

Supplemental Material for DeepVIS: Bridging Natural Language and Data Visualization Through Step-wise Reasoning

1 THE CONSTRUCTION OF NVBENCH-COT

1.1 Prompts for Generating CoT Steps

Below is the prompt we used to generating detailed CoT steps based on the ground truth VQL.

Please simulate a complete reasoning process to explain the Pre-entered Correct VQL, based on the provided Question and Database Schema below. Pretend that you are analyzing the Pre-entered Correct VQL for the first time and detail why each part of it is structured as such.

Question:
{question}
Database Schema:
{db_schema}
Pre-entered Correct VQL:
{VQL}

{additional_constraints}

Now, please reason according to the following strict format :

Step 1:

Reasoning for Chart Type: [Explain why the chart type in the Pre-entered Correct VQL is chosen based on the question and the database schema]

Chart Type: [Fill in the chart type from the Pre-entered Correct VQL here]

Step 2:

Reasoning for FROM: [Explain why the tables in the FROM clause of the Pre-entered Correct VQL are chosen based on the question and the database schema]

FROM: [Table names for the FROM clause from the Pre-entered Correct VQL]

Reasoning for SELECT: [Explain why the columns in the SELECT clause of the Pre-entered Correct VQL are chosen based on the question and the database schema]

SELECT: [Columns for the SELECT clause from the Pre-entered Correct VQL]

Reasoning for WHERE : [Explain why the conditions in the WHERE clause of the Pre-entered Correct VQL are set based on the question and the database schema]

WHERE: [Conditions for the WHERE clause from the Pre-entered Correct VQL]

Step 3:

Reasoning for GROUP BY: [Explain why the columns in the GROUP BY clause of the Pre-entered Correct VQL are used for grouping based on the question and the database schema]

GROUP BY: [Columns for the GROUP BY clause from the Pre-entered Correct VQL]

Reasoning for BIN: [If applicable, explain why the BIN_COLUMN and BIN_TIME_UNIT in the Pre-entered Correct VQL

are used based on the time - related grouping requirements and the database schema]

BIN: [BIN_COLUMN from the Pre-entered Correct VQL]

BY: [BIN_TIME_UNIT from the Pre-entered Correct VQL]

Step 4:

Reasoning for ORDER BY: [Explain why the columns in the ORDER BY clause of the Pre-entered Correct VQL are used for sorting based on the question and the database schema]

ORDER BY: [Columns for the ORDER BY clause from the Pre-entered Correct VQL]

Reasoning for SORT DIRECTION: [Based on the analysis of the question requirements and the database schema, explain why the ASC or DESC (or empty) in the Pre-entered Correct VQL is used for each column in the ORDER BY clause]

SORT DIRECTION: [ASC|DESC|empty from the Pre-entered Correct VQL]

Reasoning for LIMIT: [Explain why the LIMIT clause (if present) in the Pre-entered Correct VQL is set as such based on various factors]

LIMIT: [Number of rows for the LIMIT clause from the Pre-entered Correct VQL]

Step 5:

Final VQL: Combine the above content with the VQL template to obtain the final VQL result.

1.2 Constraints Used in the Reasoning Process

Below are the explicit constraints we include in the input prompt to guide the model to generate legal outputs.

During the reasoning process, you need to follow these constraints:

The VQL template is as follows:

Visualize [TYPE] SELECT [COLUMNS] FROM [TABLES] [WHERE] [GROUP BY] [ORDER BY] [SORT DIRECTION] [LIMIT] BIN [BIN_COLUMN] BY [BIN_TIME_UNIT]

The constraints for each part are as follows:

1. VISUALIZE field: Only BAR, PIE, LINE, and SCATTER are allowed as values. It is used to specify the data visualization chart type of the query results. Analyze the nature of the question and the database schema to explain why the [TYPE] in the Pre-entered Correct VQL is chosen.
2. SELECT field: There must be exactly two columns from the database in the SELECT clause. These columns must be either directly from the columns listed in the Database Schema or valid derivations based on those columns (COUNT, AVG, MAX and MIN). Explain why the [COLUMNS] in the Pre-entered Correct VQL are selected considering the question and the database schema.
3. FROM field: Only add the table names that must be used to complete the natural - language query. Evaluate the question and the database schema to explain why the [TABLES] in the Pre-entered Correct VQL are necessary to answer the question. Describe the connection between these tables and the data required by the question.
4. WHERE field: Analyze the natural - language query and the database schema to determine whether there is a filtering requirement. When there are clear limiting conditions, accurately explain why the WHERE clause logic expressions in the Pre-entered Correct VQL are formed this way, considering the conditional logical relationships to

match the query intent. The columns used in the conditions must be from the Database Schema or valid derivations.

5. Grouping:

If you need to group or aggregate the time column in the SELECT clause by a time unit (day, weekday, month, or year) and the current table does not have a column that meets this time - unit requirement, then use the BIN [BIN_COLUMN] BY [BIN_TIME_UNIT] part. [BIN_COLUMN] should be a time column from the Database Schema, and [BIN_TIME_UNIT] should be the appropriate time unit (day, weekday, month, or year).

When the dataset needs to be grouped and it is not related to time, add other columns in the GROUP BY clause as grouping bases in sequence, with earlier - used grouping bases placed more forward. The columns in the GROUP BY clause must be from the Database Schema or valid derivations. Explain why the GROUP BY and BIN (if applicable) parts in the Pre-entered Correct VQL are set up based on the question and the database schema.

6. LIMIT field: Add a LIMIT clause at the end of the VQL to specify the maximum number of rows to return. If there is no limit requirement, leave it empty. Explain why the LIMIT clause (if present) in the Pre-entered Correct VQL is included or not, considering factors such as the amount of data needed to answer the question and performance implications.

1.3 Some Exemplary Samples from nvBench-CoT

Below are three exemplary samples from the nvBench-CoT. The full dataset is available at <https://anonymous.4open.science/r/DeepVIS-9C33>.

1.3.1 Example 1

QUESTION

What are the different budget type codes, and how many documents are there for each? Order Budget_Type_Code in descending order.

DATABASE SCHEMA

- Table ref_document_types:
 - document_type_code: text
 - document_type_name: text
 - document_type_description: text
- Table ref_budget_codes:
 - budget_type_code: text
 - budget_type_description: text
- Table projects:
 - project_id: number
 - project_details: text
- Table documents:
 - document_id: number
 - document_type_code: text
 - project_id: number
 - document_date: time
 - document_name: text
 - document_description: text
 - other_details: text
- Table statements:
 - statement_id: number
 - statement_details: text
- Table documents_with_expenses:

- document_id: number
- budget_type_code: text
- document_details: text

• Table accounts:

- account_id: number
- statement_id: number
- account_details: text

STEP 1: REASONING FOR CHART TYPE

Reasoning for Chart Type: The user is interested in comparing different budget type codes alongside their respective document counts, which suggests that a bar chart would be suitable to visualize significant differences among categories. Bar charts are effective in displaying and comparing discrete categories due to their clarity in representing numerical values associated with each category label. In this instance, a bar chart will help to easily identify which budget types have more documents and how they compare with each other.

Chart Type: BAR

STEP 2: REASONING FOR FROM

Reasoning for FROM: To answer the question regarding budget type codes and the number of documents associated with each, the table documents_with_expenses is necessary as it contains links between documents and budget type codes. This table directly correlates budget types to document entries, which is essential for counting the number of documents per budget type. Other tables, like ref_budget_codes, could provide additional descriptions if needed, but the primary focus is on directly obtaining the counts from documents_with_expenses.

FROM: documents_with_expenses

Reasoning for SELECT: The columns selected in the query are Budget_Type_Code and COUNT(BUDGET_TYPE_CODE). The first column, Budget_Type_Code, is directly related to the budget types listed in the database schema, while the second column uses the aggregation function COUNT to tally the number of documents that correspond to each budget type code. This aligns perfectly with the question, which requires both the distinct budget types and the count of documents for those types.

SELECT: Budget_Type_Code, COUNT(BUDGET_TYPE_CODE)

Reasoning for WHERE: The natural language query does not specify any filtering conditions such as specific budget types or time frames for documents. Since the requirement is to group and count based on distinct budget type codes, and there are no restrictions present in the conditions of the question, the WHERE clause is left empty.

WHERE: [empty]

STEP 3: REASONING FOR GROUP BY

Reasoning for GROUP BY: The GROUP BY clause includes budget_type_code because we need to aggregate the document counts for each distinct budget type. Grouping by this column allows the query to summarize the data in the manner requested by the question—specifically, counting the number of documents for each budget type code.

GROUP BY: budget_type_code

Reasoning for BIN: In this scenario, there is no need for time-based grouping since the question is focused solely on budget types and document counts, without any mention of time periods. Therefore, the BIN section is not applicable here.

BIN: [empty]

BY: [empty]

STEP 4: REASONING FOR ORDER BY

Reasoning for ORDER BY: The ORDER BY clause is used to sort the results based on the Budget_Type_Code. This aligns with the requirement to display the results in descending order for the budget type codes, allowing for easier analysis of the types in the specified order.

ORDER BY: Budget_Type_Code

Reasoning for SORT DIRECTION: The question explicitly requests that the budget type codes be ordered in descending order. This implies that there is a need to see the budget types starting from the highest value down to the lowest, which guides us to use DESC as the appropriate sort direction.

SORT DIRECTION: DESC

Reasoning for LIMIT: There is no specified limit to the number of results required for this query. Since the question encompasses all distinct budget types and their document counts, it is reasonable to return all results without imposing a limit. Moreover, performance considerations are less critical here since the result set may be finite and manageable without extra constraints.

LIMIT: [empty]

STEP 5: FINAL VQL

Final VQL: Visualize BAR SELECT Budget_Type_Code, COUNT(Budget_Type_Code) FROM documents_with_expenses GROUP BY budget_type_code ORDER BY Budget_Type_Code DESC

1.3.2 Example 2

QUESTION

Show sum transaction amount from each transaction type , could you order total number of transaction amount in desc order ?

DATABASE SCHEMA

- Table customers:

- customer_id: number
- customer_first_name: text
- customer_middle_initial: text
- customer_last_name: text
- gender: text
- email_address: text
- login_name: text
- login_password: text
- phone_number: text
- town_city: text
- state_county_province: text
- country: text

- Table orders:

- order_id: number
- customer_id: number
- date_order_placed: time
- order_details: text

- Table invoices:

- invoice_number: number
- order_id: number
- invoice_date: time

- Table accounts:

- account_id: number
- customer_id: number
- date_account_opened: time
- account_name: text
- other_account_details: text

- Table product_categories:

- production_type_code: text

- product_type_description: text
- vat_rating: number

- Table products:

- product_id: number
- parent_product_id: number
- production_type_code: text
- unit_price: number
- product_name: text
- product_color: text
- product_size: text

- Table financial_transactions:

- transaction_id: number
- account_id: number
- invoice_number: number
- transaction_type: text
- transaction_date: time
- transaction_amount: number
- transaction_comment: text
- other_transaction_details: text

- Table order_items:

- order_item_id: number
- order_id: number
- product_id: number
- product_quantity: text
- other_order_item_details: text

- Table invoice_line_items:

- order_item_id: number
- invoice_number: number
- product_id: number
- product_title: text
- product_quantity: text
- product_price: number
- derived_product_cost: number
- derived_vat_payable: number
- derived_total_cost: number

STEP 1: REASONING FOR CHART TYPE

Reasoning for Chart Type: The question requires displaying a summary statistic (sum of transaction amounts) grouped by transaction type. Given that the objective is to represent these aggregated values visually to compare different transaction types, a bar chart is appropriate. A bar chart effectively represents relationships between categories and their respective values, allowing for easy comparison of the total transaction amounts across different transaction types.

Chart Type: BAR

STEP 2: REASONING FOR FROM

Reasoning for FROM: The main focus of the query is on transaction types and their amounts, which are located in the financial_transactions table. This table contains the relevant details about each transaction, such as transaction amounts and types, which are necessary to address the question. No other tables are required to supplement this information as all needed data points can be sourced directly from the financial_transactions table.

FROM: financial_transactions

Reasoning for SELECT: The SELECT clause includes transaction_type and the aggregated function SUM(transaction_amount). transaction_type is included to categorize the sum of transaction amounts. The aggregation function SUM(transaction_amount) directly corresponds to the request for the total amounts per transaction type, fulfilling the requirement outlined in the question.

SELECT: transaction_type , SUM(transaction_amount)

Reasoning for WHERE: The question does not specify any filtering criteria for transactions, nor does it imply that any conditions should be applied to the results based on the provided database schema. Therefore, there are no conditions needed in the WHERE clause, as all transaction types and their sums need to be included in the output without restriction.

WHERE: [empty]

STEP 3: REASONING FOR GROUP BY

Reasoning for GROUP BY: The GROUP BY clause is necessary to aggregate the transaction amounts by each transaction_type. This part is critical because without grouping, we would not be able to see the total transaction amount per type. The structure directly addresses the requirement to summarize the data based on the distinct transaction types present in the financial_transactions table.

GROUP BY: transaction_type

Reasoning for BIN: The query does not require time-based grouping or aggregation, as it purely focuses on summing transaction amounts by type. Since there is no time element involved in this analysis, we do not need to use a BIN clause at all.

BIN: [empty]

BY: [empty]

STEP 4: REASONING FOR ORDER BY

Reasoning for ORDER BY: In the ORDER BY clause, we utilize SUM(transaction_amount) to sort the results. This aligns with the question's request to order the total number of transaction amounts in descending order, allowing us to see which transaction types have the highest total amounts first.

ORDER BY: SUM(transaction_amount)

Reasoning for SORT DIRECTION: The requirement specified ordering in descending order, which is why DESC is used. This ensures that the transaction type with the highest summed amount appears at the top of the results, directly reflecting the question's intent.

SORT DIRECTION: DESC

Reasoning for LIMIT: There is no limit specified in the question, meaning we want to see all transaction types and their corresponding summed amounts. Therefore, no LIMIT clause is included in the query, as it would restrict the data unnecessarily and contradict the question's requirement to show the total amounts from each transaction type.

LIMIT: [empty]

STEP 5: FINAL VQL

Final VQL: Visualize BAR SELECT transaction_type , SUM(transaction_amount) FROM financial_transactions GROUP BY transaction_type ORDER BY SUM(transaction_amount) DESC

1.3.3 Example 3

QUESTION

A bar chart showing the average weight for each gender , I want to sort total number from high to low order .

DATABASE SCHEMA

- Table candidate:
 - candidate_id: number
 - people_id: number
 - poll_source: text
 - date: text

- support_rate: number
- consider_rate: number
- oppose_rate: number
- unsure_rate: number

- Table people:

- people_id: number
- sex: text
- name: text
- date_of_birth: text
- height: number
- weight: number

STEP 1: REASONING FOR CHART TYPE

Reasoning for Chart Type: The question specifies a need for a bar chart to display the average weight for each gender. Bar charts are well-suited for comparing categorical data, as they allow for easy visual interpretation of differences between categories. In this case, the categories are male and female corresponding to the sex column in the database schema. Hence, the chart type chosen aligns perfectly with the objective to visualize this comparison.

Chart Type: BAR

STEP 2: REASONING FOR FROM

Reasoning for FROM: The tables mentioned in the FROM clause need to contain the necessary data to fulfill the query. Here, the question focuses on average weight segmented by gender, which directly relates to the people table. This table contains the relevant columns (sex and weight) that are required to calculate the average weight based on gender. The candidate table is not relevant for the analysis requested about average weights and demographics.

FROM: people

Reasoning for SELECT: The SELECT clause must include columns that are pertinent to the analysis outlined in the question. The sex column identifies the gender categories, and the weight column is needed to compute the average weight for those categories. Therefore, these two columns are appropriately selected from the people table to provide the data required.

SELECT: Sex, AVG(WEIGHT)

Reasoning for WHERE: The question does not impose any specific filtering conditions; it simply requires an average based on gender. Since there are no restrictions or filters articulated in the question that would necessitate a WHERE clause, it is appropriate to exclude it in this context.

WHERE: [empty]

STEP 3: REASONING FOR GROUP BY

Reasoning for GROUP BY: The aim is to aggregate the average weight data by gender, so the sex column in the GROUP BY clause serves to group the results correctly. Since we are calculating the average weight per gender and there's no need for further grouping (time aggregation is irrelevant in this context), only the sex column is sufficient.

GROUP BY: sex

Reasoning for BIN: In this case, a BIN clause is not applicable since there is no time-related grouping needed—our aggregation is purely based on gender categories. Thus, there's no requirement for time bins in the analysis.

BIN: [empty]

BY: [empty]

STEP 4: REASONING FOR ORDER BY

Reasoning for ORDER BY: The question specifies that the results should be sorted based on the total average weight. Thus, it is logical to order by the AVG(WEIGHT), ensuring that the categories (genders) are aligned in decreasing order of their average weights for optimal visual analysis.

ORDER BY: AVG(WEIGHT)

Reasoning for SORT DIRECTION: The sorting direction DESC is chosen since the question explicitly requests to sort the total number of weights from high to low. This allows readers to quickly identify which gender has the larger average weight.

SORT DIRECTION: DESC

Reasoning for LIMIT: The question does not specify a limit on the number of rows to return, as it is likely interested in displaying both genders rather than limiting the output. Consequently, it is appropriate to leave the LIMIT clause out, indicating that all relevant data for both genders should be included in the results.

LIMIT: [empty]

STEP 5: FINAL VQL

Final VQL: Visualize BAR SELECT Sex , avg(weight) FROM people GROUP BY sex ORDER BY avg(weight) DESC

1.4 Prompt of the Inconsistency Check in Data Filtering

Below is the prompt we used to identifying 1,351 samples exhibiting inconsistency between queries and VQLs.

Determine whether the aggregation requirements in the Question align with the Aggregation Operations, and provide a detailed reasoning process.

Aggregation Operations:
GROUP BY:
{GROUP_BY_fields}
BIN BY:
BIN {BIN_BY_field} BY {BIN_TIME_UNIT}

Question:
{Question}

1.5 Detailed Statistics of nvBench-CoT

Table 1: Statistics of the nvBench-CoT Dataset

| Chart Type | Number of Samples | Avg. Question Length | Avg. CoT Length |
|------------|-------------------|----------------------|-----------------|
| BAR | 11254 (78.25%) | 25 | 566 |
| PIE | 1127 (7.84%) | 16 | 552 |
| SCATTER | 1098 (7.63%) | 18 | 557 |
| LINE | 904 (6.28%) | 27 | 582 |

2 THE FEW-SHOT PROMPT OF GENERAL PURPOSE LLMs

Given the Question and Database Schema, generate VQL based on the following example.

Input:
Question: [Question]
Database Schema: [Database Schema]

Example:
Question:
Visualize a bar chart showing the average age of captains in each class.
Database Schema

Table captain: [captain_id: number; name: text; ship_id: number; age: text; class: text; rank: text]
Table ship: [ship_id: number; name: text; type: text; built_year: number; class: text; flag: text]

Output:
Step 1:
Reasoning for Chart Type: The question asks to visualize the average age of captains according to their class. A bar chart is appropriate in this case because it allows for easy comparison of the average ages across different classes. Bar charts are particularly effective when comparing discrete categories, such as different classes of ships, which aligns well with the data structure provided.

Chart Type: BAR

Step 2:
Reasoning for FROM: The question specifically requires data regarding captains, which is inherently stored in the "captain" table. Since the average age is being calculated and categorized by class, the "ship" table is unnecessary.
FROM: captain
Reasoning for SELECT: The selected columns "Class" and "AVG (AGE)" directly match the query. "Class" allows grouping, and "AVG(AGE)" performs the required aggregation.
SELECT: Class, AVG(AGE)
Reasoning for WHERE: No specific filtering conditions are mentioned in the question.
WHERE: (empty)

Step 3:
Reasoning for GROUP BY: Required to compute the average age per class.
GROUP BY: CLASS
Reasoning for BIN: No temporal grouping is mentioned.
BIN: (empty)
BY: (empty)

Step 4:
Reasoning for ORDER BY: No sorting requirement mentioned.
ORDER BY: (empty)
Reasoning for SORT DIRECTION: No ORDER BY clause present.
SORT DIRECTION: (empty)
Reasoning for LIMIT: No limit requirement present.
LIMIT: (empty)

Step 5:
Final VQL: Visualize BAR SELECT Class, AVG(age) FROM captain GROUP BY CLASS

3 THE PROMPT OF INTERACTIVE REFINEMENT

3.1 The Self-Correction Prompt

Please correct the Initial Answer based on the Question and Database Schema. Note that the Self-correction Nodes Result identifies specific errors in the Initial Answer that need to be addressed. During the correction process, ensure that all related VQL components are updated consistently when making changes.

Question:
{Question}

Database Schema:
{Database Schema}

Initial Answer:
{Initial Answer}

Self-correction Nodes Result:
{Self-correction nodes result}

3.2 The Manual Correction Prompt

Please correct the Initial Answer based on the Question and Database Schema. Note that the Manual-correction Nodes Result identifies specific errors in the Initial Answer that need to be addressed incorporating the User Feedback. During the correction process, ensure that all related VQL components are updated consistently when making changes.

Question:
{Question}

Database Schema:
{Database Schema}

```
Initial Answer:
{Initial Answer}

Manual-correction Nodes Result:
{Manual-correction nodes result}

User Feedback:
{User Feedback}
```

4 IMPLEMENTATION DETAILS

4.1 Base Model Architecture

Our experiments were conducted using the Llama-3.1-8B-Instruct model, which is characterized by a Transformer-based architecture comprising 32 layers. Each layer in this model includes self-attention mechanisms and feed-forward networks, which are pivotal for processing sequential data and capturing long-range dependencies. The model’s total parameter count stands at 8 billion, allowing it to learn complex patterns and generate coherent text based on given instructions.

4.2 Hyperparameters

The hyperparameters for fine-tuning the llama-3.1-8B-Instruct model are listed in Table 2.

| Table 2: Hyperparameters | |
|-----------------------------|----------|
| Parameter | Value |
| Learning rate | 4e-5 |
| Learning Rate Scheduler | cosine |
| Batch size | 4 |
| Gradient Accumulation Steps | 8 |
| Number of epochs | 4 |
| Optimizer | adamw_hf |

4.3 Training Budget and Hardware Footprint

Training was conducted over 18.6 hours using a dataset of 11,260 samples. The hardware configuration for training consisted of an NVIDIA RTX 4090 GPU, which is equipped with 24 GB of graphics memory, ensuring efficient handling of the model’s parameters and the dataset.

5 VISUALIZATIONS GENERATED BY BASELINE METHODS AND OUR METHODS.

5.1 Example 1

Question: Draw a chart showing the ten majors with the highest numbers of students whose city of residence is known, with the counts tallied for each major.

Generated VQLs:

```
-- Seq2Vis:
VISUALIZE BAR SELECT MAJOR, COUNT(*) FROM STUDENT GROUP BY MAJOR

-- Transformer:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) DESC

-- ncNet:
VISUALIZE BAR SELECT MAJOR, COUNT(*) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY MAJOR DESC LIMIT 10

-- RGVisNet:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) ASC LIMIT 10

-- Llama3.1-8B:
```

```
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY MAJOR ASC LIMIT 10

-- Llama3.1-8B-SFT:
VISUALIZE LINE SELECT MAJOR, COUNT(*) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(*) DESC LIMIT 10

-- GPT-4o-mini:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) ASC LIMIT 10

-- GPT-o1:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT GROUP BY MAJOR ORDER BY COUNT(STUID) DESC LIMIT 10

-- GPT-o3:
VISUALIZE LINE SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) DESC LIMIT 10

-- Gemini-2.5-Pro:
VISUALIZE BAR SELECT MAJOR, COUNT(*) FROM STUDENT WHERE CITY_CODE != '' GROUP BY MAJOR ORDER BY COUNT(*) DESC LIMIT 10

-- Claude-3.7-Sonnet:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) LIMIT 10

-- DeepSeek-R1:
VISUALIZE LINE SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) DESC LIMIT 10

-- ChartGPT:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR

-- NL2VIS-CoT:
VISUALIZE BAR SELECT MAJOR, COUNT(STUID) FROM STUDENT WHERE CITY_CODE IS NOT NULL GROUP BY MAJOR ORDER BY COUNT(STUID) DESC LIMIT 10
```

5.2 Example 2

Question: Create a visual chart displaying the five most popular courses among students who have a recorded phone number, along with the number of students enrolled in each course, with the requirement that only courses with more than 100 students be shown in the chart.

Generated VQLs:

```
-- Seq2Vis:
VISUALIZE BAR SELECT COURSE_ID, COUNT(*) FROM InterestCourses GROUP BY COURSE_ID

-- Transformer:
VISUALIZE BAR SELECT COURSE_ID, COUNT(*) FROM InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY COURSE_ID ORDER BY COURSE_ID LIMIT 5

-- ncNet:
VISUALIZE BAR SELECT COURSE_ID, COUNT(*) FROM InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY COURSE_ID ORDER BY COUNT(STUID) DESC LIMIT 5

-- RGVisNet:
VISUALIZE BAR SELECT COURSE_ID, COUNT(*) AS STUDENT_COUNT FROM InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY COURSE_ID ORDER BY STUDENT_COUNT DESC

-- Llama3.1-8B:
```

```
VISUALIZE LINE SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses GROUP BY COURSE_ID ORDER BY COUNT(STUID)
DESC LIMIT 5

-- Llama3.1-8B-SFT:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses GROUP BY COURSE_ID ORDER BY COUNT(STUID)
DESC LIMIT 5

-- GPT-4o-mini:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER != '' GROUP BY COURSE_ID
ORDER BY COUNT(STUID) DESC LIMIT 5

-- GPT-o1:
VISUALIZE TABLE SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY
COURSE_ID ORDER BY COUNT(STUID) DESC LIMIT 5

-- GPT-o3:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY
COURSE_ID HAVING COUNT(STUID) > 100 ORDER BY COUNT(STUID)
LIMIT 5

-- Gemini-2.5-Pro:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER PHONE != '' GROUP BY
COURSE_ID HAVING COUNT(STUID) > 100 ORDER BY COUNT(STUID)
DESC LIMIT 5

-- Claude-3.7-Sonnet:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER PHONE != '' GROUP BY
COURSE_ID HAVING COUNT(STUID) > 100 ORDER BY COUNT(STUID)
DESC LIMIT 5

-- DeepSeek-R1:
VISUALIZE BAR SELECT COURSE_ID, COUNT(*) FROM
InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY
COURSE_ID HAVING COUNT(*) > 100 ORDER BY COUNT(STUID) DESC
LIMIT 5

-- ChartGPT:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses GROUP BY COURSE_ID ORDER BY COUNT(STUID)
DESC

-- NL2VIS-CoT:
VISUALIZE BAR SELECT COURSE_ID, COUNT(STUID) FROM
InterestCourses WHERE PHONE_NUMBER IS NOT NULL GROUP BY
COURSE_ID HAVING COUNT(STUID) > 100 ORDER BY COUNT(STUID)
DESC LIMIT 5
```