Across Images and Graphs for Question Answering

Zhenyu Wen[1,2,6,*], Jiaxu Qian[1,*], Bin Qian[3], Qin Yuan[4], Jianbin Qin[5], Qi Xuan [1,2,#], Ye Yuan[4]
[1]Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, China
[2]Binjiang Cyberspace Security Institute of ZJUT, Hangzhou, China
[3]School of Computing, Newcastle University, Newcastle Upon Tyne, UK
[4]School of Computer, Science and Technology, Beijing Institute of Technology, Beijing, China
[5]School of Computer, Shenzhen University, Shenzhen, China
[6] Department of Electronic Engineering and Information Science, University of Science and Technology of China

***Abstract*—Cross-source query serves as a proxy for scene understanding to support many web applications such as recommendation systems, e-commerce, and e-learning applications. In this paper, we propose SVQA that *semantically combines the knowledge from available images and graphs to answer the complex question.* To this end, we design a graph-based method to unify various data sources into one representation. We then develop a complex question parse method that utilizes the structure of languages to transform the query into a query graph. A graph query engine that performs the query graph over the unified data source while optimizing the query process. To evaluate the proposed system, we build a vanilla dataset called MVQA and show that the state-of-the-art (SOTA) VQA models fail to perform our task. The comprehensive evaluations show that the proposed SVQA is able to reason implicit relationships over multiple images and external knowledge to correctly answer a complex query. We hope that our first attempt provides researchers with a fresh taste of multimodal data analysis.**

I. Introduction

Cross-source question answering is becoming essential in modern online services, which require extracting valuable information from multiple data sources [1]. This paper considers a query Q that operates on a set of images $\mathbb{I}$ and a graph G to realize complex analytics. This kind of analytics becomes essential. For example, a data lake application often requires correlating and integrating the data from different sources that have various formats [2]. However, correlating and interacting data in images and graphs is challenging. The images are represented as a collection of pixels and the graph is composed of vertices V and edges E. How to semantically link the images and graphs is an uncovered problem in the data management community.

Example 1. *Consider an online analytics service provider that has various data sources, as shown in Figure 1. For example, it contains images $\mathbb{I}$ of movies, and a graph G that describes the relationship between characters in movies. Then, we consider answering the following query.*

Example query Q. What kind of clothes are worn by the wizard who is most frequently hanging out with Harry Potter's girlfriend?

Intuitively, to answer Q with SVQA framework requires the following steps: 1) Q is decoupled into three sub-queries, namely $\{ q_1, q_2, q_3 \}$, for the following reasons: q_1 is the main clause in Q that requires querying types of clothes. q_2, q_3 are two conditions of Q, where q_2 is the reference of q_1 and q_3 is the constraint of q_2. 2) Apart from the query decomposition, we need to interact with the $\mathbb{I}$ and G to find the correct answer. For example, the information about Harry Potter's girlfriends (i.e., Ginny Weasley and Cho Chang) is available on G. Moreover, 3) the information that the wizard appearing with Ginny Weasley and Cho Chang needs to be discovered from $\mathbb{I}$, and from these traversed images we can count the cloth that the wizard is wearing most frequently.

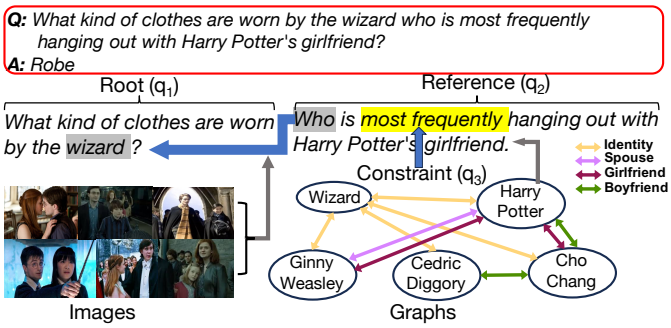


Fig. 1: An example of complex question answering.

Most VQA research [3], [4], [5], [6] focus on improving the ability to understand questions and images through deep learning technologies such as RNN [7] or Transformer [8]. These methods do not consider incorporating the content of $I \in \mathbb{I}$ and G. More recent research [9], [10] has started to look at using graph G to answer the questions to overcome the issue that the contents of an image are not sufficient to answer. Large models, such as ChatGPT [11], Claude [12], are parametric knowledge systems that function as a black box [13], with a primary focus on enhancing the processing of text, images, and multi-modal conversations. SAM [14] and OVSeg [15] excel in their image understanding capabilities, enabling them to recognize and label all instances within an image. SEEM [16] is designed to process multi-modal data, such as texts and images, but the model is still focused on classical computer vision tasks i.e., object segmentation and recognition. Although the large models have impressive text and visual comprehension and producing abilities, they lack the ability to reason the implicit information across images or texts and images. In future work, to achieve better knowledge

(entities) extracting from both images and texts we can interact SVQA with the large models.

Our contributions. In this paper, we go one step forward and propose a new query-answering framework SVQA to tackle the challenge as shown in example 1. Answering this type of question requires complementing the implicit relations among images and further performing cross-image reasoning, while interacting with a merged graph. To address the challenge, we make the following contributions to this paper.

(1) Question answering framework (§II). SVQA converts $I \in \mathbb{I}$ to a scene graph $G_{sg}(I)$ to obtain the entities of I, and then combine G_{sg} with graphs G as a merged graph G_{mg}. Next, a complex query Q is decomposed into a query graph G_q that is a set of simple and ordered sub-queries Q_{sub}. Finally, the G_q is optimized and can be executed over G_{mg}.

(2) Data aggregator (§III). To improve the efficiency of acquiring G_{mg}, we develop a sub-graph caching mechanism that caches the most frequently sub-graphs while linking entities between the $G_{sg}(I)$ and G.

(3) Query graph generator (§IV). We develop a simple and efficient query decomposition algorithm based on computational linguistics for decomposing complex query Q into a hierarchical query graph G_q. Unlike the deep learning model-based method (e.g., Large language model), our algorithm does not require the use of labeled data for model tuning.

(4) Query executor(§V). Based on the generated query graph, we design an iterative strategy for searching the database and aggregating the returned results of each node from the query graph. We show that the time complexity of our algorithm is $O(N(|V/2| \cdot |V/2|))$. We parallelize our algorithm to further improve its performance of the algorithm.

(5) A new VQA task and dataset MVQA (§VI). We constructed a new dataset called MQVA to assist the task understanding and solution proposal. The whole image dataset contains 4,233 images selected from COCO, and we manually create 100 complex $\langle question, answer \rangle$ pairs to evaluate the proposed solution.

II. Problem definition and SVQA framwork

Multimodal data. In this paper we consider performing a complex query over a set of heterogeneous data sources, such as graphs and multimedia data (e.g., videos, images).

Graph. A directed labeled graph is defined by $G = (V, E, L)$, where V is a set vertices, and E is a set of edges linking the vertices. $L(v)$ and $L(e)$ represent the label of $v \in V$ and $e \in E$.

Images. An image I captures the scene that might include various elements such as objects, people, landscapes, or other details that contribute to the overall understanding or interpretation of the image. The video data is the collection of I, i.e., $\bigcup_{i=1}^{n} I_i$.

Complex Query. A complex query is composed of multiple clauses, denoted by $Q = \{c^1, ..., c^n\}$. Each c^i corresponds to a SPOC. The SPOC is a quadruple abstract structure whose subject, predict, object, and constraint are denoted by v_s, v_p, v_o, and v_c corresponding to c_s, c_p, c_o, and c_c, respectively.

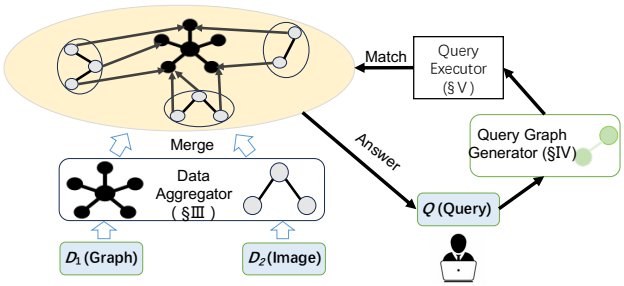


Fig. 2: SVQA framework

Problem. Given a complex query Q described in natural language, a set of related images $\mathbb{I}$, and graph G, the objective of this paper is to propose a framework that can integrate semantic information from both $\mathbb{I}$ and G to return the correct answer to Q.

Our SVQA framework is based on the following assumptions.

Assumption 1. The images $\mathbb{I}$ and the graph G are query-independent.

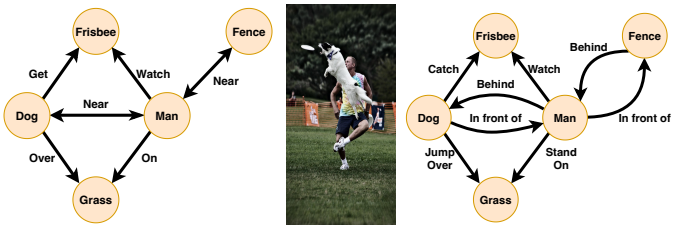
This differs from previous VQA approaches [17], [18], [19] (i.e., query-dependent scenarios) in which an image i and question q are paired and fed into a deep learning model. For example, given an i that shows a man wearing a hat and the q is "What's the color of the hat that the man is wearing?" Our proposed framework can achieve the same performance as the representative work [20] if the scene graph generation model obtains sufficient semantic information from I. Also, we can use a small set of questions Q to fine-tune the scene graph generation model M to generate a more specific scene graph for given images I, if the pre-trained M can't obtain the semantic information from I to answer the questions Q.

Assumption 2. The contents contained within the images $\mathbb{I}$, as well as the graph G, possess the necessary knowledge to answer the given questions.

Framework. Figure 2 shows SVQA framework to tackle the above-mentioned problem. First, in order to more efficiently query (or analyze) multimodal data (e.g., images and graphs), we propose a data aggregation method (Data Aggregator §III) to unify data of different modalities into a merged graph. Next, to answer a complex query, we decompose the query into an ordered query graph consisting of multiple sub-queries. Executing the sub-queries sequentially over the merged graph, we can reason the implicit information and obtain the correct answer (Query Graph Generator §IV). Finally, to execute the generated query graph over **BIG** merged graph, we develop a sub-graph search method and its optimization mechanism to improve the query efficiency (Query Executor §V).

III. Data Aggregator

We first present how to construct a *merged graph* G_{mg} via connecting images $\mathbb{I}$ and graphs G. Given an image $I \in \mathbb{I}$, we first convert it into a structured scene graph G_{sg}, representing all detected objects and their relationship linkages. We then construct a *merged graph* G_{mg} by integrating all generated



(a) Biased Scene Graph (b) Raw Image (c) Unbiased Scene Graph

Fig. 3: Example of Scene Graph Generation in SVQA

scene graphs $\{G_{sg}(I)|\forall I \in \mathbb{I}\}$ to the graph G, where V and E represents the nodes and links of G, respectively.

A. Scene Graph Generation

A scene graph $G_{sg}(I)$ consists of a set of objects $V_{sg}(I)$ and a relational matrix $E_{sg}(I)$, denoted by $G_{sg} = (V_{sg}, E_{sg})$ [21], where V_{sg} represents all detected objects in I, and E_{sg} represents relationship linkages between all pairs of V_{sg}.

Each object $v_i \in V_{sg}(I)$ is composed of a bounding box $b_i \in \mathbb{R}^4$, a feature map m_i and a class label $l_i \in \mathbb{R}$, denoted by $v_i = (b_i, m_i, l_i)$. The links between v_i and v_j is written as $r_{ij} \in E_{sg}(I)$.

Object Detection. To extract objects from an image I, we present an object detection method based on Mask R-CNN [22]. Let N_I be a set of objects obtained from I. The objects $\{v_i|\forall i \in N_I\}$ for I are extracted in the following two steps: 1) generating bounding boxes from the images. 2) classifying objects within the bounding boxes.

(1) *Bounding Boxes and Feature Maps.* Let $B = \{b_i|\forall i \in N_I\}$ and $M = \{m_i|\forall i \in N_I\}$ be candidate bounding boxes and feature maps generated by Mask R-CNN. Each bounding box b_i is a tuple (x_i, y_i, w_i, h_i), where (x_i, y_i) are the coordinates of the top-left corner of the bounding box, (w_i, h_i) are the width and height of the bounding box. Region proposal network (RPN) is a module within the Mask R-CNN to generate feature maps m_i for each b_i.

(2) *Object labels.* L are the predicted class labels for the bounding boxes and l_i is the label for each b_i.

Linkage Generation. Linkages are represented by a relational matrix that indicates the relationship $r_{ij} \in E_{sg}$ between two detected objects v_i and v_j, denoted as $E_{sg}(I) = (r_{ij})_{1\le i,j\le N_I}$. Due to the training bias, which may lead to inaccurate relationship expressions for two objects, we develop our linkage generation into two steps: 1) generating initial links. 2) removing bias from links.

(1) *Generating Initial Links.* The methodology for generating links between objects v_i and v_j can be expressed as:

$$\{(b_i, m_i, l_i), (b_j, m_j, l_j)\} \rightarrow \{p_{r_{ij}}\} \quad (1)$$

The input consists of the bounding boxes $B = \{b_i, b_j|i \neq j\}$, feature maps $M = \{m_i, m_j\}$, and their object labels $L = \{l_i, l_j\}$. $p_{r_{ij}}$ represents the probability that the relationship between v_i and v_j belongs to class E_{sg}. We utilize an RNN-based MOTIFNET model [23] for extracting $p_{r_{ij}}$.

(2) *Removing Bias from Links.* When inferring r_{ij} from v_i and v_j, the explicit relationship between the objects may be obscured by the ubiquitous relationships that exist within the l_i and l_j. Such a training bias thus needs to be deducted from $p_{r_{ij}}$. We infer such bias with the previously trained MOFITNET as follows:

$$\{(b_i, Mask(m_i), l_i), (b_j, Mask(m_j), l_j)\} \rightarrow \{p'_{r_{ij}}\} \quad (2)$$

We mask the feature maps m_i and m_j and set them as zero vectors. Then we follow the same procedure as in Equation 1 to predict the relationship probability $p'_{r_{ij}}$ between bounding boxes b_i and b_j.

Finally, we get the unbiased relationships between v_i and v_j with the following equation:

$$r_{ij} = argmax(p_{r_{ij}} - p'_{r_{ij}}) \quad (3)$$

Specifically, after $p'_{r_{ij}}$ from $p_{r_{ij}}$, the relationship r_{ij} with the highest probability is the most accurate relationship between v_i and v_j. Incorporating such Total Direct Effect (TDE) [24] mechanism helps to mitigate bias and improve the accuracy of our generated scene graphs.

Example 2 *An example of scene graph generation is shown in Figure 3, where the raw image in Figure 3(b) shows a dog jumping over the grass to catch a frisbee, while a man watching from behind. The initial links generated between objects are shown in Figure 3(a) where we see many obscure predicates like* on *and* near *in Figure 3(a), i.e.,* $\{$*Dog, near, Man*$\}$*. However, with TDE-based linkage generation, the implicit relationship between bounding boxes is deduced and we see in Figure 3(c), explicit relationship descriptions between objects, i.e.,* $\{$*Dog, in front of, Man*$\}$*;* $\{$*Man, behind, Dog*$\}$*.*

B. Graph Merging

To obtain G_{mg}, we link the generated $\{G_{sg}(I)$ with the given graph G, by connecting the vertices $V_{sg}(I)$ with their corresponding vertices in V. The aggregation requires expensive time and memory overhead. Therefore, we have developed a caching mechanism to speed up graph merging. Before introducing the mechanism, we first provide the formal definitions.

Definition 1: *K-th order neighbours of a vertex t are defined as the vertices that can be reached from the vertex t in K hops.*

Definition 2: *Let* $S(t, k)$ *be the vertices of k-th order neighbors of a vertex t,* $G[S(t, k)]$ *represents the induced subgraph of* $S(t, k)$ *extracted from G.*

Example 3 *Consider the graph depicted as G in Figure 3(a), a vertex "Fence" is the target t node, and its K-th (e.g., K=1) order neighbors is "man". Then, a set* $S(\text{“}Fence\text{”}, 1)$ *indicates the 1-th order neighbors of "Fence", which consists of the vertices "Fence" and "Man". Therefore,* $S(\text{“}Fence\text{”}, 1)$ *can construe a subgraph* $G[S(\text{“}Fence\text{”}, 1)]$ *including two directed edges ("Fence"* $\rightarrow$ *"Man") and ("Man"* $\rightarrow$ *"Fence").*

Key idea. The key idea of the graph merging algorithm is to cache a frequently visited subgraph to reduce the complexity of linking the entities between G_{sg} and G. To this end, we first analyze the G_{sg} to obtain a set of categories, denoted as $t \in T$, that frequently appear. We then construct

Algorithm 1: Graph Attachment Method for Aligning Scene Graphs and Graph

Input : Scene graphs $\{G_{sg}(I) = (V_{sg}(I), E_{sg}(I))|\forall I \in \mathbb{I}\}$, graph $G = (V, E)$, frequency threshold c', subgraph generation threshold k,

Output: merged graph G_{mg}

1 **Initialize** $T \leftarrow []$, $G_{\mathcal{N}} \leftarrow []$ /* Initial Stage */
2 $T \leftarrow statistics(\{G_{sg}(I)|\forall I \in \mathbb{I}\})$ // Count image category
3 **for** *(count c, category* t_{sg}*) in* T **do**
4 **if** $c > c'$ **then** // Generate subgraphs
5 vertex $t \leftarrow find(t_{sg}, V)$
6 $G[S(t,k)] \leftarrow subgraph(t, k, G)$
7 $G_{\mathcal{N}} \leftarrow G[S(t,k)]$
8 **for** v *in* $\{V_{sg}|\forall I \in \mathbb{I}\}$ **do** /* Attach Stage */
9 **for** $v\prime$ *in* $G[S(t,k)] \in G_{\mathcal{N}}$ **do**
10 **if** $v == v\prime$ **then**
11 $connect(v, v\prime)$ // Link the vertices
12 **else**
13 $v = Query(v, G)$;
14 $connect(v, v)$
15 $G_{mg} = G$
16 **return** G_{mg}

subgraphs of the category $G[S(t,k)]$ from G. Subsequently, the $G[S(t,k)]$ performs as the cache while linking the G_{sg} with G. Algorithm 1 summarizes the workflow of the graph merging algorithm.

(1) Initial Stage (line 1-7). $G[S(t,k)]$ is extracted for vertex t that may frequently be used during the alignment. For all $\{V_{sg}(I)|\forall I \in \mathbb{I}\}$, we count the occurrence frequency of types of $V_{sg}(I)$ and sort them in descending order, denoted by T (line 2). Then, for all categories $t_{sg} \in T$ in order, we generate $G[S(t,k)]$ by 1) searching for corresponding vertices t in V by the category t_{sg} (line 5), 2) starting from $t \in V$, traversing through the whole G and extracting all vertices and links that are within K hops away from t (line 6). All acquired $G[S(t,k)] = (V[S(t,k)], E[S(t,k)])$ are appended to a cache list $G_{\mathcal{N}}$ in descending order as well (line 7).

(2) Attach Stage (line 8-16). With $\{G[S(t,k)]|\forall t \in T\}$, we now proceed to the scene graph aggregation process. Recall in Section III-A, we have generated multiple scene graphs $\{G_{sg}(I) = (V_{sg}(I), E_{sg}(I))|\forall I \in \mathbb{I}\}$. For each vertex $\{v \in V_{sg}(I)|\forall I \in \mathbb{I}\}$, we perform a traversal through $G_g[S(t,k)] = (V_g[S(t,k)], E_g[S(t,k)])$ and link v with the corresponding vertex v' in $V_g[S(t,k)]$ (lines 10-11). In the case of rare vertices v in V_{sg}, where we cannot find a corresponding v' in $V_g[S(t,k)]$, we directly query G from storage to find corresponding vertices in V_g and link them (lines 13-14).

Based on our observations in MVQA, detailed in §V, we set $k = 2$, generate subgraphs $\{G[S(t, k = 2)|\forall t \in T]\}$ for all vertices T that occur more than 5 times. Note that our extraction method for $G[S(t,k)]$ does not store a part of G independently; instead, it adds an index to G to distinguish $G[S(t,k)]$. In MVQA, approximately 58% of vertex types occur more than 5 times, and nearly 82% of vertices are covered in finally generated subgraphs.

Algorithm 2: Complex Query to Query Graph

Input : Complex query Q

Output: Query Graph $G_q = (V_q, E_q)$

1 **Initialize**$G_q = (V_q, E_q) \leftarrow 0$, $C \leftarrow 0$, $Que \leftarrow []$
2 $POS \leftarrow$ `getPOS`(Q) /* Initial Stage */
3 $DT \leftarrow$ `getDependencyTree`(Q)
4 **if** $DT \neq NULL$ **then** /* Parse Stage */
5 $C \leftarrow$ `getClauses` (DT, POS)
6 $V_q \leftarrow C$
7 **for** c *in* C **do**
8 $[c_s, c_p, c_o, c_c] \leftarrow$ `getSPOC`(c)
9 ENQUEUE$(Que, [c_s, c_p, c_o, c_c])$
10 **if** $Que \neq NULL$ **then** /* Connect Stage */
11 $u \leftarrow$DEQUEUE(Que)
12 **for** *element* v *in* V_q **do**
13 **if** `SOOverlap` *(v,u)* **then** // Compare overlapping part of vertex
14
15 $e_g \leftarrow$`SOMatching` (v,u) // record the overlap part
16 ENQUEUE(Que, v)
17 $E_g \leftarrow e_g$
18 **return** G_g

IV. Query Graph Generator

In this section, we develop a language parse method that utilizes the language structure, e.g., non-normal grammar [25], to transform the complex query Q into a query graph G_q.

The ABCD [26] was designed to transform a complex sentence into simpler sentences, each containing only one clause. However, this approach does not align with our specific requirements. In our framework, the query Q needs to be decomposed into a set of SPOC elements, which are then linked together to construct a G_q.

Definition 3: *Query Graph. A* query graph *is a directed graph* $G_q = (V_q, E_q)$*, where each* $v \in V_q$ *is a SPOC of the clause* $c \in C$ *in* Q *and a directed edge* $e \in E_q$ *represents the dependencies between two vertices in* G_q.

A. Method Overview

Algorithm 2 summarizes three main stages of the query graph generation method.

(1) Initial Stage(line 1-3) The method consumes user's query Q, and then parse it into part-of-speech (POS) and a dependency tree(DT) [27], [28] via *Stanford POS Tagger [29]* and *Stanford Parser [28]*. The *Stanford POS Tagger* can be formulated as:

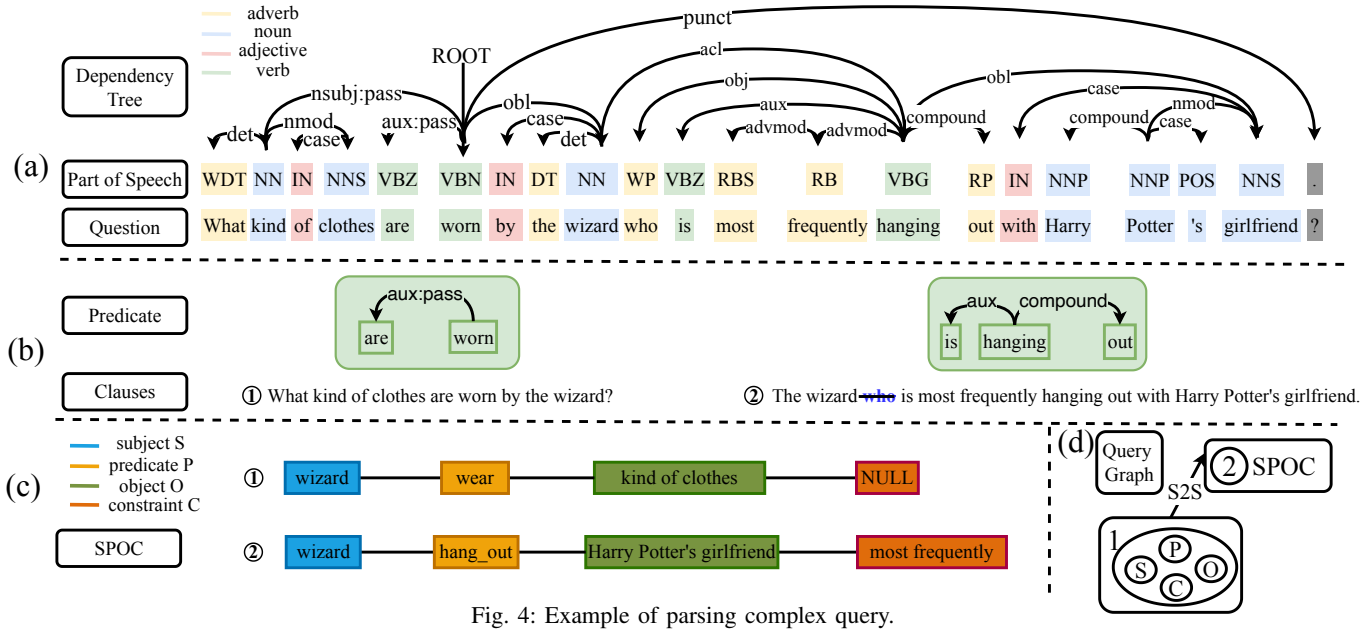


Fig. 4: Example of parsing complex query.

$$POS(x) = \arg\max(\exp\left(\sum_i \lambda_i f_i(x,y)\right)/Z(x)) \quad (4)$$

where λ_i is a weight parameter for feature i, $f_i(x,y)$ is the value of feature i for word x and tag y, and $Z(x)$ is a normalization factor. As for the *Stanford Parser* can be expressed as follows:

$$A_{1:n} = \arg\max_A \sum_{i=1}^{n+1} w(f(S_i, A_i)) + g(S_{i-1}, A_i) \quad (5)$$

Here, S_i represents the i-th state, and A_i represents the action taken at the S_i. The feature vector $f(S_i, A_i)$ represents the feature values associated with performing A_i in S_i. The weight $w(A_i)$ is a value calculated for each possible A_i, which measures the significance of the $f(S_i, A_i)$. $g(S_{i-1}, A_i)$ is a transfer function to calculate the cost from the S_{i-1} to the S_i.

(2) Parse Stage(line 4-9) Based on the POS and its DT, we generate the clause set C (line 5) which deconstructs from Q and the vertices V_q of the G_q. Next, we develop a state machine to extract $[c_s, c_p, c_o, c_c]$ quadruple from the clause c detailed in §IV-B (line 8), where c_s, c_p, c_o are key elements to construct the query graph.

(3) Connect Stage(line 10-16) Afterward, a set of vertices V_q is arranged to create a hierarchical graph denoted as $G_q = (V_q, E_q)$. Each vertex $v \in V_q$ is an extracted SPOC quadruple and we denote it as $v = [c_s, c_p, c_o, c_c]$. The directed edges between the vertices V_q are based on the phrase structure rules for sentences. As a result, G_q is ordered based on the structure of languages to determine the sequence in which each clause c is executed, as explained in more detail in §IV-C.

B. Extracting SPOCs from the Complex Query

Extracting SPOCs from Q involves two steps: 1) Pruning the redundancy information for each c; and 2) Transforming the previously obtained information into a structured SPOC.

According to the outputs of the initial stage, we have the following observation: (1) There are 45 tags produced by *Stanford POS Tagger* which is redundant for segmentation clauses $\{c^1, ..., c^n\}$ from Q. (2) The c_p in modern English can be identified in two cases, namely subject-link verb-predicative (SLVP) structure (i.e. *be* verb such as *is*, *are*, etc.) and verb fixed collocation [30].

Thus, we adopt the following strategies to obtain the SPOCs from Q. (1) We only use 4 tags (i.e., "nouns", "verbs", "adjectives" and "adverbs") out of 45 to split a Q into a set of C by following [31], [32], [33]. (2) To extract the clauses from Q, we first find all the verbs in the sentence and then obtain the words that have the edges with the verbs in the DT. Hence, a predicate c_p is a verb along with its connected words. Upon identifying c_p, we traverse all connected words in DT to locate the nouns that represent the subject (c_s) and object (c_o) of the clause c. The determination of c_s and c_o is depended on the voice of the clause, whether passive or active.

Additionally, we may require cross-sentence references to replenish the hidden elements. In DT, the *acl* means the noun is modified by another c [33], [32]. For example, the *acl* edge connects from *hanging* to *wizard*, and the *hanging* is belonging to the second clause. According to this relation, the pronoun (i.e., "who") in the second clause can be replaced by the noun (i.e., "wizard") in the first clause.

Example 4 *Figure 4(a) shows the generated POS (Part of Speech)* and DT (Dependency Tree) of a given question Q via Stanford POS Tagger and Stanford Parser. Next, there are two c_p (i.e., are worn, is hanging out), which are used to obtain corresponding clause c, while removing the pronoun (i.e., *who*) as shown in Figure 4(b). We use the first clause c in Figure 4(b) as an example to illustrate how we extract the SPOC from the clause c. Starting with the predicate of the first clause's c_p (*are worn*), we use the *nsubj:pass* and *obl* edge to find the first clause's c_s (*kind*) and c_o (*wizard*). Through *nmod* and *case* edges, we obtain the complete c_s (*kind of clothes*). Finally, we change the passive voice (*are*

worn) to simple present(*wear*). The SPOC of the first clause is shown in Figure 4(c).

C. Construct a Query Graph

As we discussed above, G_q is a directed graph that indicates the order of executing sub-queries. To establish the directed edge between clauses, we define the following five types of dependencies, i.e., subject-subject($S2S$), subject-object($S2O$), object-subject($O2S$), object-object($O2O$), and no dependency($NULL$). Based on the defined dependencies, we traverse the V_q and create the edges $e_{ij} \in E_q$ for any v_i and v_j that exhibit the specified dependencies. This process enables us to build a structured query graph that reflects the relationships between sub-queries. It's important to note that the $S2S$ dependency indicates that the subject labels $L(c_s)$ of vertex $v_i \in V_q$ and $L(c_s)$ of vertex $v_j \in V_q$ share the same semantics, which is a key consideration in determining this type of dependency. For example, as depicted in Figure 4(steps (c) and (d)), two vertices share the same subject (v_s), which is "wizard". In such cases, we can create an $S2S$ edge between the two vertices.

We can represent the logical organization of complex queries Q by constructing a graph G_q using the approach described above. It's worth noting that G_q captures both the hierarchical relationships and dependencies between queries, which play a crucial role in guiding the execution order of G_q within the context of G_{mg} in the next section.

V. Executing Query Graph over merged graph

Given one query graph G_q, we obtain the answer by querying the merged graph G_{mg} with G_q. We have three types of questions: counting, reasoning, and judgment questions following [34], corresponding to answers in the form of a number, an entity, and a judgment word (i.e., Yes/No), respectively. Performing the G_q over G_{mg} is not trivial and time-consuming, especially when we query a large number of G_q simultaneously from the G_{mg}. Thus in this section, we introduce a sub-graph matching method for single query execution in §V-A and the optimization mechanism to deal with multiple-query execution in §V-B.

A. Query Graph Executor

The inputs of the Algorithm 3 are the query graph G_q, merged graph G_{mg}, and a set of predefined words [35] for relation matching, and the output is the generated answers. The QueryGraphExecutor processes all the vertices in G_q according to the specified dependency relationships, querying relevant features for each vertex and supporting the query of the next vertex until reaching the end of the graph G_q.

We first initialize an empty *Queue* for storing all temporary results during the execution process (line 1). we also obtain labels of all edges T in G_{mg} (line 2), which will be used for relation matching later. Upon analyzing vertices in G_q, we obtain all the vertices v with 0 in-degree and regard them as the starting vertices, pushing them into the *Queue*, waiting to be processed. (line 3-4)

Algorithm 3: QueryGraphExecutor

Input : query graph $G_q = (V_q, E_q)$, merged graph $G_{mg} = (V_{mg}, E_{mg})$, predefined word set $\mathbb{S}$
Output: Answer *ans*

1 $Queue \leftarrow []$
2 $T \leftarrow$ getLabels(E_{mg})
// label statistics in E_{mg}
3 $v \leftarrow getStartVertices(G_q)$
4 $Queue$.put(v)
5 **while** $Queue \neq []$ **do**
6 $u = [c_s, c_p, c_o, c_c] \leftarrow Queue$.pop()
/* Query Stage */
7 $RP \leftarrow$ getRelationpairs(u, G_{mg})
// all relation pairs connecting u in G_{mg}
8 $P \leftarrow$ maxScore($L(c_p)$, T)
// most similar label of c_p in E_{mg}
9 $Con \leftarrow$ maxScore($L(c_c)$, $\mathbb{S}$)
// c_c's most similar keywords in predefined word set $\mathbb{S}$
10 $AP \leftarrow$ filter(RP, P, Con)
// filter in RP with P and Con
/* Update Stage */
11 $S(u, 1) \leftarrow$ getNeighbors(u, G_q)
// 1-hop neighbors of u in G_q
12 **forall** *vertex* $v' \in S(u, 1)$ **do**
13 $e \leftarrow$ getEdge(v', u)
14 **If** $L(e) == O2O$ **then** Replace($v'.c_o$, $AP.Obj$)
15 **elif** $L(e) == O2S$ **then** Replace($v'.c_o$, $AP.Sub$)
16 **elif** $L(e) == S2O$ **then** Replace($v'.c_s$, $AP.Obj$)
17 **elif** $L(e) == S2S$ **then** Replace($v'.c_s$, $AP.Sub$)
// replace Sub or Obj of AP with c_s or c_o of v'
18 $Queue$.put(v')
19 $ans \leftarrow$ getFinalanswer(u, RP)
20 **return** *ans*
21 **getRelationpairs** *(u, G_{mg})* **:**
22 $[c_s, c_p, c_o, c_c] = u$
23 $Sub \leftarrow$ matchVertex($L(c_s)$, G_{mg})
// find vertices with c_s in G_{mg}
24 $Obj \leftarrow$ matchVertex($L(c_o)$, G_{mg})
25 $E_{so} \leftarrow$ getRelations (Sub, Obj)
// get relationships between Sub and Obj
26 **return** $RP \leftarrow (Sub - E_{so} - Obj)$

Query Stage: While *Queue* is not empty, we pop a vertex u from the *Queue*, and query from G_{mg} for all relevant relation pairs, via *getRelationpairs* function (line 6-7). The obtained RP contains potential answer sets $(Sub - E_{so} - Obj)$ for the current vertex u. Note any u is represented by $[c_s, c_p, c_o, c_c]$, containing information for filtering the answer from RP. In order to do so, we first use the *maxScore*, which allows us to obtain the label $P \in T$ that bears the closest resemblance

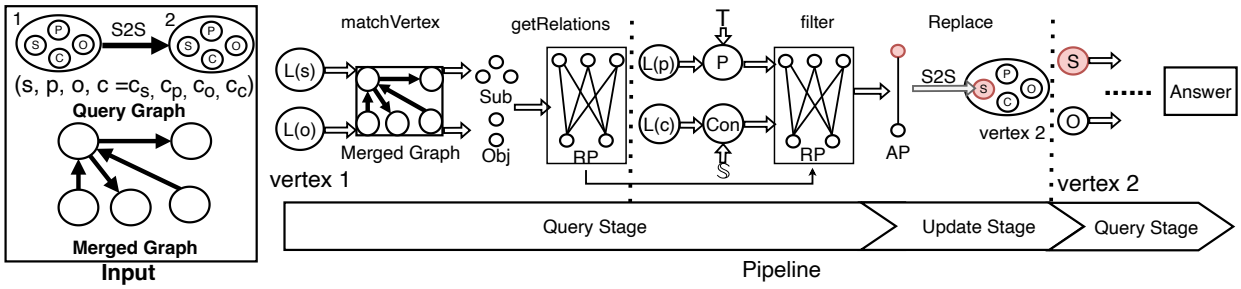


Fig. 5: Example of executing query graph with 2 vertices

to c_p (line 8). *maxScore* works by converting the inputs to embeddings [36] and filtering out the most similar type based on cosine similarity. We also locate the most similar keyword $Con \in \mathbb{S}$ of the constraint c_c via *maxScore*. (line 9). When we have obtained the relation pair RP, the label P, and the constraint Con, we use *filter* to extract the relation subset AP from RP by applying conditions P and Con (line 10).

Update Stage: In this stage, we update the answers queried for vertex u to all its neighboring vertices $S(u, 1)$ in the query graph G_q. For each neighboring vertex $v' \in S(u, 1)$, we get the edges e between the u and v' (line 13). Then we replace either c_s or c_o of the v' with the ans, depending on the properties indicated by e (line 14-17). We then push the updated v' to the $Queue$ to wait for the querying process (line 18). The algorithm terminates when the last vertex in G is processed, and the final answer of G is obtained via *getFinalanswer* function (line 19). The *getFinalanswer* function utilizes u to determine the query type: counting, reasoning, or judgment query. The corresponding answer is generated as well.

The **getRelationpairs** function finds the vertices in G_{mg} with the same subject c_s and object c_o as u through the *matchVertex* function and returns as the Sub and Obj (line 21-22). With the Sub and Obj, we can obtain the corresponding edges RP between them (line 18). The *matchVertex* function uses the Levenshtein Distance (LD) [37] to find $v \in V_{mg}$ whose distance is less than the empirical threshold. As for the input of the *matchVertex* function: subject c_s or object c_o, we check if they are simple nouns. If not, the function obtains its main noun as the input of the LD. For non-simple nouns, the function continues to obtain other parts except the main noun, uses cosine similarity [38] to calculate the similarity between them and the labels of the edges in T, and selects the most similar label. Then, starting from the main noun, the function follows the edge whose label is specified above to obtain the corresponding set V_{mg} and returns it as the Sub and Obj.

Example 5 *We show an example in Figure 5 for executing the generated query graph for the question in Figure 4, which contains two vertices 1 and 2 and relation S2S. For simplicity, we use the letters s, p, o, and c to represent the c_s, c_p, c_o, and c_c of the vertex The* QueryGraphExecutor *begins by starting from vertex 1 to generate a set of relation pairs that correspond to the SPOC semantics in the merged graph. It then matches the corresponding vertices Sub and Obj in G_{mg} of the labels of c_s and c_o from vertex 1 in the* matchVertex. *Once Sub and Obj are obtained, the relation pairs RP of them are generated and filtered with c_p and c_c to obtain the subset pairs AP in the* getRelationpairs. *Subsequently, we update c_s of vertex 2 (colored in pink) with the AP.Sub provided by vertex 1 (also colored in pink), as well as the attribute S2S of the edge e^{12}. We then proceed to query vertex 2 to obtain the final answer for the entire G_q.*

Complexity Analysis: In the worst-case scenario, answering a complex query Q using Algorithm 3 may take cubic time. Specifically, answering a vertex $v \in V_q$ takes $O(|V|)$ time to find v_s and v_o, and $O(|V/2| \cdot |V/2|)$ time to select corresponding relationships in the merged graph G_{mg}. Moreover, a complex query Q can be logically divided into multiple clauses $c_i \in C$, where $v_i \in V_q$ ($i \in [1...N]$). Therefore, it can take up to $O(N(|V/2| \cdot |V/2|)) = O(N(|V|^2/4))$ time to answer the complex query Q.

B. Executor Optimization with Multiple Queries

When receiving a set of N questions, a simple approach is to process them in the order they were received, by putting them in a queue L, where $L < N$. The straightforward approach described above may result in duplicate queries of the same question, which can lead to unnecessary resource usage and waste. Therefore, it is necessary to schedule the processing of the N questions to prevent resource waste and speed up the response time. We propose a *Key-Centric Caching* mechanism and an *Optimized Queries Scheduling* mechanism to speed up the whole query process.

Key-Centric Caching Mechanism: Considering the time-consuming nature of both *matchVertex* and *getRelations* operations, we develop a caching mechanism in the *getRelationpairs* function. Specifically, *matchVertex* requires to compare with all the labels of V_{mg} to obtain the corresponding vertex set Sub and Obj, and we named it as "scope". The *getRelationpairs* needs to traverse all neighbors for the input vertices v and v', and further to determine whether the input vertices are relation endpoints, so that all relation pairs RP are returned, and we named it as "path".

We further optimize our caching strategy by employing the Least Frequently Used [39] (written by LFU) algorithm. This algorithm dynamically shuffles the least referenced item from the cache, simultaneously storing the frequency of the remaining items. By introducing LFU algorithm, our caching mechanism significantly improves the response speed of the matching process.

Optimized Queries Scheduling: Before executing N query graphs on G_{mg}, we conduct a pre-analysis for each vertex v

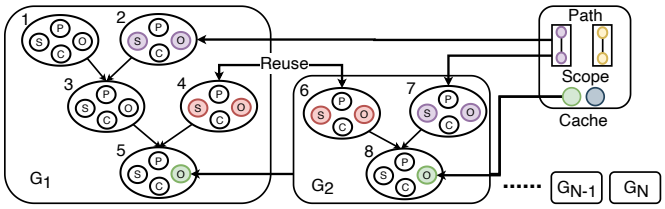


Fig. 6: Optimized query scheduler with key-centric cache.

in G_q for potential reuse of each $[c_s, c_p, c_o, c_c]$ in the vertex v. Specifically, a verification is performed to determine for each v if there exists a similar c_s and c_o pair in *path*. If there is no cache available for the pair, we check whether there is a cache in *scope* for finding the vertices in G_{mg} of $L(c_s)$ and $L(c_o)$, and utilize it if available. If no cache is found in both, the new vertex information is added to both *path* and *scope*.

Once we have pre-analyzed the N query graphs, we construct a dictionary to record the unique vertices and their frequency of appearance. We also attach a frequency ratio to each vertex based on its frequency of appearance. For each query graph G_q, we calculate the sum of the frequency ratios of each vertex within it to obtain a score for G_q. This score is used to sort all the query graphs in descending order. Sorting the query graphs in descending order ensures that the query graphs containing more frequently used vertices are processed first, which helps to reduce the overall processing latency. This approach is particularly effective when there is limited memory available for query execution. By following these steps, we can efficiently handle multiple complex queries, and reduce the overall response time of SVQA, and we validated within the Exp-2 VII.

Example 6 *Figure 6 shows an example of executing N query graphs. We first analyze all the N query graphs to get the frequency ratio of each and sort them in order. We see that* G_1 *contains the most frequent vertices and contains more vertices than* G_2*, thus is processed first. Then during querying for* G_1*, we cache all vertices in* path *and* scope*, with vertex 2, 4, and 5 being re-used by* G_2 *during the process. Thus when querying answers for* G_2*, most information could be directly retrieved from the cache, saving a huge amount of time.*

VI. MVQA Dataset

In this section, we analyze the limitations of current VQA datasets, and build MVQA to evaluate our framework.

A. Limitation of existing VQA dataset

Most VQA [19], [42], [17] tasks focus on improving the performance of answering queries from **one image**, which mostly contain questions such as "What sort of vehicle uses this item?". In line with such research, many datasets have been proposed, i.e., DAQUAR [40] for indoor images of human question-answer pairs; Visual7W [41] created questions more closely aligned with natural language; The VQAv2 [18] extends VQAv1 [17] with complementary images and creates more balanced question-answer pairs.

The aforementioned datasets are relatively simple and do not consider the **external information** of the entities contained in the image. KB-VQA [42] first attempts to understand the image with given external knowledge. Expanding the KB-VQA, FVQA [19] supports answers with <image, question, answer>. Subsequently, OK-VQA [34] is based on open knowledge instead of a given knowledge. Extra external knowledge is required to correctly answer the questions in these tasks. Table I summarizes representative datasets for VQA tasks, ranging from knowledge-based to non-knowledge-based tasks. However, none of the existing datasets consider answering questions that require cross-image query, i.e., answers that require information from both **multiple images** while **external knowledge** complementing the implicit relations among images. Thus, we build MVQA to fill this gap, assisting the evaluation of complex question answering in SVQA.

B. MVQA *Building Process*

Image Selection. The images are mainly extracted from the COCO datasets, which are widely used in VQA. To efficiently support our SVQA, we need to ensure that the images in MVQA have relevance. To do this, we first set the types of images as humans, animals, vehicles, and buildings, which have the highest proportion and crossover rate in COCO. We next manually filter out images that contain only a single object, so that the images in our dataset have potential relationships with others. In this way, 4,233 images are extracted from the COCO image pool (13,808 images). Compared with the other datasets in Table I, MVQA is suitable for labeling cross-image questions due to the visual complexity of images.

Generating Question-Answer Pairs. With all collected images from the COCO dataset, we manually generate questions and answers based on the image captions to reflect the implicit relationship among them. First of all, we need to define a **complex question**: a question that contains **multi-clauses** and **semantic complexity**. (1) *multi-clauses:* A complex question in the proposed VQA task is composed of multiple clauses. (2) *semantic complexity* refers to the complexity of a question that may require reasoning and understanding from multiple related images and external knowledge.

The questions in MVQA are generated in three steps. (1) First, we manually create many questions with each containing multiple objects from the entire image set. (2) Then, we filtered out questions that could be answered by a single image as associated objects may be contained in the same image. (3) Finally, we labeled the question-answer pairs along with the correct relevant images generated by three different annotators. To further increase the **semantic complexity**, we also modified the clause of the questions so that they would require external knowledge for reasoning.

The finally generated complex questions consist of three types: 1) the judgment question, 2) the counting question, and 3) the reasoning question. The first two types aim to judge or accumulate the interested relationships or objects in the images, but without the direct presents of objects or relationships. To answer such a question requires outside knowledge or other images' contents to reason the interested entities or relationships in the questions.

	Number of images	Knowledge based?	Cross images query?	Image source	Goal	Avg. Query length
DAQUR[40]	1,449	✗	✗	NYU-V2	visual: counts, colors, objects	11.5
Visual 7W[41]	47,300	✗	✗	COCO	visual: object-grounded queries	6.9
VQA(2.0)[18]	200K	✗	✗	COCO	visual understanding with commonsense	6.1
KB-VQA[42]	700	✓	✗	COCO	visual reasoning with given knowledge	6.8
FVQA[19]	2,190	✓	✗	COCO/ImageNet	visual reasoning with given knowledge	9.5
OK-VQA[34]	14,031	✓	✗	COCO	visual reasoning with open knowledge	8.1
MVQA (ours)	4,223	✓	✓	COCO	visual reasoning across images	16.9

TABLE I: Comparison of various VQA datasets.

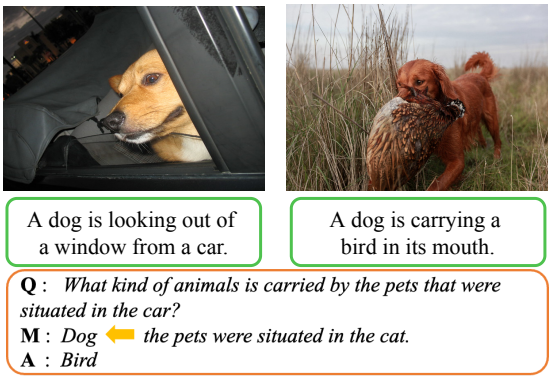


Fig. 7: The example of collected images and their corresponding question in MVQA have been shown. The text in the green box is a description of the image above, while the orange box is the complex query and corresponding answer.

	Questions	Clauses	SPOs	Average Images
Judgement	40	94	58	1593
Counting	16	35	28	2182
Reasoning	44	90	70	1201

TABLE II: The information of the MVQA

Example 7. Figure 7 shows an example of MVQA with generated question-answer pairs as well as two relevant pictures. The question *"What kind of animals is carried by the pets that were situated in the car?"* contains **two clauses**: (1) *"What kind of animals is carried by the pets?"* and (2) *"What pets were situated in the car?"*. These two simple questions can be answered by the two images (1) *"A dog is carrying a bird in its mouth."* and (2) *"A dog is looking out of a window from a car."* respectively. With proper reasoning, the middle answer for the clause "the pets were situated in the car." is "Dog", and the final answer for the complex question is "Dog".

C. MVQA *in Detail*

In our MVQA, the questions include an average of 2.2 clauses. Among them, there are 40 questions with constraints. In total, 100 questions with 219 clauses, in which there are a total of 136 unique subject-predicate-object (SPO).

The table II clearly demonstrates the differences within the dataset among three categories: 1) Judgement, 2) Counting, and 3) Reasoning. There are 40 instances of Judgement, 16 instances of Counting, and 44 instances of Reasoning. These complex questions correspond to 94, 35, and 90 clauses, respectively. Among these clauses, there are 58, 28, and 70 unique SPO triples for each category. It should be noted that the "unique" here refers to internal uniqueness within each category, and there may be some overlaps among the SPOs across different categories. However, the whole dataset contains 136 unique SPOs. It should be noted that some of these images require multiple visits to obtain the correct answer. On average, each complex question in the Judgement category requires at least 1593 images in the dataset to obtain the correct answer, while for the Counting category, the average number of images needed is 2182, and for the Reasoning category, it is 1201.

Discussion. Although our MVQA is a vanilla version dataset for the cross-images question answering, it is complicated enough for evaluating SVQA. Since answering each question requires reasoning multiple images out of the whole image base, which is a combinatorial explosion problem, making our handcrafted dataset (4233 images, 100 questions) hard enough for evaluation. Table I shows that our dataset has the longest question length where no existing method can be directly applied for answering. Also, the image selection process and question-answer generation process of MVQA ensure the diversity, generation, and complexity of the built dataset. In the future, we will apply the criteria to expand the dataset in various real-world cases.

VII. Evaluation

With the MVQA and modified VQAv2 [18], we experimentally evaluated the SVQA for its (1) accuracy on MVQA, (2)accuracy on modified VQAv2, (3) the impact of scene graph generation (SGG) on the accuracy, (4) the effectiveness of query parse, (5) the effectiveness of the caching mechanism.

Experimental Setting. We start with the settings.

Datasets We conduct our experiments on two datasets: MVQA and modified VQAv2. MVQA is only applicable for SVQA and the baseline methods fail to perform on this dataset. Therefore, we make the following modification for VQAv2 which is much simpler than MVQA but still requires to reason multiple images – 1) applying count questions to multiple images and asking the accumulated results of these questions; 2) combining two related simple questions into a complex question. As for the determination of the answer, we choose the top-1 response as the selected answer. When evaluating accuracy, judgment questions require a yes/no response, while counting questions necessitate a numerical answer. Regarding the reasoning questions, we employ cosine similarity to

Method	Latency(Sec.)	Judgment	Accuracy Counting	Reasoning
SVQA	10.38	90.0%	80.0%	87.5%

TABLE III: Experimental results of answering complex queries on the MVQA.

evaluate the answer by comparing semantic similarities. For instance, if the correct answer is "dog" and the provided answer is "puppy", we consider them to be consistent.

Hardware configuration We conduct our experiments on a Ubuntu 16.04 Linux server equipped with 8 Nvidia Tesla V100 GPU, 64 Intel Gold 5218 CPUs, and 512GB memory.

Baselines Most State-of-the-art baselines are constrained in the ability to answer complex questions, as they are limited by the fact that the input is a single image and the questions are based solely on that image. This makes them ill-suited for addressing more complex questions based on multiple images. To ensure a fair comparison, we modified the VQAv2 dataset as illustrated below to support the baseline methods to perform cross-image queries. To this end, we first utilize the SVQA' query graph generation module to generate a set of ordered simple questions. Then, the baseline methods perform the queries over the regrouped dataset with the decomposed questions and aggregate the obtained results.

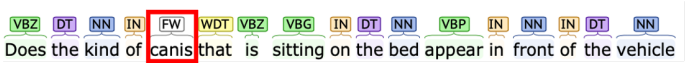
We choose three prominent VQA models that cover the three main tasks: (a) **dual-stream** (separate processing of visual and textual information): VisualBert [43] utilizes both visual embedding and BERT to adapt linguistic and visual input, using a dual-stream approach, **single-stream** (combined processing of visual and textual information): Vilt [6] is a single-stream approach, using a transformer to learn all visual and textual information, and **unified large-scale models** for handling all multimedia tasks: OFA [20] is a comprehensive unified sequence-to-sequence learning framework that addresses a wide range of multimedia tasks.

Exp-1: The Effectiveness of Answering Complex Query.

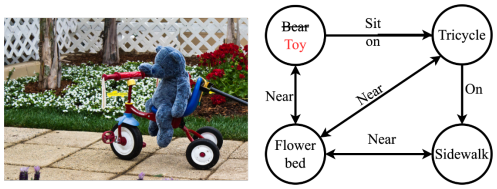
We measure the performance of SVQA on performing various types of complex queries, including judgment questions, counting questions, and reasoning questions (detailed in §VI).

Table III shows that the response time (latency) and the average accuracy are 10.38 seconds and 85.83%, respectively. Also, we observe that our system achieves the best accuracy on judgment questions and counting questions.

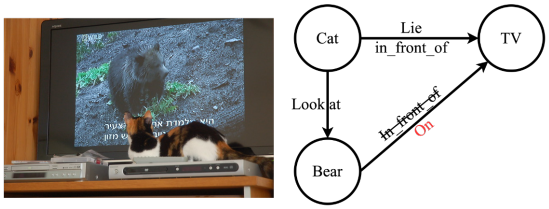
These differences are caused by the following steps: *Statement Parsing*, *Object Detection*, and *Relationship Generation*. Figure 8 shows examples of the causes of accuracy drops. Figure 8(a) demonstrates the query error caused by misunderstanding the question. Where the word "canis" marked with red box is parsed as a foreign word (FW), and SVQA fails to obtain a correct POS. The second reason that causes the query error is the erroneous recognition of an object. Figure 8(b) shows that a toy bear is recognized as a bear, resulting in an error in SGG. Lastly, the incorrect relationships between objects can also cause an error in SGG. Figure 8(c), for example, the bear should appear **on** the TV instead of appearing **in front of** the TV.



(a) The error of statement parsing



(b) The error of object detection



(c) The error of relationship generation

Fig. 8: Error analysis of SVQA

Method	Latency(Sec.)	Judgment	Accuracy Counting	Reasoning
Visual-Bert[43]	3375.56	72.0%	60.0%	68.5%
Vilt[6]	4216.34	76.5%	77.4%	67.0%
OFA[20]	866.36	95.5%	87.0%	79.0%
SVQA	10.38	93.0%	83.8%	83.2%

TABLE IV: Comparison of VQA on the modified VQAv2[18]

Exp-2: Performance Comparison Over Modified VQAv2.

Table IV shows that SVQA outperforms baseline models in response time (i.e., latency). Understandably, SVQA only needs to traverse the generated graph to obtain the required information for answering the questions. Unlike other baselines, which need to process each image and thus aggregate the related information to answer the questions. Also, the results show that OFA has better accuracy than that of SVQA in answering judgment questions and counting questions.

This is affected by the SGG model that is trained to describe a scene and the questions are not trained with the SGG model. The input for training the baseline models includes both questions and images, guided by a specified level of supervision in terms of visual information. Additionally, the OFA model is a large-scale model trained with plenty of data, and may be able to offer more fine-grained object recognition.

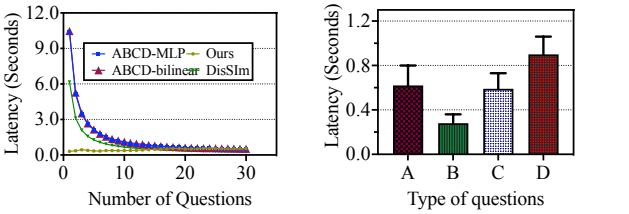
Exp-3: The Impact of Scene Graph Generation Method.

This experiment aims to investigate the impact of different methods for SGG on SVQA. Three frameworks, namely Neural Motifs, VCTree, and VTransE, are utilized for the generation of scene graphs. Additionally, the study examines the influence of bias in SGG. A comparison is made between the original models and modified models that employ causal inference, referred to as TDE (explained in §III-A).

Table 5 presents the accuracy of various models using different inference methods and their performance on complex queries within the SVQA system. The evaluation metric used for SGG is Mean Recall@K (mR@K). The results indicate a

Model	Method	SGG mR@20/50/100	Accuracy of SVQA (%)
VTransE[44]	Original	3.7 / 5.1 / 6.1	72.2
	TDE[24]	5.8 / 8.1 / 9.9	84.1
VCTree[45]	Original	4.2 / 5.8 / 6.9	74.1
	TDE	6.3 / 8.6 / 10.5	86.3
Nerual-Motifs[23]	Original	4.2 / 5.3 / 6.9	75.4
	TDE	6.9 / 9.5 / 11.3	87.2

TABLE V: Comparison of relation predication of the SGG.



(a) Latency of different methods (b) Latency of generating query graph for various questions

Fig. 9: For (a), we compare our methods with other deep learning-based sentence split methods. Figure 9(b) indicates the latency of different types of questions. *A* represents the average latency of all types of questions; *B*, *C* and *D* represent the questions containing a clause, 2 clauses, and 3 clauses.

positive correlation between model accuracy and overall system accuracy. Neural Motifs and VCTree outperform VTransE, with mR@K 20/50/100 averages exceeding 0.5/0.45/0.8. Consequently, the SVQA accuracy is expected to be, on average, 4.6% higher. Furthermore, when the TDE module is employed, the SVQA system achieves an average accuracy increase of 7.96% compared to the original method.

Exp-4: The Effectiveness of Query Parse.

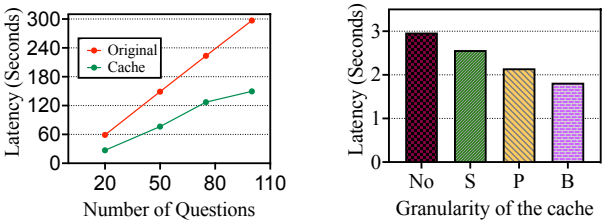
We compare our method with three baselines: ABCD-MLP [26], ABCD-bilinear [26] and DisSim [46]. Note that the baseline methods aim to split complex sentences and rephrase them into a set of clauses, which is step one of query graph generation (see §IV-B). Comparing the accuracy between them is not an apples-to-apples comparison. Thus, we compare the execution time of each method to demonstrate the efficiency of the query graph generation method.

Figure 9(a) shows that our method outperforms other deep learning-based solutions when the number of questions is small. But this advantage is less significant with the increase in the number of questions. This is because deep learning-based solutions require loading the trained models before processing the questions, which takes a considerable amount of time. Without considering the latency of model loading, our method has higher computational complexity. However, our method features high parallelization to further reduce the latency.

Figure 9(b) shows the latency of generating query graphs from different types of questions. The latency of generating a query graph increases with the complexity of the question, and the average latency is only 0.63s.

Exp-5: The Effectiveness of Caching Mechanism.

Lastly, we evaluate the efficiency of our caching mechanism. We measure the query latency with two different cache strategies and cache pool sizes. We measure the size of the cache



(a) Latency comparison between SVQA with cache and without cache

(b) Latency under different cache granularity on 100 questions, *No*, *S*, *P*, and *B* indicate the following: *No-cache*, *Scope*, *Path*, and *Both*. The size of the cache pool is set to 100.

Fig. 10: Effect of different cache policies on response time.

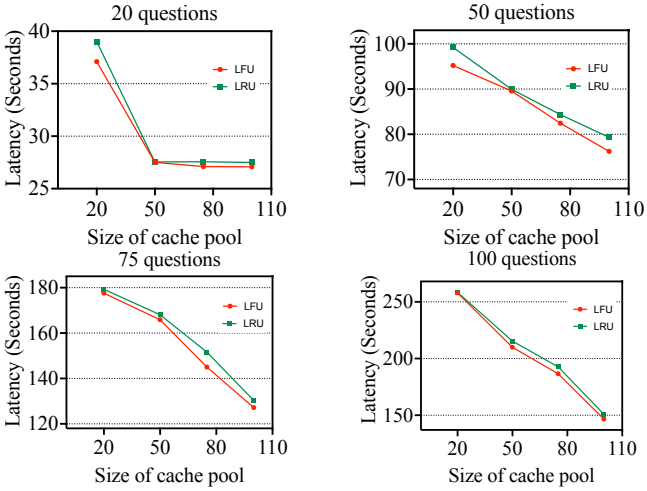


Fig. 11: Impact of cache pool size on latency.

pool by computing the number of items (i.e., scope items and path items), which is detailed in §V-B. In MVQA, the size of the scope item and path item is 6KB and 96KB respectively.

Figure 10(a) shows that our caching mechanism averagely reduces 48.89% latency, compared to the original SVQA without a caching module. With the increase in the number of questions, we see more improvements, where the latency is reduced to 49.67% that of the original one. Figure 10(b) exhibits the contributions of each component for reducing the latency. The figure illustrates that caching query scope and the path between the scope can reduce 13.46% and 27.61% latency, respectively, and the combined solution can reduce 38.72% latency, compared to the original solution.

Understandably, catching more information can achieve better performance, but consume more memory. The following discusses the impact of the size of the cache pool. Figure 11 illustrates the impact of cache size for latency while varying the number of questions. With 20 questions, after the size of the cache pool is greater than 50, increasing the size of the cache pool will not further reduce the latency. This is because all required information is cached, and the extra cache is not used. Additionally, the LFU [39] strategy achieves slightly better performance than that of LRU [47] in most cases. This is due to the distribution of questions that some *SPOC* has very high reuse rates but apart from different questions.

Summary. We find the following. (1) The SVQA excels at complex question reasoning, requiring multiple steps and interpretations to produce precise answers. It boasts a remark-

able average accuracy of 85.83% on the MVQA. In terms of efficiency, the execution time of SVQA is significantly lower compared to VisualBert, Vilt, and OFA. Specifically, SVQA, only takes 10.38s, accounting for 0.307%, 0.246%, and 1.197% of the respective execution times of VisualBert, Vilt, and OFA. Additionally, on the regroup VQAv2 dataset, SVQA outperforms VisualBert, Vilt, and OFA by 14.7%, 16.2%, and 4.2% respectively, as demonstrated in Exp-1 and Exp-2. (2) The impact of SGG on the overall SVQA has been validated through multiple experiments. SVQA exhibits a high level of modularity, allowing for individual modules to be updated independently. Enhancements made to the image extraction module have the potential to significantly improve the accuracy of the entire system. In Exp-3, although the mR@K shows a modest increase of 3.8%, the overall accuracy of the system experiences a significant improvement of 7.9%. (3) The language parse method significantly reduces the time of query graph generation. The experimental results in Exp-4 show our proposed method is approximately 10x faster than the baseline solutions. Additionally, reusing query elements can significantly reduce unnecessary overhead. By employing caching techniques, system latency can be reduced by approximately 150s for the same set of 100 queries, with each query being answered in about 1.2s, as shown in Exp-5.

VIII. Related Work

Entity Resolution. Entity resolution has been widely studied by the communities of computer science for relational data that is specified by a pattern [48]. The mainstream entity resolution systems employ the machine learning system to transfer to new tasks or exploit embedding of graph [49], [50] or JSON [51] for ER systems. Different from previous works, we study entity resolution across unstructured data images and structure data relations beyond relational data.

Entity resolution has also been studied for multi-models that jointly analyzing multi-modal descriptions [52], such as textual or image-based descriptions of the same entities. However, no previous methods work well across images $\mathbb{I}$ and relations G. They rely on the schema of the entities and consider only the edit distance and vertex description.

Visual Question Answering. It has seen significant advancements with various methods proposed for understanding both questions and images. These methods [3], [4], [5], [6] introduce joint embedding techniques that combine a Convolutional Neural Network or Vi-Transformer for image understanding, and a Recurrent Neural Network or Transformer (e.g., BERT) for question understanding. These have demonstrated promising performance, although the effectiveness may vary depending on the quality of the dataset. As part of our baseline for normal VQA, we include the Vilt method [6]. Now many pre-trained models or paradigms are proposed [43], [20], [53], trying to achieve the unity of multi-modalities with a modal-independent framework. These do achieve good accuracy, but the downstream task is limited by fine-tuning. As it achieves great performance on VQA, we include OFA and VisualBert among our baselines. Our SVQA system adopts an idea similar to the composition model, and divides VQA into Visual and Question Answering. For visual, we abstract it as Scene Graph. And for Question Answering, analyzing and understanding the query and abstracting it into query graphs. Because of the cross-image query, the question is necessarily complicated.

Regarding Question Answering, Natural Language Processing researchers have proposed various techniques for query understanding [27], [28], [54], [55] and conversion to query graphs [56], [26]. These techniques have proven effective in enhancing queries and query graphs. However, early studies primarily focused on questions with constraints and multi-hop relationship paths. More complex questions consisting of multiple clauses and constraints have not been well studied. These methods rely on training datasets and struggle to handle complex queries, which results in low-precision query graphs.

Large Models. In recent times, there have been numerous advancements in large language models, such as ChatGPT [11], Claude [12], and large vision models like SAM [14] and OVSeg [15]. Additionally, the emergence of large multi-modal models like SEEM [16] has further expanded the capabilities in processing both text and images. However, these models have limitations in terms of input constraints, such as processing only text, limited images, or specific multi-modal pairs.

To address the challenge of performing question-and-answer tasks while avoiding forgetting effects [57], our next step involves combining a formalized knowledge base, specifically a Knowledge Graph, with a parameterized knowledge base represented by a Large Language Model. Enabling us to overcome these limitations and leverage the powerful language understanding and image comprehension capabilities of these models. By incorporating both formalized knowledge and learned knowledge, we aim to achieve a deeper understanding and enhance our system's overall performance.

IX. Conclusion

In this paper, we propose SVQA, a novel framework for complex query answering. In order to assist the evaluation, we have constructed a new dataset called MVQA, which comprises many new types of complex <question, answer> pairs. As the questions may require extra information than provided to be correctly answered, we build a complete knowledge base with the help of a knowledge graph and complete the missing relationship between the images. We have also implemented an efficient pipeline for decomposing complex queries and performing queries in the knowledge base. Our evaluation results show that SVQA achieves 85.83% accuracy on the cross-image query answering.

Acknowledgment

Qi Xuan is the corresponding author. This work is supported by the National Key R&D Program of China (Grant No. 2022YFB2702100), the Key R&D Program of Zhejiang (Grant No. 2022C01018), the National Natural Science Foundation of China (Grant No. U21B2001), the NSFC (Grant Nos. 61932004, 62225203, U21A20516) and the DITDP (Grant No. JCKY2021211B017).

REFERENCES

[1] C. Li, Z. Miao, Q. Zeng, B. Glavic, and S. Roy, "Putting things into context: Rich explanations for query answers using join graphs (extended version)," *arXiv preprint arXiv:2103.15797*, 2021.

[2] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1986–1989, 2019.

[3] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1–9.

[4] B. Zhou, Y. Tian, S. Sukhbaatar, A. Szlam, and R. Fergus, "Simple baseline for visual question answering," *arXiv preprint arXiv:1512.02167*, 2015.

[5] M. Ren, R. Kiros, and R. Zemel, "Image question answering: A visual semantic embedding model and a new dataset," *Proc. Advances in Neural Inf. Process. Syst*, vol. 1, no. 2, p. 5, 2015.

[6] W. Kim, B. Son, and I. Kim, "Vilt: Vision-and-language transformer without convolution or region supervision," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5583–5594.

[7] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[8] M. Eberts and A. Ulges, "Span-based joint entity and relation extraction with transformer pre-training," *arXiv preprint arXiv:1909.07755*, 2019.

[9] K. Marino, M. Rastegari, A. Farhadi, and R. Mottaghi, "Ok-vqa: A visual question answering benchmark requiring external knowledge," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, 2019, pp. 3195–3204.

[10] N. Garcia, C. Ye, Z. Liu, Q. Hu, M. Otani, C. Chu, Y. Nakashima, and T. Mitamura, "A dataset and baselines for visual question answering on art," in *European Conference on Computer Vision*. Springer, 2020, pp. 92–108.

[11] OpenAI, "Gpt-4 technical report," 2023.

[12] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, C. Chen, C. Olsson, C. Olah, D. Hernandez, D. Drain, D. Ganguli, D. Li, E. Tran-Johnson, E. Perez, J. Kerr, J. Mueller, J. Ladish, J. Landau, K. Ndousse, K. Lukosuite, L. Lovitt, M. Sellitto, N. Elhage, N. Schiefer, N. Mercado, N. DasSarma, R. Lasenby, R. Larson, S. Ringer, S. Johnston, S. Kravec, S. E. Showk, S. Fort, T. Lanham, T. Telleen-Lawton, T. Conerly, T. Henighan, T. Hume, S. R. Bowman, Z. Hatfield-Dodds, B. Mann, D. Amodei, N. Joseph, S. McCandlish, T. Brown, and J. Kaplan, "Constitutional ai: Harmlessness from ai feedback," 2022.

[13] D. Castelvecchi, "Can we open the black box of ai?" *Nature News*, vol. 538, no. 7623, p. 20, 2016.

[14] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023.

[15] F. Liang, B. Wu, X. Dai, K. Li, Y. Zhao, H. Zhang, P. Zhang, P. Vajda, and D. Marculescu, "Open-vocabulary semantic segmentation with mask-adapted clip," 2023.

[16] X. Zou, J. Yang, H. Zhang, F. Li, L. Li, J. Gao, and Y. J. Lee, "Segment everything everywhere all at once," *arXiv preprint arXiv:2304.06718*, 2023.

[17] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual Question Answering," in *International Conference on Computer Vision (ICCV)*, 2015.

[18] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh, "Making the v in vqa matter: Elevating the role of image understanding in visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6904–6913.

[19] P. Wang, Q. Wu, C. Shen, A. van den Hengel, and A. R. Dick, "FVQA: fact-based visual question answering," *CoRR*, vol. abs/1606.05433, 2016. [Online]. Available: http://arxiv.org/abs/1606.05433

[20] P. Wang, A. Yang, R. Men, J. Lin, S. Bai, Z. Li, J. Ma, C. Zhou, J. Zhou, and H. Yang, "Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework," *arXiv preprint arXiv:2202.03052*, 2022.

[21] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3668–3678.

[22] F. Massa and R. Girshick, "maskrcnn-benchmark: Fast, modular reference implementation of instance segmentation and object detection algorithms in pytorch," 2018.

[23] R. Zellers, M. Yatskar, S. Thomson, and Y. Choi, "Neural motifs: Scene graph parsing with global context," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5831–5840.

[24] K. Tang, Y. Niu, J. Huang, J. Shi, and H. Zhang, "Unbiased scene graph generation from biased training," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 3716–3725.

[25] P. R. Asveld, "Generating all permutations by context-free grammars in greibach normal form," *Theoretical computer science*, vol. 409, no. 3, pp. 565–577, 2008.

[26] Y. Gao, T.-H. Huang, and R. J. Passonneau, "ABCD: A graph framework to convert complex sentences to a covering set of simple sentences," 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)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3919–3931. [Online]. Available: https://aclanthology.org/2021.acl-long.303

[27] S. Kübler, R. McDonald, and J. Nivre, "Dependency parsing," *Synthesis lectures on human language technologies*, vol. 1, no. 1, pp. 1–127, 2009.

[28] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 740–750.

[29] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2003, pp. 252–259.

[30] R. Huddleston, G. K. Pullum, and B. Reynolds, *A student's introduction to English grammar*. Cambridge University Press, 2021.

[31] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, jun 1993.

[32] J. Nivre, M.-C. de Marneffe, F. Ginter, J. Hajič, C. D. Manning, S. Pyysalo, S. Schuster, F. Tyers, and D. Zeman, "Universal dependencies v2: An evergrowing multilingual treebank collection," *arXiv preprint arXiv:2004.10643*, 2020.

[33] J. Nivre, M.-C. De Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira *et al.*, "Universal dependencies v1: A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, pp. 1659–1666.

[34] K. Marino, M. Rastegari, A. Farhadi, and R. Mottaghi, "OK-VQA: A visual question answering benchmark requiring external knowledge," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 3195–3204. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Marino_OK-VQA_A_Visual_Question_Answering_Benchmark_Requiring_External_Knowledge_CVPR_2019_paper.html

[35] K. Luo, F. Lin, X. Luo, and K. Zhu, "Knowledge base question answering via encoding of complex query graphs," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 2185–2194.

[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[37] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[38] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," in *The 7th international student conference on advanced science and technology ICAST*, vol. 4, no. 1, 2012, p. 1.

[39] S. Maffeis, "Cache management algorithms for flexible filesystems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 21, no. 2, pp. 16–25, 1993.

[40] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1–9.

[41] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei, "Visual7w: Grounded question answering in images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4995–5004.
[42] P. Wang, Q. Wu, C. Shen, A. v. d. Hengel, and A. Dick, "Explicit knowledge-based reasoning for visual question answering," *arXiv preprint arXiv:1511.02570*, 2015.
[43] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang, "Visualbert: A simple and performant baseline for vision and language," *arXiv preprint arXiv:1908.03557*, 2019.
[44] H. Zhang, Z. Kyaw, S.-F. Chang, and T.-S. Chua, "Visual translation embedding network for visual relation detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5532–5540.
[45] K. Tang, H. Zhang, B. Wu, W. Luo, and W. Liu, "Learning to compose dynamic tree structures for visual contexts," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 6619–6628.
[46] C. Niklaus, M. Cetto, A. Freitas, and S. Handschuh, "Dissim: A discourse-aware syntactic text simplification frameworkfor english and german," 2019. [Online]. Available: https://arxiv.org/abs/1909.12140
[47] E. J. O'neil, P. E. O'neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," *Acm Sigmod Record*, vol. 22, no. 2, pp. 297–306, 1993.
[48] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "End-to-end entity resolution for big data: A survey," *arXiv preprint arXiv:1905.06397*, 2019.
[49] C. Koutras, M. Fragkoulis, A. Katsifodimos, and C. Lofi, "Rema: Graph embeddings-based relational schema matching." in *EDBT/ICDT Workshops*, 2020.
[50] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1335–1349.
[51] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis, "The return of jedai: End-to-end entity resolution for structured and semi-structured data," *Proceedings of the VLDB Endowment, Vol. 11, No. 12*, vol. 11, no. 12, pp. 1950–1953, 2018.
[52] Z. Lin, Z. Zhang, M. Wang, Y. Shi, X. Wu, and Y. Zheng, "Multi-modal contrastive representation learning for entity alignment," *arXiv preprint arXiv:2209.00891*, 2022.
[53] W. Wang, H. Bao, L. Dong, and F. Wei, "Vlmo: Unified vision-language pre-training with mixture-of-modality-experts," *arXiv preprint arXiv:2111.02358*, 2021.
[54] P. Minervini, E. Arakelyan, D. Daza, and M. Cochez, "Complex query answering with neural link predictors," in *International Conference on Learning Representations*, 2021.
[55] J. Bai, Z. Wang, H. Zhang, and Y. Song, "Query2Particles: Knowledge graph reasoning with particle embeddings," in *Findings of the Association for Computational Linguistics: NAACL 2022*. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 2703–2714. [Online]. Available: https://aclanthology.org/2022.findings-naacl.207
[56] Z.-Y. Chen, C.-H. Chang, Y.-P. Chen, J. Nayak, and L.-W. Ku, "Uhop: An unrestricted-hop relation extraction framework for knowledge-based question answering," *arXiv preprint arXiv:1904.01246*, 2019.
[57] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," 2023.