37	8.23	19.53	12.02	2.80	1.24	59.85	45.38
PokeMQA [Gu *et al.*, 2024]	15.77	7.64	19.93	9.14	13.05	7.46	63.97	37.76
CHECK	**39.10**	**24.10**	**52.80**	**33.72**	**45.22**	**31.08**	**81.69**	**57.51**

Table 2: Per-case and per-question accuracy across the MQuAKE subsets. The highest accuracy per column and per model is bolded. The second highest accuracy is underlined.

cally as follows:

$$o_{n-1} = \begin{cases} o_e, & \text{if } \cos(\varepsilon_c, \varepsilon_e) > \tau, \\ \text{None}, & \text{if } \cos(\varepsilon_c, \varepsilon_e) <= \tau. \end{cases} \quad (7)$$

If no new o_{n-1} is found through semantic embedding similarity, then the LLM $\mathcal{F}$ is prompted to generate the next o_{n-1}. First, $\mathcal{F}$ is prompted using in-context learning to generate a question $\mathcal{Q}_{LLM}$ based on o_n and r_n. The question-generating in-context learning prompt is provided in Section 7 of the Appendix. The LLM-generated question $\mathcal{Q}_{LLM}$ is then answered by $\mathcal{F}$ using another in-context learning prompt to ensure that only a single entity o_{n-1} is provided as an answer.

Once o_{n-1} is generated through one of the previous substeps, it is paired with r_{n-1} to complete the previous two sub-steps to find o_{n-2}. This process is iteratively completed $[(o_n, r_n, o_{n-1}) \rightarrow (o_{n-1}, r_{n-1}, o_{n-2}) \rightarrow ... \rightarrow (o_1, r_1, o_0)]$ until all r_n have been used and the final answer o_0 is found.

5 Results

The code for CHECK is available at https://github.com/dominic-simon/CHECK.

Baselines: We compare against other editors that do not rely on outside sources of factual information. Specifically, we compare against MeLLo [Zhong *et al.*, 2023], PokeMQA [Gu *et al.*, 2024], DeepEdit [Wang *et al.*, 2024b], and the question-answering portion of GMeLLo [Chen *et al.*, 2024]. We also provide comparisons against the parameter-based knowledge editors ROME [Meng *et al.*, 2022] and MEMIT [Meng *et al.*, 2023] as well as model finetuning.

Datasets: We use the MQuAKE [Zhong *et al.*, 2023] dataset to evaluate the editors. MQuAKE is composed of two subsets. The counterfactual subset contains 3000 edit cases. The subset contains questions with 2, 3, and 4 hops with 1000 cases of each. Each edit case contains betwen 1 and 4 individual edits. The temporal subset is composed of 1868 edit cases containing 1421 2-hop questions, 445 3-hop questions, and 2 4-hop questions each with only 1 edit. Two additional subsets have also been added to MQuAKE [Wang *et al.*, 2024b]. The counterfactual subset contains conflicting edit cases, so MQuAKE-2002 removes all cases with conflicting edits, resulting in a counterfactual dataset containing only 2002 edit cases. The other new subset contains 429 edit cases each with 4 hops and 4 edits.

Evaluation Metrics: Each edit case in MQuAKE contains 3 multi-hop questions conveying the same idea with the same number of hops in slightly different words. An edit case is considered correct if the editor correctly answers at least 1 question. We also track the number of questions each editor has answered correctly. Per-case accuracy is determined as correct cases ÷ total cases and per-question accuracy is determined as correct questions ÷ total questions.

Models: We compare the baselines across 3 models: GPT-J [Wang and Komatsuzaki, 2021], Vicuna-7B [Chiang *et al.*, 2023], and Falcon-7B [Almazrouei *et al.*, 2023].

Implementation: MeLLo and PokeMQA were limited to 5 hops to keep the experiment time reasonable. Similarly, DeepEdit was allowed 5 additional knowledge candidates. Additionally, they were each allowed a maximum of 200 new tokens to be generated for each LLM call. CHECK used a co-

Dataset	MQuAKE-CF-3k	MQuAKE-T
Model	GPT-J	Size: 6B
FT*	7.70	3.10
ROME*	7.60	4.10
MEMIT*	8.10	10.60
CHECK	**42.27**	**78.69**
Model	Vicuna	Size: 7B
FT*	4.80	23.10
ROME*	8.40	5.00
MEMIT*	7.60	1.70
CHECK	**47.57**	**81.64**
Model	Falcon	Size: 7B
FT*	5.60	17.20
ROME*	1.70	7.30
MEMIT*	2.30	1.60
CHECK	**39.10**	**81.69**

Table 3: Per-case accuracy of compared against parameter-based knowledge editors. Approaches marked with (*) indicate results from a previous work.

sine similarity threshold of 0.8 and was limited to a maximum of 50 new tokens per model call.

Hardware Setup: All experiments were conducted on 1 NVIDIA A100 GPU and 8 CPU cores.

We present two core experiments in the following sections: an evaluation of CHECK against other knowledge editors across 4 datasets and 3 LLMs in Section 5.1, and an ablation study on the performance of CHECK over varying numbers of hops and edits in Section 5.2. Additional experiments are provided in the Appendix.

5.1 Editing Accuracy

The evaluation of CHECK and other state-of-the-art multi-hop knowledge editors is provided in Table 2. Across all models, GMeLLo, MeLLo, DeepEdit, and PokeMQA consistently struggle to achieve a 20% per-case accuracy on three of the four subsets. The only subset they are able to find better performance on is MQuAKE-T, which contains the least number of hops and edits per question. This is unsurprising as they all rely on LLMs doing large amounts of reasoning at once. DeepEdit is able to break the 20% per-case accuracy mark in a few of the results and PokeMQA overpeforms on Vicuna when compared to its performance on GPT-J and Falcon. CHECK does not share this struggle of breaking 20% per-case accuracy, achieving a 31.57%, 28.51%, 24.79%, and 16.77% increase in accuracy over the next highest on the MQuAKE-CF-3k, MQuAKE-2002, MQuAKE-Hard, and MQuAKE-T subsets respectively.

Parameter-based knowledge editors have proven unsuccessful on mutli-hop questions. An evaluation of the accuracy of CHECK and other parameter-based knowledge editors on MQuAKE subsets is provided in Table 3. The accuracies are from [Shi *et al.*, 2024]. CHECK outperforming the parameter-based editors is in line with previous works, further affirming that storage-based knowledge editors are better able to handle the intermediate reasoning steps required for MQA than parameter-based knowledge editors.

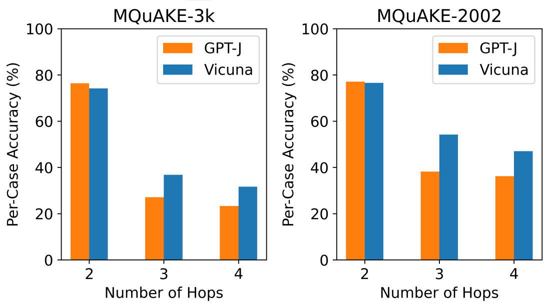

Figure 4: The accuracy of CHECK on the MQuAKE-3k and MQuAKE-2002 datasets across different numbers of question hops.

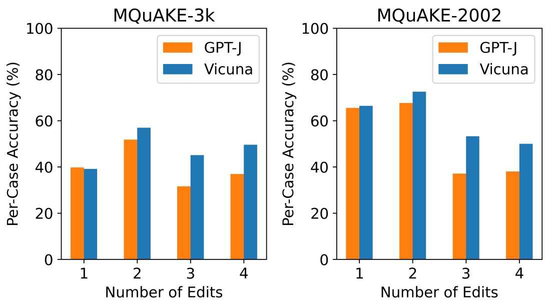

Figure 5: The accuracy of CHECK on the MQuAKE-3k and MQuAKE-2002 datasets across different numbers of edits per multi-hop question.

5.2 MQA Hop and Edit Ablation

We provide a breakdown of CHECK's per-case accuracy on MQuAKE-3k and MQuAKE-2002 over the number of hops and edits in Figure 4 and Figure 5, respectively. As the the number of hops increases, the accuracy of CHECK decreases. This is an expected outcome since longer multi-hop questions require longer relationship chains, introducing more areas for both question decomposition and subquestion resolution to fail. Similarly, as the number of edits increases, the accuracy also drops. Greater number of edits correspond to longer questions, which are more difficult to correctly answer. CHECK's over-performance on questions with 2 edits can be attributed to the in-context learning prompt used during question decomposition.

6 Conclusion

We present the CHECK framework for multi-hop knowledge editing. The main insight of CHECK is that the LLM subquestion reasoning process can be checked for semantic consistency. CHECK decomposes multi-hop questions into a chain of relationships and ensures the semantic consistency of that chain. The chain is then iteratively traversed, answering each of the subquestions that make up the chain and inserting edits where necessary until the answer to the multi-hop question is reached.

Acknowledgements

This material is in part sponsored by UF startup funds and DARPA under agreement number FA8750-23-2-0501. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

[Almazrouei *et al.*, 2023] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of open language models, 2023.

[Ayoola *et al.*, 2022] Tom Ayoola, Shubhi Tyagi, Joseph Fisher, Christos Christodoulopoulos, and Andrea Pierleoni. ReFinED: An efficient zero-shot-capable approach to end-to-end entity linking. In *NAACL*, 2022.

[Baldazzi *et al.*, 2023] Teodoro Baldazzi, Luigi Bellomarini, Stefano Ceri, Andrea Colombo, Andrea Gentili, and Emanuel Sallinger. Fine-tuning large enterprise language models via ontological reasoning. In Anna Fensel, Ana Ozaki, Dumitru Roman, and Ahmet Soylu, editors, *Rules and Reasoning*, pages 86–94, Cham, 2023. Springer Nature Switzerland.

[Brown *et al.*, 2020] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[Chen *et al.*, 2024] Ruirui Chen, Weifeng Jiang, Chengwei Qin, Ishaan Singh Rawal, Cheston Tan, Dongkyu Choi, Bo Xiong, and Bo Ai. Llm-based multi-hop question answering with knowledge graph integration in evolving environments, 2024.

[Cheng *et al.*, 2024] Keyuan Cheng, Gang Lin, Haoyang Fei, Yuxuan Zhai, Lu Yu, Muhammad Asif Ali, Lijie Hu, and Di Wang. Multi-hop question answering under temporal knowledge editing. In *First Conference on Language Modeling*, 2024.

[Chiang *et al.*, 2023] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[Faiz *et al.*, 2024] Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Chukwunyere Osi, Prateek Sharma, Fan Chen, and Lei Jiang. LLMCarbon: Modeling the end-to-end carbon footprint of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[Gu *et al.*, 2024] Hengrui Gu, Kaixiong Zhou, Xiaotian Han, Ninghao Liu, Ruobing Wang, and Xin Wang. PokeMQA: Programmable knowledge editing for multi-hop question answering. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8069–8083, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[Gupta *et al.*, 2023] Anshita Gupta, Debanjan Mondal, Akshay Krishna Sheshadri, Wenlong Zhao, Xiang Lorraine Li, Sarah Wiegreffe, and Niket Tandon. Editing common sense in transformers. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[Gupta *et al.*, 2024] Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. Model editing at scale leads to gradual and catastrophic forgetting. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 15202–15232, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics.

[Hase *et al.*, 2023] Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. Methods for measuring, updating, and visualizing factual beliefs in language models. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2714–2731, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[Izacard *et al.*, 2021] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning, 2021.

[Li *et al.*, 2023] Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Xuying Meng, Siqi Fan, Peng Han, Jing Li, Li Du, Bowen Qin, Zheng Zhang, Aixin Sun, and Yequan Wang. Flm-101b: An open llm and how to train it with $100k budget, 2023.

[Madaan *et al.*, 2022] Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. Memory-assisted prompt editing to improve GPT-3 after deployment. In *ACL 2022 Workshop on Commonsense Representation and Reasoning*, 2022.

[Meng *et al.*, 2022] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 35, 2022.

[Meng *et al.*, 2023] Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023.

[Mitchell *et al.*, 2022] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Memory-based model editing at scale. In *International Conference on Machine Learning*, 2022.

[Shi *et al.*, 2024] Yucheng Shi, Qiaoyu Tan, Xuansheng Wu, Shaochen Zhong, Kaixiong Zhou, and Ninghao Liu. Retrieval-enhanced knowledge editing in language models for multi-hop question answering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 2056–2066, New York, NY, USA, 2024. Association for Computing Machinery.

[Wang and Komatsuzaki, 2021] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[Wang *et al.*, 2024a] Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[Wang *et al.*, 2024b] Yiwei Wang, Muhao Chen, Nanyun Peng, and Kai-Wei Chang. Deepedit: Knowledge editing as decoding with constraints, 2024.

[Yu *et al.*, 2023] Charles Yu, Sullam Jeoung, Anish Kasi, Pengfei Yu, and Heng Ji. Unlearning bias in language models by partitioning gradients. In *Proc. The 61st Annual Meeting of the Association for Computational Linguistics (ACL2023) Findings*, 2023.

[Yu *et al.*, 2024] Lang Yu, Qin Chen, Jie Zhou, and Liang He. MELO: enhancing model editing with neuron-indexed dynamic lora. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19449–19457. AAAI Press, 2024.

[Zhong *et al.*, 2023] Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[Zhu *et al.*, 2024] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, Mexico City, Mexico, June 2024. Association for Computational Linguistics.