RGVisNet: A Hybrid Retrieval-Generation Neural Framework Towards Automatic Data Visualization Generation

Yuanfeng Song
The Hong Kong University of Science and Technology & WeBank Co., Ltd
Hong Kong, China
songyf@cse.ust.hk

Xuefang Zhao
AI Group, WeBank Co., Ltd
Shenzhen, China
summerzhao@webank.com

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
Hong Kong, China
raywong@cse.ust.hk

Di Jiang
AI Group, WeBank Co., Ltd
Shenzhen, China
dijiang@webank.com

ABSTRACT

Recent years have witnessed the burgeoning of data visualization (DV) systems in both the research and the industrial communities since they provide vivid and powerful tools to convey the insights behind the massive data. A necessary step to visualize data is through creating suitable specifications in some declarative visualization languages (DVLs, e.g., Vega-Lite, ECharts). Due to the steep learning curve of mastering DVLs, automatically generating DVs via natural language questions, or text-to-vis, has been proposed and received great attention. However, existing neural network-based text-to-vis models, such as Seq2Vis or ncNet, usually generate DVs from scratch, limiting their performance due to the complex nature of this problem.

Inspired by how developers reuse previously validated source code snippets from code search engines or a large-scale codebase when they conduct software development, we provide a novel hybrid retrieval-generation framework named RGVisNet for text-to-vis. It retrieves the most relevant DV query candidate as a prototype from the DV query codebase, and then revises the prototype to generate the desired DV query. Specifically, the DV query retrieval model is a neural ranking model which employs a schema-aware encoder for the NL question, and a GNN-based DV query encoder to capture the structure information of a DV query. At the same time, the DV query revision model shares the same structure and parameters of the encoders, and employs a DV grammar-aware decoder to reuse the retrieved prototype. Experimental evaluation on the public *NVBench* dataset validates that RGVisNet can significantly outperform existing generative text-to-vis models such as ncNet, by up to 74.28% relative improvement in terms of overall accuracy. To the best of our knowledge, RGVisNet is the first framework that seamlessly integrates the retrieval- with the generative-based approach for the text-to-vis task.

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539330

CCS CONCEPTS

• **Human-centered computing** → **Visualization systems and tools**;

KEYWORDS

text-to-vis, hybrid retrieval-generation approach

ACM Reference Format:
Yuanfeng Song, Xuefang Zhao, Raymond Chi-Wing Wong, Di Jiang. 2022. RGVisNet: A Hybrid Retrieval-Generation Neural Framework Towards Automatic Data Visualization Generation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539330

1 INTRODUCTION

Living in the era of Big Data, almost all businesses in every domain have become data-driven, whether designing new products, making real-time decisions, or conducting social marketing activities. The value of *data visualizations* (DVs) is evident in these scenarios since they provide practical and powerful ways of summarizing insights and rules behind the massive data to support the final decisions. Thus, DV has attracted significant attention in the database [17, 25, 32] and the data mining communities [25, 28]. For example, [25] in KDD 2021 studied the problem of automatically recommending potential DVs given a massive dataset.

A necessary step to conduct DVs is through composing *visualization specifications* in some *declarative visualization languages* (DVLs), which specify *what* data are required and *how* the data would be visualized. In the community, there are already a substantial amount of DVLs such as Vega-Lite [27], ggplot2 [34], ZQL [30], ECharts [14], and VizQL [11], each of which enjoys a quite diversified grammar and syntax. Mastering DVs via composing suitable visualization specifications requires the users to have good knowledge of the domain data as well as expertise in these DVLs, a great challenge especially for beginners and non-technical users.

In response to the demand for lowering barriers to create DVs, a task named *Text-to-Vis* has been proposed to automatically translate natural language (NL) questions into DVs [17–19]. Since text-to-vis shows insights into unlocking the power of the relational database and visualization system to the users who have a limited technical

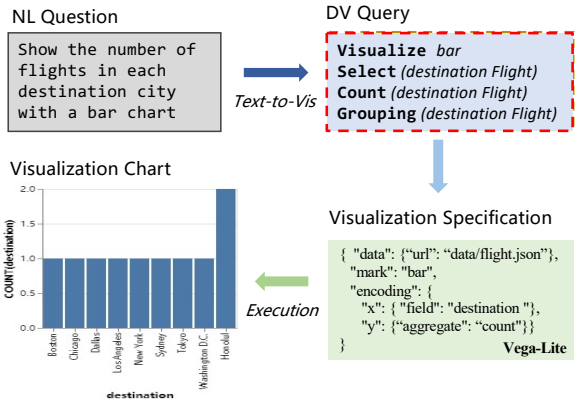


Figure 1: An example of the text-to-vis task, which first automatically translates the given NL question into its DV query, then gets the visualization specification (in Vega-Lite), and finally renders the corresponding DV Chart.

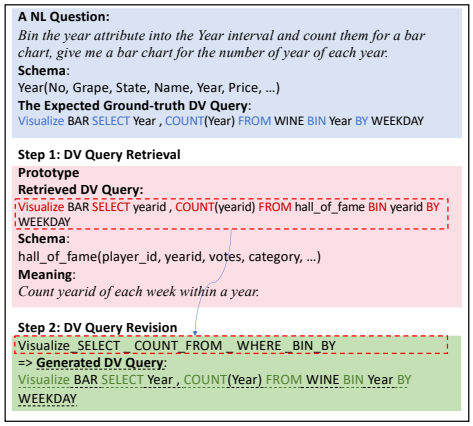


Figure 2: An illustrating example showing the pipeline of our hybrid Retrieval-Generation framework for text-to-vis.

background, it has become an essential task that researchers have extensively explored in recent years [7, 8, 18, 19, 24].

Indeed, automatically synthesizing DVs from the NL questions is quite challenging, since a suitable model should have the ability to understand the semantics expressed in the NL questions, and at the same time, compose the corresponding DVs according to their complex and strict grammar. Some studies have been attempted to deal with this problem. To name a few, NL4DV [24] supports to generate data visualization using NL questions, mainly based on NLP semantic parses. DeepEye [18] employs a rule-based methodology for creating DV charts. Seq2Vis [19] adopts a sequence-to-sequence neural network with an attention mechanism that could achieve end-to-end conversion of NL questions to *DV queries*, which follows a SQL-like grammar that defines the visualization details (e.g., chart type) and the data operations (e.g., aggregation, binning, filtering, sorting). These DV queries abstract and capture all the possible DVLs and can easily be converted into the corresponding visualization specifications in any DVLs to render the final DV chart. From an example in Figure 1, we can see that text-to-vis enables the user to query the DV system by simply asking an NL question *"Show the number of flights in each destination city with a bar chart"*, rather than directly composing a specification in some DVLs. The NL question is automatically translated into the corresponding DV query and then the specification in a DVL (i.e., Vega-Lite), and finally the rendered bar chart is shown to the user as the DV chart.

Despite all these efforts, existing text-to-vis studies still suffer a fundamental issue: *they employ a pure generative approach, i.e., synthesizing the DV from scratch.* None of them consider reusing the previously validated DVs that may fulfill similar functionality from a codebase. As such, we argue that the accuracies of these models can further be improved by involving related DVs as prototypes to avoid the complexity of DV generation from scratch. Actually, developers rarely implement code functionalities from scratch in daily practice. Instead, they usually reuse previous source code snippets by searching code search engines (e.g., Google Code Search[1], Github[2]) or a large-scale codebase, and then revise the code snippets into their desired functionality. In a parallel research field, dialogue system, the reuse of previous utterances to enhance the response generation task is also quite common, formally known as the *retrieval-based* natural language generation (NLG) [12]. While the *generative* NLG tends to produce highly coherent new responses, the retrieval-based approach is well-recognized preferring to produce more accurate, controllable, and diverse results [36]. An integration of both the retrieval- and the generation-based approach is proved to combine the merits of both methods in dialogue system [31, 36].

Inspired by the above-mentioned observations, we claim that it is worthwhile exploring the effectiveness of the retrieval-based approach on text-to-vis. We follow the common practice of Seq2Vis [19] and ncNet [20], which first translates NL questions into DV queries, and then convert these queries into specifications in any DVLs to get the final DV charts. Although it is somewhat promising, this approach is quite challenging in the text-to-vis scenario due to the following two reasons: (i) achieving accurate DV retrieval is quite hard since it is quite different from existing code search works for general-purpose programming languages (GPLs) such as Python or Java [9]; The structures between DV queries play a far more critical role in deciding the relevance than its semantics, while in code search for GPLs we usually only care about the semantics between the NL query and the code snippets. (ii) even when the model can retrieve accurate DV query candidates, these candidates still need significant revision and can not be directly used as DV query results. These two aspects make the hybrid approach in text-to-vis quite different from the common practice in other tasks, such as dialogue systems [31], where the retrieved utterances can directly be used as responses or only need little revision [31, 36].

To address the aforementioned challenges, in this work, we propose a hybrid retrieval-generation framework named **RGVisNet**, which aims to combine retrieval- and generation-based approaches to achieve more accurate text-to-vis conversion. As shown in Figure 4, RGVisNet works in a two-step pipeline analogy to the aforementioned developers' programming practices, that is: (i) retrieve the most relevant DV query concerning the given NL question from a codebase, and (ii) take the retrieved DV query as a prototype and revise it according to the specific requirement (schema and NL

[1]https://developers.google.com/code-search
[2]https://github.com

question) to generate the desired DV query. Specifically, for the first step, RGVisNet uses a novel structure-aware *DV query retrieval architecture*, which can employ a GNN-based structure to capture structural information as well as the semantic similarity simultaneously. For the second step, RGVisNet proposes a novel neural *DV query revision architecture* that adopts a DV grammar-aware decoder structure to generate the desired DV query. To further boost the performance, the two networks share a common schema-aware encoder for the NL question and the DV query.

To sum up, the main contributions of this work are threefold:

- To the best of our knowledge, we are the first to seamlessly integrate a retrieval- with a generation-based approach in automatic DV generation. We hope this work will inspire more research on combining the merits of both retrieval and the generation-based approach for other tasks.
- We propose a RGVisNet framework, which adopts an exact two-step pipeline analogy to developers' practice when conducting software development. The framework contains two novel networks - the DV query retrieval model and the DV query revision model, the former of which retrieves the most relevant candidates, while the latter of which revises the candidates to the desired DV query. Furthermore, the two networks share the designed NL question encoder and the DV query encoder.
- Extensive experimental evaluations show that the RGVisNet framework can significantly outperform existing pure generative text-to-vis models, such as Seq2Vis and ncNet, by up to a 74.28% improvement in terms of overall accuracy, proving the necessity to incorporate the previously validated similar DV query as a prototype.

The rest of this paper is organized as follows: we first introduce some concepts and the problem definition in Section 2. Then we discuss the details of our proposed RGVisNet framework, including the DV query retrieval network as well as the DV query Revision network in Section 3. The performance analysis is then presented in Section 4, followed by the related work in Section 5. Finally, we conclude the work in Section 6.

2 CONCEPTS AND PROBLEM DEFINITION

In this section, we first introduce some preliminary concepts that could improve the understanding of the following work and then give the formal definition of the text-to-vis problem.

Natural Language Question. An NL question is a human-understandable utterance describing the desired DV, and it is more user-friendly for users with limited DV background and programming skills to manipulate the data, especially when the desired DV is complex and esoteric.

Visualization Specification. Composing visualization specifications is an essential step to visualize the data as graphical charts. A visualization specification usually follows the grammar of a common *declarative visualization language* (DVL), specifying the details of the visualization construction (e.g., chart type, color, size, mapping function, properties for marks such as canvas size, legend, etc). Typical DVLs in the market include Vega-Lite [27], ggplot2 [34],

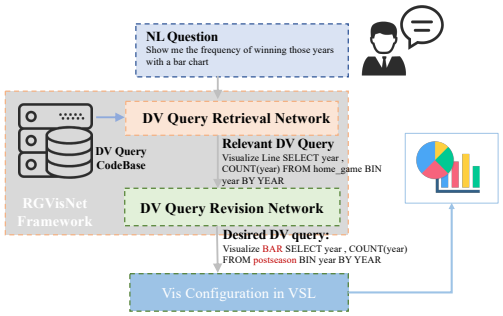


Figure 3: The working pipeline of the RGVisNet framework

ZQL [30], ECharts [14], and VizQL [11], each of which has its grammar. In Figure 1, a specification in Vega-Lite is given (i.e., the JSON object), defining attributes such as data path, mark, and encoding.

Data Visualization Query. The concept of the DV query is proposed by [17, 18], which aims to abstract all possible DVLs. Like a SQL query, a DV query is usually executed on a database to obtain the desired data. Furthermore, the DV query specifies additional visualization details to visualize these data. The corresponding DV query to the above-mentioned question is listed in Figure 1, in which the user defines the visualization chart to be a *"bar"* type with a SQL-like data manipulation operation *"Select · Count · Grouping ·"*. A DV query can easily be transformed into a visualization specification in any of these DVLs, and then the corresponding visualization engine will render the *DV Chart*. In Figure 1, the DV query is transformed into a specification in Vega-Lite as an example. However, it is trivial to transform the query into other DVLs such as ECharts.

Text-to-Vis. After defining the previous concepts, the definition of the *text-to-vis* problem is relatively straightforward, and it aims to translate the NL question into DVs (in terms of DV query). Mathematically, given an NL query $x = \{q, s\}$, where q ($q = \{q_i\}_{i=1}^{|q|}$) is an NL question describing the user's visualization need and s is the corresponding database schema, the text-to-vis task aims to synthesize the corresponding DV query y. Specifically, the schema s includes the collection of tables $T_x = \{t_i\}_{i=1}^{n_t}$, and the collection of columns $C_x = \{c_{i,j}\}_{j=1}^{L_i}$ for each table $t_i \in T_x$, where n_t is the number of tables in the schema and L_i is the number of columns in table t_i. Then, the complete training dataset can be represented as $\mathcal{D} = \{x^{(o)}, y^{(o)}\}_{o=1}^{N}$, where N is the dataset size. What is more, it should be noticed that the user's NL query and the schema are not restricted to one domain. The desired text-to-vis models should be able to generalize to unseen data sets in different domains (i.e., cross-domain inference).

3 THE RGVISNET FRAMEWORK

In this section, we are ready to describe the details of the proposed hybrid retrieval-generation framework - RGVisNet.

3.1 Framework Overview

As shown in Figure 3, RGVisNet takes the NL question and schema as input, retrieves the relevant DV query from the codebase that may fulfill the functionality, and adaptively generates the desired DV query. Mathematically, by employing a hybrid retrieval-generation approach, the RGVisNet framework decouples the text-to-vis task into two independent subtasks, which can be denoted as

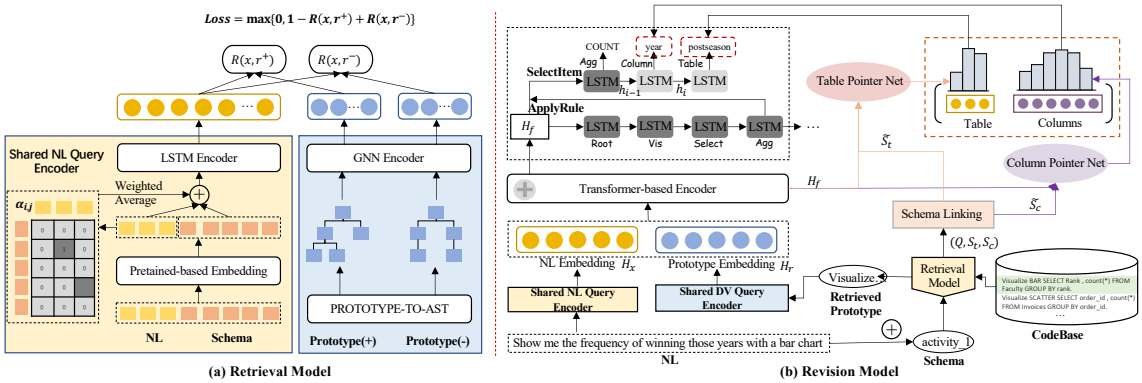


Figure 4: The Network Structure of the DV Query Retrieval and Revision Model in RGVisNet Framework

$$p(y|x) \propto \underbrace{p(r|x;\mathcal{B})}_{\text{DV Query Retrieval}} \cdot \underbrace{p(y|x,r)}_{\text{DV Query Revision}}, \tag{1}$$

where $\mathcal{B}$ is the codebase consisting of DV queries, r represents the prototype retrieved from the codebase with respect to the NL query x, and r also contains its own NL question and the corresponding schema, denoted as $r = \{x', y'\} = \{q', s', y'\}$. In this framework, $p(r|x;\mathcal{B})$ represents an *DV Query Retrieval Model*, aiming to recommend the most relevant DV query candidate from the codebase. $p(y|x,r)$ is an *DV Query Revision Model*, which generates the target DV query with the help of the recommended prototype. We will go through the details of the framework from these two aspects in the following sections.

3.2 DV Query Retrieval Model

As shown in Figure 4(a), our proposed DV query retrieval model mainly consists of a *Schema-aware NL Query Encoder*, a *GNN-based DV Query Encoder* and a *Similarity Module*.

3.2.1 Schema-aware NL Query Encoder. We first embed the NL question q with its DB schema s into embeddings, by serializing q and each table (including its table name and columns) in s. Each token q_i in the question q is first initialized to its embedding by a pre-trained GloVe vector. Different from the question, each item in schema s may consist of multiple words. Thus, for schema s, once we get an item from the collection of tables or columns, we first convert each word in the item into an embedding vector and then the average of these word embeddings is employed as the item embedding. Therefore, we could get the initialized embedding $\boldsymbol{Q} = \{\boldsymbol{q}_i\}_{i=1}^{|q|} \in \mathbb{R}^{|q|\times d_e}$ for the question and $\boldsymbol{S} = \{\boldsymbol{s}_i\}_{i=1}^{|s|} \in \mathbb{R}^{|s|\times d_e}$ for the schema, where d_e is the dimension of GloVe vector. Concretely, $\boldsymbol{S}$ can be further divided into table embedding $\boldsymbol{S}_t \in \mathbb{R}^{n_t\times d_e}$ and columns embedding $\boldsymbol{S}_c \in \mathbb{R}^{n_c\times d_e}$, which represent the embedding of the distinct table and column item respectively, and $n_c = \sum_{i=1}^{n_t} L_i$ indicates the total number of column items.

It is well-recognized that the NL question may contain mentioned keywords that appeared in its database schema, and explicitly identifying these references of columns, tables, and conditional value, also known as *schema linking*, would lead to more accurate synthesized SQL in the parallel text-to-SQL task [13, 38]. Based on the same motivation, we also explicitly consider the schema linking issue when generating the representations for the NL queries.

Specifically, luong-style attention is used to get the schema-aware representation $\boldsymbol{H}_x \in \mathbb{R}^{|q|\times d_m}$ for the NL query, where d_m is the hidden size of the used model. Firstly, we denote the correlation probability $\alpha_{i,j}$ of the token q_i with the schema element s_j as

$$e_{i,j} = \boldsymbol{q}_i^T \boldsymbol{W}_a \boldsymbol{s}_j, \tag{2}$$

$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{|s|} exp(e_{i,k})}, \tag{3}$$

where $1 \le i \le |q|$, $1 \le j \le |s|$ and $\boldsymbol{W}_a \in \mathbb{R}^{d_e\times d_e}$ is a learnable matrix. Then, $\boldsymbol{H}_x$ is learned through a bi-directional LSTM (BiLSTM) as

$$\boldsymbol{H}_x = \texttt{BiLSTM}(g(\boldsymbol{Q}, \boldsymbol{S}; \Theta)), \tag{4}$$

and a sentence-level embedding $\boldsymbol{h}_x \in \mathbb{R}^{d_m}$ for NL question is extracted by average pooling strategy, where $\Theta = \{e_{i,j}\}$ and $g(\cdot)$ function is a weighted average of word embeddings [13] as follows

$$g(\boldsymbol{Q}, \boldsymbol{S}; \Theta) = \{\mathbf{q}_i + \sum_{j=1}^{|s|} \alpha_{i,j} \cdot \boldsymbol{s}_j\}_{i=1}^{|q|}. \tag{5}$$

3.2.2 GNN-based DV Query Encoder. We also need to embed the DV query candidates from the codebase into embeddings. Note that a DV query already contains both its semantic and structural information, and thus, it suffices only to encode the query and exclude its corresponding schema. However, to preserve the structural as well as the semantic information, we first represent each query in the form of an abstract syntax tree (AST) [37]. In our implementation, we choose the extended grammar of SemQL [10, 19], as shown in Figure 5, to convert the DV queries into ASTs. Specifically, the tree is constructed recursively until reaching a leaf node, and each node contains its following grammar node and its attribute as child nodes. To use the DV query as a prototype for the following revision model, we only preserve its sketch information and ignore the intrusive specialized information, that is, the leaf nodes, including C (column) and T (table), are pruned from the built AST tree.

Then we generate each query's embedding using a graph neural network (GNN) structure [29]. Specifically, given a DV query (in the form of a AST) as a graph $G = (V, E)$, where V and E denote the nodes and edges, the representations $\boldsymbol{h}_v \in \mathbb{R}^{d_m}$ of each node $v \in V$ is constructed recursively by aggregating its neighbors $\mathcal{N}(v)$. The l-th layer of the GNN is calculated by

$$\boldsymbol{a}_v^{(l)} = f_{agg}^{(l)}\left(\left\{\boldsymbol{h}_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right), \tag{6}$$

$$\boldsymbol{h}_v^{(l)} = f_{cmd}^{(l)}\left(\boldsymbol{h}_v^{(l-1)}, \boldsymbol{a}_v^{(l)}\right), \tag{7}$$

where $f_{agg}(\cdot)$ and $f_{cmd}(\cdot)$ are the aggregate and combine operation. We initialize the representation of each node using a recurrent neural network. Given a DV query candidate r, we finally obtain its node level embedding $\boldsymbol{H}_r = \left\{\boldsymbol{h}_v^{(L)} : v \in V\right\} \in \mathbb{R}^{|V|\times d_m}$ and extract a graph level embedding $\boldsymbol{h}_r \in \mathbb{R}^{d_m}$ by aggregating the representations of all nodes in the final layer

$$\boldsymbol{h}_r = f_{out}\left(\boldsymbol{H}_r\right), \tag{8}$$

where L is the total number of layers in the GNN, and f_{out} is a graph-level pooling function.

3.2.3 Similarity and Model Training. After getting the embeddings $\boldsymbol{h}_x$ and $\boldsymbol{h}_r$ of the NL query x and the DV query candidate r, we use the cosine similarity to measure their relevance score R, defined as

$$R(x, r) = \cos(\boldsymbol{h}_x, \boldsymbol{h}_r) = \frac{\boldsymbol{h}_x^T \cdot \boldsymbol{h}_r}{\|\boldsymbol{h}_x\| \cdot \|\boldsymbol{h}_r\|}. \tag{9}$$

To train the whole retrieval network, we further employ a widely-used pairwise hinge loss function, which is defined as

$$\mathcal{L}_1(x, r^+, r^-) = \max\{0, 1 - R(x, r^+) + R(x, r^-)\}, \tag{10}$$

where the DV query candidate r^+ is more relevant than r^- with respect to the given NL query x.

3.3 DV Query Revision Model

As shown in Figure 4(b), the DV query revision model consists of four main components: a *Schema-aware NL Query Encoder*, a *GNN-based DV Query Encoder*, a *Transformer-based Encoder* and a *Grammar-aware Decoder*. Even though the input of the GNN-based DV query encoder is the most relevant DV queries from the retrieval model, the network still shares the same structure as the one proposed in the DV query retrieval model in our implementation. Hence, we directly employ the same structure of the DV query retrieval model and share their parameters. We mainly discuss the third and fourth components of the network in this section.

3.3.1 Transformer-based Encoder. To capture the relationship between the NL query and the DV prototype, a Transformer-based encoder is further employed, aiming to reinforce the correlated elements in both sequences and obtain fused embeddings as the inputs for the decoder. Our encoder is inspired by the famous Transformer structure [33], and it consists of a stack of Transformer blocks, each of which is composed of a multi-head attention mechanism, a fully connected feed-forward network, and a layer normalization. The module concatenates the NL embedding $\boldsymbol{H}_x$ and the DV embedding $\boldsymbol{H}_r$ as inputs, and outputs fused embedding $\boldsymbol{H}_f \in \mathbb{R}^{(|q|+|V|)\times d_m}$, shown as follows

$$\boldsymbol{H}_f = \texttt{Transformer}(\boldsymbol{H}_x \oplus \boldsymbol{H}_r), \tag{11}$$

where $\oplus$ denotes the concatenation operation, and $\texttt{Transformer}(\cdot)$ represents stacked Transformer-based encoder modules.

3.3.2 Vis Grammar-aware Decoder. Since DV query is a programming language with a concrete and strict grammar, it has been proved by a parallel task, text-to-SQL [10], that encoding the grammar information as prior knowledge is quite effective in guiding the code generation process. As such, we extend the basic SemSQL grammar to support the DV query, which is represented in Figure 5.

Then, we customize a popular grammar-aware neural structure in the related text-to-SQL task [10] as our decoder, which adopts an

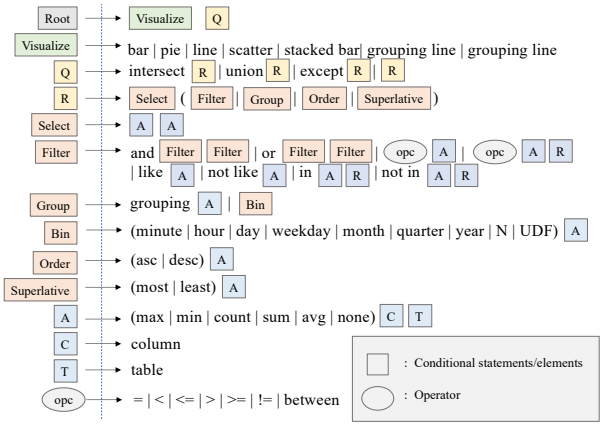


Figure 5: The Grammar for Converting a DV Query into an AST

LSTM-based structure to compose SemQL by selecting a sequence of actions. Mathematically, we could formalize the generation process of a SemQL DV query $\tilde{y}$ as

$$p(\tilde{y}|x, r) = \prod_{i=1}^{T} p(a_i|x, r, a_{<i}), \tag{12}$$

where a_i represents an action applied at step i, $a_{<i}$ denotes all the previous actions of step i, and T denotes the total number of the actions to get $\tilde{y}$. We further categorize the actions in Eq. (12) into two types (i) ApplyRule: utilizing a production rule to the current tree till finishing the DV query sketch. (ii) SelectItem: selecting an item from columns or tables to complete the SemQL DV query.

ApplyRule. We construct a context-free grammar tree with a method similar to [10], and in each step, we select the most probable branch given the previous route with an LSTM model. More specifically, at each step i, we update the LSTM state given previous state $\boldsymbol{h}_{i-1} \in \mathbb{R}^{d_m}$, previous action embedding $\boldsymbol{a}_{i-1} \in \mathbb{R}^{d_a}$, previous action type embedding $\boldsymbol{n}_{i-1} \in \mathbb{R}^{d_t}$ and previous context representation of LSTM $\boldsymbol{v}_{i-1}$, where d_a and d_t are the dimensionalities of the action embedding and the action type embedding, respectively. Then we calculate an attention context over the encoder time steps and score the production rule based on the softmax distribution as Eq. (16).

$$\boldsymbol{h}_i = \texttt{LSTM}([\boldsymbol{a}_{i-1} \oplus \boldsymbol{n}_{i-1} \oplus \boldsymbol{v}_{i-1}], \boldsymbol{h}_{i-1}), \tag{13}$$

$$\boldsymbol{v}_i = \texttt{Softmax}(\boldsymbol{h}_i^T \boldsymbol{W}_h \boldsymbol{H}_f^T)\boldsymbol{H}_f, \tag{14}$$

$$\boldsymbol{u}_i = \tanh(\boldsymbol{W}_u[\boldsymbol{h}_i \oplus \boldsymbol{v}_i] + \boldsymbol{b}_u), \tag{15}$$

$$p(\tilde{y}_i = a_i|x, s, a_{<i}) = \texttt{Softmax}(\tanh(\boldsymbol{W}_p \boldsymbol{u}_i + \boldsymbol{b}_p)), \tag{16}$$

where $\boldsymbol{W}_h \in \mathbb{R}^{d_m \times d_m}$, $\boldsymbol{W}_u \in \mathbb{R}^{d_m \times 2d_m}$ and $\boldsymbol{W}_p \in \mathbb{R}^{n_a \times d_m}$ (n_a is the number of actions correlated to the given grammar) are the learnable weights, $\boldsymbol{b}_u \in \mathbb{R}^{d_m}$ and $\boldsymbol{b}_p \in \mathbb{R}^{n_a}$ are learnable biases, and the initial state $\boldsymbol{h}_0$ is obtained by a average-pooling operation of the output $\boldsymbol{H}_f$ from the encoder.

SelectItem. To fill in the specific items contained in a DV query, an LSTM-based module is also employed. The main objective of this module is to decide which item (column or table) is involved in the text in the condition of the given schema. To find the item mentioned in the NL question, an NL-aware representation for the schema is also obtained. Let us take the column selection part as

Table 1: Performance Comparison.

Method	Validation Set				Test Set			
	Vis Acc.	Axis Acc.	Data Acc.	Acc.	Vis Acc.	Axis Acc.	Data Acc.	Acc.
Seq2Vis	97.10%	4.81%	24.39%	4.62%	97.85%	2.18%	11.39%	1.95%
Transformer	98.01%	4.81%	20.67%	4.62%	97.18%	3.26%	10.55%	2.76%
ncNet	99.09%	55.46%	63.37%	46.06%	98.82%	36.15%	50.89%	25.78%
RGVisNet	98.10%	69.72%	62.28%	**57.03%**	98.02%	63.51%	48.99%	**44.93%**

an example. Given the initial embeddings of a NL question and columns, denoted as $\boldsymbol{Q}$ and $\boldsymbol{S}_c$ in Section 3.2.1, we first perform attention mechanism over the question embedding for the columns, and then a joint representation $\tilde{\boldsymbol{S}}_{c_k} = \{\tilde{\boldsymbol{s}}_{c_k}\}_{c_k=1}^{n_c} \in \mathbb{R}^{n_c \times d_e}$ is calculated as follows

$$\beta_{k,j} = \frac{\boldsymbol{s}_{c_k}^T \boldsymbol{q}_j}{\|\boldsymbol{s}_{c_k}\| \left\|\boldsymbol{q}_j\right\|}, \tag{17}$$

$$\tilde{\boldsymbol{s}}_{c_k} = \boldsymbol{s}_{c_k} + \textstyle\sum_{j=1}^{|Q|} \beta_{k,j} \boldsymbol{q}_j, \tag{18}$$

where $\boldsymbol{s}_{c_k}$ is the embedding of column c_k and $\boldsymbol{q}_j$ is the embedding for j-th token in the NL question calculated in Section 3.2.1.

Finally, due to the variability in the schema of every prediction case, a column pointer net is also used to infer the probability of selecting a column item. The calculation of the selection probability for column c_k at step i is as

$$\gamma_{k,i} = (\tilde{\boldsymbol{s}}_{c_k})^T \boldsymbol{W}_c \boldsymbol{u}_i, \tag{19}$$

$$p(\tilde{y}_i = \text{SelectColumn}(c_k) | x, s, a_{<i}) = \frac{exp(\gamma_{k,i})}{\sum_{j=1}^{n_c} exp(\gamma_{j,i})}, \tag{20}$$

where SelectColumn$(\cdot)$ is the action implemented, and $\boldsymbol{W}_c \in \mathbb{R}^{d_e \times d_m}$ is a learnable weights. After the column item is confirmed, denoted as c_k, the procedure of SelectTable$(\cdot)$ works in a similar style except that the table candidates ($T(c_k)$) are restricted to the ones where the selected column corresponds to, that is, $T(c_k) = \{t_i | c_k \in C_x = \{c_{i,j}\}_{j=1}^{L_i}\}$. Then, we have

$$p(\tilde{y}_i = \text{SelectTable}(t_k) | x, s, a_{<i}) = \frac{exp(\gamma_{k,i})}{\sum_{t_k \in T(c_k)} exp(\gamma_{j,i})}, \tag{21}$$

3.3.3 Model Training. We train the revision model by maximizing the log-likelihood of the ground truth action sequences, defined as

$$\begin{aligned} \mathcal{L}_2 = \max \sum_{(x,s,y) \in \mathcal{D}} \Bigg[& \sum_{a_i \in \text{ApplyRule}} \log p(\tilde{y}_i = a_i | x, s, a_{<i}) \\ & + \sum_{a_i \in \text{SelectColumn}} \log p(\tilde{y}_i = a_i | x, s, a_{<i})) \\ & + \sum_{a_i \in \text{SelectTable}} \log p(\tilde{y}_i = a_i | x, s, a_{<i})) \Bigg]. \end{aligned} \tag{22}$$

We choose the teacher-forcing strategy to train the model with Adam optimizer. At the same time, the parsing is done in an autoregressive fashion until all the items have been filled as the termination condition.

4 EXPERIMENTS

This section provides a detailed performance evaluation of our proposed framework in terms of quantitative metrics. We first introduce the experimental setup, evaluation measurements, and baselines and then demonstrate the effectiveness of the proposed framework by comparing them with several strong baselines.

4.1 Experimental setup

4.1.1 Datasets. The public text-to-vis dataset *NVBench* [19] is used in our evaluation, which is composed of 7219 (NL question - DV query) pairs and is originally proposed for evaluating models conducting text-to-vis conversion. Since NVBench is modified based on a text-to-SQL dataset, the DV queries are from diverse domains, which also makes this dataset suitable for cross-domain evaluations. The detailed statistics of the NVBench is summarized in Table 4.

4.1.2 Baselines. Three carefully implemented baselines together with our proposed RGVisNet framework have been implemented in our experiment to compare the performance, namely, Seq2Vis [19], Transformer [33], and ncNet [20]. To ensure fairness and reproducibility, all the methods were trained on the same training set and evaluated on the same validation and testing set. We tune their parameters to achieve their best performance.

4.1.3 Evaluation Metrics. Five popular metrics [19], namely *Top-N (N=1,3,5), Vis Accuracy, Data Accuracy, Axis Accuracy, and Overall Accuracy*, are used in our experiment to evaluate the performance.

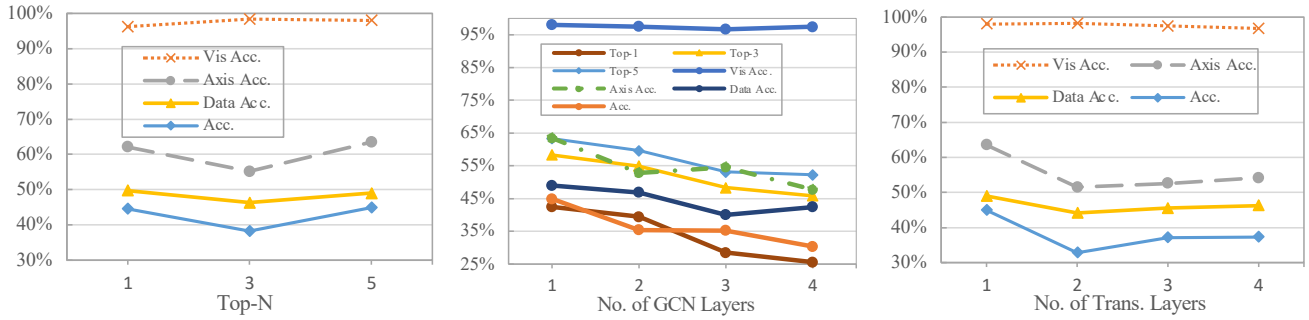
4.2 Experimental Results

4.2.1 Comparison of Accuracy. We list the performance of our proposed model together with the baselines on the NVBench dataset in Table 1, and several observations can be reported from the results.

The basic end-to-end approach, Seq2Vis, performs not very well and is not quite competitive as a baseline, with the main reasons from two aspects: (i) it cannot understand the semantics contained in the NL question and its database schema, (ii) it is limited in generating the DV queries since DV query is a programming language with strict and complex grammar. Other models such as Transformer and ncNet improve the basic Seq2Vis from these two aspects. For example, the ncNet also employs a GNN-based encoder to incorporate the schema information, which performs much better since it has been using GNN to capture the relationship in the database schema, resulting in 23.83% absolute query accuracy improvement. This also proves the necessity of involving schema as an essential information source. The ncNet also improves the Seq2Vis by optimizing the generation process by incorporating DV grammar. Among all these methods, our proposed method RGVisNet first retrieves a relevant DV query candidate from the codebase, and

Table 2: Ablation Study Results.

Method	Validation Set							Test Set						
	Top@1	Top@3	Top@5	Vis Acc.	Axis Acc.	Data Acc.	Acc.	Top@1	Top@3	Top@5	Vis Acc.	Axis Acc.	Data Acc.	Acc.
RGVisNet	51.31%	67.63%	72.44%	98.10%	69.72%	62.28%	57.03%	42.47%	58.30%	63.31%	98.02%	63.51%	48.99%	44.93%
w/o GNN	42.07%	60.02%	67.63%	97.91%	70.17%	58.93%	53.67%	37.16%	55.11%	61.76%	97.31%	54.03%	46.51%	35.62%
w/o retrieval	-	-	-	98.46%	65.46%	56.48%	49.95%	-	-	-	98.45%	44.62%	38.94%	26.28%
w. keyword retrieval	10.79%	15.32%	17.68%	97.19%	70.63%	56.39%	51.68%	16.77%	25.64%	31.49%	95.43%	61.16%	44.62%	38.61%
w/o grammar	51.31%	67.63%	72.44%	99.18%	4.99%	26.20%	4.71%	42.47%	58.30%	63.31%	98.45%	4.23%	12.74%	3.36%



(a) No. of Retrieval Prototypes (b) No. of GNN layers (c) No. of Transformer Layers

Figure 6: Hyper-parameters Study Result on the Testing Set.

uses it as a prototype to generate the desired DV query. Thus, it can significantly outperform all compared methods, including ncNet, proving the efficacy and the validity of involving a retrieval-based approach in the text-to-vis scenario.

4.2.2 Ablation Studies. In this section, we conduct ablation studies to show the effectiveness and contribution of each designed component in RGVisNet, especially the retrieval-based mechanism (i.e., the DV query search component). In particular, we first evaluate the RGVisNet with all the designed components as the baseline. Then we remove or replace some components of RGVisNet to check its performance. To evaluate the effectiveness of the DV query search component, we replace it with two combinations: (i) no DV query sampler (**w/o retrieval**), (ii) a basic keyword-based retrieval model (**w. basic retrieval**). We also remove some components and test the performance for each sub-model (i.e., the retrieval model and the revision model). Specifically, for the retrieval mode, we remove the GNN-based encoder and name it **w/o GNN**. Lastly, for the advanced decoder, we replace it with a basic LSTM-based one and name this baseline **w/o grammar**. The results are shown in Table 2.

We take the overall accuracy of the test set as the primary indicator, and the other metrics reflect similar conclusions. First, integrating the retrieval-based approach into our generation framework brings about 70.97% relative performance improvement (44.93% v.s 26.28%). These sets of ablation studies and significant improvements validate the necessity of involving the retrieval-based approach in the generation-based one for the DV generation task, which is never explored by existing literature. Then for each sub-model, compared with a basic keyword matching retrieval model, our designed DV query search model brings about 16.37% relative improvement, showing the effectiveness of using advanced NN-based models in understanding the semantics behind the DV query and the NL question. Other components show similar observations. For example, the GNN-based structure brings about 26.14% relative improvement, and the specially-designed DV decoder brings a 41.57% improvement.

4.2.3 Parameter Study. To study the influence of parameter variations on the performance of RGVisNet, we conduct another set of experiments. The main factor influencing the performance of RGVisNet is the number of layers in GNN, the number of layers in the Transformer, and the number of prototypes given to the revision model. We adjust one parameter and fix the rest each time. The experiment results are listed in Figure 6.

Again, we take the final accuracy on the test set as the primary indicator. The first set of results is about how the performance varies with respect to the number of retrieved prototypes, as shown in Figure 3(a). The performance reaches a low bound at number 3 and then increases when the number of prototypes overpasses or underpasses this optimal value. However, from the results in Figure 3(b), we can see that increasing the GCN layers does not always result in a better performance. This is consistent with previous studies on GNN [5], and it is because the model's learning capacity increases when we add the number of layers, but too many layers do not help due to the smoothing problem [15]. For another parameter - the number of Transformer layers, we could also get similar observations from Figure 3(d). After the number exceeds the value 1, the performance drops as the number increases.

4.2.4 Case Study. We also provide an example to concretely show the DV queries and their corresponding charts generated by the baselines and the RGVisNet framework in Table 3. Given the NL question, the Seq2Vis baseline misses a large part of the meaning contained in the question and only captures the visualization type (i.e., Bar chart). The Transformer performs better than the Seq2Vis model by capturing more essential keywords such as "*from high to low* (i.e., DESC)". Among all these models, the RGVisNet framework first accurately retrieves back the prototype "*VISUALIZE bar SELECT year, COUNT(year) FROM exhibition ORDER BY COUNT(year) DESC BIN year BY WEEKDAY*", and then generates the correct desired DV query. Furthermore, it accurately understands the meaning of "*x axis date y axis the number of dates*" in the NL question and

Table 3: The DV queries and the corresponding DV charts generated by different models on a NL question from the testing set.

NL Question	Bar chart x axis date y axis the number of date, could you display by the total number from high to low ?
Target DV Query	VISUALIZE bar SELECT date, COUNT(date) FROM weather ORDER BY COUNT(date) DESC BIN date BY WEEKDAY
Seq2Vis	VISUALIZE bar SELECT date, level_date FROM station ORDER BY COUNT(date) DESC BIN date BY YEAR
Transformer	VISUALIZE bar SELECT date, COUNT(date) FROM station ORDER BY COUNT(date) DESC BIN date BY WEEKDAY
ncNet	VISUALIZE bar SELECT date, COUNT(date) FROM weather GROUP BY date ORDER BY COUNT(date) DESC
RGVisNet	**Retrieved DV Query**:VISUALIZE bar SELECT year, COUNT(year) FROM exhibition ORDER BY COUNT(year) DESC BIN year BY WEEKDAY **Generated DV Query**: VISUALIZE bar SELECT date, COUNT(date) FROM weather ORDER BY COUNT(date) DESC BIN date BY WEEKDAY

(a) Seq2Vis(×)

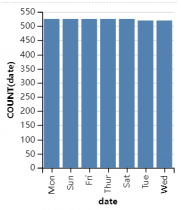
No image due to errors in the DV Query

(b) Transformer(×)

No image due to errors in the DV Query

(c) ncNet(×)

(d) RGVisNet(✓)



generates a corresponding DV query "*VISUALIZE bar SELECT date, COUNT(date)...*".

5 RELATED WORK

Our work is closely related to the research field of source code retrieval, text-to-vis, and the retrieval- and generation-based approaches in NLP, as is briefly surveyed in the following.

Source Code Search. The source code search problem correlates with our work since the performance of our framework is greatly affected by the code search step. In the software engineering field, source code search has quite a long history with countless studies even in these years [4, 9, 21, 41]. These studies usually focus on general-purpose programming languages (GPLs) such as Java or Python, and they usually incorporate advanced information retrieval (IR) and natural language processing (NLP) techniques in this field. To name a few, the Extended Boolean model is used in CodeHow [21] to conduct code retrieval for API recommendations. RACS [16] models the API calling relationships in the source code with call relationship (MCR) graphs. PageRank algorithms are used by Portfolio [22] on the Function Call Graph (FCG) in the source code to learn accurate code representations. However, these methods usually suffer the problem of limited ability to capture the hidden semantics. The most common practice in this field now usually employs advanced DNN-based architectures to learn the representations for both code and NL queries. For example, the CodeEE model is proposed in [9], which mainly uses the recurrent neural networks (RNNs) to convert code snippets and NL descriptions into vectors to calculate their similarities. FastText [2] is used in NCS [26] to get the embeddings for the query and code snippets. Cambronero et al. [3] explore the performance of various DNN-based models on this problem under different parameters. Convolutional neural networks (CNNs) with layer-wise attention are leveraged in COSEA [35] to improve the code search performance.

The *DV query search* problem can be considered as a particular case of the code search problem for GPLs. However, there are significant distinctions between code search works for DV queries and GPLs. These distinctions can be summarized into two aspects: (i) For DV query search, the user's query consists of NL questions describing the information needed and the database schema where the desired DV query would be executed. However, the latter does not exist in code search for GPLs. (ii) the structure of the DV query plays a far more critical role in deciding its relevance concerning the given NL question, while for GPLs we usually only care about the semantics. Thus, in our framework, we use the model specifically designed for the DV query search problem in our framework.

Text-to-Vis. Text-to-vis has engaged great attention from the database and visualization communities since it allows non-experts to interact with the visualization system with NL queries. Existing approaches treat this task as a machine translation problem and employ learning-based methods to handle it. To name a few, Cui *et al.* proposes text-to-viz which employs rule-based methods to transform text commands into infographics [7]. Draco-Learn [23] designs a collection of constraints to model the visualization design knowledge and then proposes an approach to optimize weights for these constraints. Data2Vis [8] also treats the DV generation task as a machine translation problem and aims to map the data series to visualization specifications in a declarative language. DataEye [17] handles the DV problem in three steps: visualization recognition, visualization ranking, and visualization selection. NL4DV [24] provides a python toolkit that supports various high-level operations to help users to create NL-based DV systems. To further promote the development of the text-to-vis field, Luo *et al.* further designed a method to generate the text-to-vis dataset NVBench based on a popular text-to-SQL benchmark. A Seq2Vis model is then developed on this dataset to prove the practicability of text-driven DV query generation on this benchmark [19].

Retrieval- and Generation-based Approaches. The retrieval- and generation-based approaches are two paradigms widely used in NLP applications such as dialogue systems, document summarization, and code generation. Let us take the dialogue system for illustration. The former approach usually selects the most relevant samples from a repository as the response, while the latter approach employs machine learning models to generate the response [12]. For the retrieval-based approach, the candidates in the repository usually accumulate from human dialogues, and thus, the retrieved responses are more diverse and informative than the generated responses. However, the size of the repository largely affects the final performance of this approach. In contrast, the generation-based approach could generate new responses given the query, but it also suffers from the problem like the dumb response (i.e., "I don't

know") [6].

Recently, the combination of retrieval-based and generation-based approaches has also been explored mainly in the NLP community to combine the merits of both methods. For example, Liu *et al.* [31] ensemble retrieval-based and generation-based approaches for the conversation system. Together with the query, the retrieved candidates are used to generate the response. Specifically, the retrieved candidates and the generated ones are fed into a re-ranking module to select the final reply. Yang *et al.* [36] propose a neural model to combine retrieval-based with the generation-based approaches for conversation systems.

Unfortunately, prior studies mainly focus on fields such as dialogue systems, and none of them have been dedicated to integrating retrieval- and generation-based approaches for text-to-vis. Compared with response generation in the dialogue system or other fields, DV query generation is more challenging since it is a programming language with strict and complex grammar. As such, the retrieved DV query can only be used as a prototype to guide the DV generation, and it needs much revision to achieve the final stage.

6 CONCLUSION

In this paper, we propose a novel hybrid retrieval-generation text-to-vis framework, RGVisNet, and prove its superiority compared with its baselines. RGVisNet is the first work in the literature that seamlessly integrates retrieval- and generation-based approaches together, aiming at achieving the merits of both retrieval and generation methods to generate DVs automatically. We hope it will show some insights into new methods for this text-to-vis task. Following this line of research, we would like to explore the hybrid retrieval-generation framework with the pre-training mechanism to bridge the gap between DV queries and NL questions.

Acknowledgement. The research of Yuanfeng Song and Raymond Chi-Wing Wong is supported by PRP/026/21FX.

REFERENCES

[1] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.
[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL* (2017).
[3] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *ESEC/FSE*.
[4] Wing-Kwan Chan, Hong Cheng, and David Lo. 2012. Searching connected API subgraph via text phrases. In *SIGSOFT*.
[5] Tianwen Chen and Raymond Chi-Wing Wong. 2020. Handling information loss of graph neural networks for session-based recommendation. In *SIGKDD*.
[6] Shaobo Cui, Rongzhong Lian, Di Jiang, Yuanfeng Song, Siqi Bao, and Yong Jiang. 2019. DAL: Dual Adversarial Learning for Dialogue Generation. In *NeuralGen Workshop in NAACL*.
[7] Weiwei Cui, Xiaoyu Zhang, Yun Wang, He Huang, Bei Chen, Lei Fang, Haidong Zhang, Jian-Guan Lou, and Dongmei Zhang. 2019. Text-to-viz: Automatic generation of infographics from proportion-related natural language statements. *TVCG* (2019).
[8] Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications* 39, 5 (2019), 33–46.
[9] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of ICSE*. IEEE, 933–944.
[10] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *ACL*.
[11] Pat Hanrahan. 2006. Vizql: a language for query, analysis and visualization. In *SIGMOD*.
[12] Kristiina Jokinen and Michael McTear. 2009. Spoken dialogue systems. *Synthesis Lectures on Human Language Technologies* 2, 1 (2009), 1–151.
[13] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *Proceedings of EMNLP*. 6943–6954.
[14] Deqing Li, Honghui Mei, Yi Shen, Shuang Su, Wenli Zhang, Junting Wang, Ming Zu, and Wei Chen. 2018. ECharts: a declarative framework for rapid construction of web-based visualization. *Visual Informatics* 2, 2 (2018), 136–146.
[15] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *ICCV*.
[16] Xuan Li, Zerui Wang, Qianxiang Wang, Shoumeng Yan, Tao Xie, and Hong Mei. 2016. Relationship-aware code search for JavaScript frameworks. In *SIGSOFT*.
[17] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. Deepeye: Towards automatic data visualization. In *ICDE*.
[18] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*.
[19] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *SIGMOD*.
[20] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2021. Natural Language to Visualization by Neural Machine Translation. *TVCG* (2021).
[21] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 2015. Codehow: Effective code search based on api understanding and extended boolean model (e). In *ASE*.
[22] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: finding relevant functions and their usage. In *ICSE*.
[23] Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *TVCG* (2018).
[24] Arpit Narechania, Arjun Srinivasan, and John Stasko. 2020. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *TVCG* (2020).
[25] Xin Qian, Ryan A Rossi, Fan Du, Sungchul Kim, Eunyee Koh, Sana Malik, Tak Yeon Lee, and Joel Chan. 2021. Learning to Recommend Visualizations from Data. In *SIGKDD*.
[26] Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: a neural code search. In *MAPL*.
[27] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *TVCG* (2016).
[28] Rafael Savvides, Andreas Henelius, Emilia Oikarinen, and Kai Puolamäki. 2019. Significance of patterns in data visualisations. In *SIGKDD*.
[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *TNN* (2008).
[30] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless Data Exploration with zenvisage: An Expressive and Interactive Visual Analytics System. *VLDB* (2016).
[31] Yiping Song, Cheng-Te Li, Jian-Yun Nie, Ming Zhang, Dongyan Zhao, and Rui Yan. 2018. An ensemble of retrieval-based and generation-based human-computer conversation systems. In *IJCAI*.
[32] Manasi Vartak, Silu Huang, Tarique Siddiqui, Samuel Madden, and Aditya Parameswaran. 2017. Towards visualization recommendation systems. *ACM SIGMOD Record* 45, 4 (2017), 34–39.
[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.
[34] Randle Aaron M Villanueva and Zhuo Job Chen. 2019. ggplot2: elegant graphics for data analysis.
[35] Hao Wang, Jia Zhang, Yingce Xia, Jiang Bian, Chao Zhang, and Tie-Yan Liu. 2020. COSEA: Convolutional Code Search with Layer-wise Attention. *ArXiv* (2020).
[36] Liu Yang, Junjie Hu, Minghui Qiu, Chen Qu, Jianfeng Gao, W Bruce Croft, Xiaodong Liu, Yelong Shen, and Jingjing Liu. 2019. A hybrid retrieval-generation neural conversation model. In *CIKM*.
[37] Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *ACL*.
[38] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In *NAACL*.
[39] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*.
[40] Albert Zeyer, Parnia Bahar, Kazuki Irie, Ralf Schlüter, and Hermann Ney. 2019. A comparison of Transformer and LSTM encoder decoder models for ASR. In *ASRU*.
[41] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *ICSE*.
[42] Xiangyu Zhao, Longbiao Wang, Ruifang He, Ting Yang, Jinxin Chang, and Ruifang Wang. 2020. Multiple knowledge syncretic transformer for natural dialogue generation. In *The Web Conference*.

APPENDIX

A EXPERIMENTAL SETUP

A.1 Datasets

The public text-to-vis dataset *NVBench* [19] is used in our evaluation, which is composed of 7219 (NL question - DV query) pairs and is originally proposed for evaluating models conducting text-to-vis conversion. Since NVBench is modified based on Spider [39] dataset, the DV queries are from diverse domains, which also makes this dataset suitable for cross-domain evaluations. We randomly split the 141 databases in NVBench into 98 train, 29 test, and 14 validation sets to ensure all questions from the same database are in the same split. The detailed statistics of the NVBench is summarized in Table 4.

The partitioned datasets are used in both the retrieval and the revision models. We first extract prototypes for all DV query in the training set and use these prototype to construct the codebase. Then in the training phase, five different prototypes are randomly selected as the negative samples for constructing the retrieval model, and top-k prototypes recalled by the retrieval model are incorporated into each revision instance. As for the testing phase, only the top-1 prototype retrieved is incorporated to infer the targeted DV query for each NL query.

A.2 Baselines

Three popular baselines together with our proposed RGVisNet framework have been implemented in our experiment to analyze the performance.

- Seq2Vis: Seq2Vis is proposed in [19], and it converts the text-to-vis problem into a machine translation problem. Then they directly use the sequence-to-sequence model [1] with an attention mechanism to tackle this problem.
- Transformer: Transformer [33] has been proved promising in many NLP tasks such as machine translation, dialogue system [42], and ASR [40]. We also employ the Transformer model as a baseline for performance comparison.
- ncNet: ncNet [20] is the previous state-of-the-art text-to-vis model that based on Transformer. However, they also include several novel visualization-aware optimizations, such as using attention-forcing to optimize the learning process and visualization-aware rendering to produce better visualization results.
- RGVisNet: RGVisNet is our proposed hybrid retrieval-generation framework for text-to-vis.

To ensure fairness and reproducibility, all the methods were trained on the same training set and evaluated on the same testing set. We tune their parameters to achieve their best performance.

A.3 Evaluation Metrics

- Top-N Accuracy: This metric corresponds to the number of relevant results among the top N retrieved results, where N is set to 1, 3, and 5. It is mainly used to analyze the quality of the DV query retrieval model.
- Overall Accuracy: This metric directly measures the matches between the predicted DV query and the ground truth DV query. The accuracy is calculated as $Acc = N_{dv}/N$, where N_{dv} is the number of the matched DV queries and N is the size of the evaluated set. Compared with other metrics, this one reflects the comprehensive performance of the models.
- Vis Accuracy: Since each DV query contains three kinds of components: vis type, x/y/z-axis, and data transformations. This measurement reflects the matches of the vis types components between the generated DV query and the ground truth query. The accuracy is formally defined as $Vis\ Acc = N_{vis}/N$, where N_{vis} is the number of vis type components matching the ground truth result.
- Data Accuracy: Similarly, this measurement reflects the matches of the data transformation components between the generated DV query and the ground truth query. The accuracy is formally defined as $Data\ Acc = N_{data}/N$, where N_{data} is the number of data transformation components matching the ground truth result.
- Axis Accuracy: This measurement calculates the matches of the x/y/z-axis components between the generated DV query and the ground truth query. The accuracy is formally defined as $Axis\ Acc = N_{axis}/N$, where N_{axis} is the number of x/y/z-axis components that match the ground truth result.

Table 4: The Statistics of the NVBench Dataset

Statistic	size
Train Size	11715
Test Size	2976
Valid Size	616
Number of Instances	7219
Average Length of the DV Query	20
Average Length of the NL Question	24
Vocabulary Size of the Description	1286

A.4 Implementation Details.

Our models are trained by Adam optimizer, with the mini-batch size set to 64 and the learning rate to 1e-4. The hyper-parameters are set following the previous studies, where the dropout is set to 0.3 to avoid overfitting and the hidden size of encoder and decoder are both set to 512. The NL encoder has one LSTM layer, and the number of GCN layers of the DV encoder is set to 1. As for the fused Transformer-based encoder, one block is stacked, and the number of heads is 4. The word embedding dimension of the pre-trained Glove is 300, and unseen words are initialized by '<unk>'. The dimensionalities of the action embedding and the type embedding in the grammar-aware decoder are set to 128. The inference in the revision model is conducted by beam search with a beam size equals to 5. The experiments were conducted on a server with a 314 GB memory, 72 Intel Core Processor (Xeon), Tesla K80 GPU, and CentOS. All the methods are implemented with Python 3.6.